

---

# **Sage Reference Manual: Manifolds**

*Release 8.0*

**The Sage Development Team**

**Jul 23, 2017**



# CONTENTS

<b>1</b>	<b>Topological Manifolds</b>	<b>3</b>
1.1	Topological Manifolds	3
1.2	Subsets of Topological Manifolds	31
1.3	Manifold Structures	42
1.4	Points of Topological Manifolds	44
1.5	Coordinate Charts	58
1.5.1	Coordinate Charts	58
1.5.2	Coordinate Functions	92
1.5.3	Symbolic Coordinate Functions	99
1.6	Scalar Fields	116
1.6.1	Algebra of Scalar Fields	116
1.6.2	Scalar Fields	122
1.7	Continuous Maps	148
1.7.1	Sets of Morphisms between Topological Manifolds	148
1.7.2	Continuous Maps Between Topological Manifolds	150
<b>2</b>	<b>Differentiable Manifolds</b>	<b>175</b>
2.1	Differentiable Manifolds	175
2.2	Coordinate Charts on Differentiable Manifolds	210
2.3	The Real Line and Open Intervals	223
2.4	Scalar Fields	233
2.4.1	Algebra of Differentiable Scalar Fields	233
2.4.2	Differentiable Scalar Fields	239
2.5	Differentiable Maps and Curves	254
2.5.1	Sets of Morphisms between Differentiable Manifolds	254
2.5.2	Differentiable Maps between Differentiable Manifolds	258
2.5.3	Curves in Manifolds	269
2.6	Tangent Spaces	282
2.6.1	Tangent Spaces	282
2.6.2	Tangent Vectors	285
2.7	Vector Fields	294
2.7.1	Vector Field Modules	294
2.7.2	Vector Fields	312
2.7.3	Vector Frames	330
2.7.4	Group of Tangent-Space Automorphism Fields	345
2.7.5	Tangent-Space Automorphism Fields	351
2.8	Tensor Fields	359
2.8.1	Tensor Field Modules	359
2.8.2	Tensor Fields	365
2.8.3	Tensor Fields with Values on a Parallelizable Manifold	394

2.9	Differential Forms . . . . .	414
2.9.1	Differential Form Modules . . . . .	414
2.9.2	Differential Forms . . . . .	422
2.10	Affine Connections . . . . .	435
<b>3</b>	<b>Pseudo-Riemannian Manifolds</b>	<b>455</b>
3.1	Pseudo-Riemannian Metrics . . . . .	455
3.2	Levi-Civita Connections . . . . .	480
<b>4</b>	<b>Utilities for Calculus</b>	<b>487</b>
<b>5</b>	<b>Indices and Tables</b>	<b>495</b>
	<b>Python Module Index</b>	<b>497</b>
	<b>Index</b>	<b>499</b>

This is the Sage implementation of manifolds resulting from the [SageManifolds project](#). This section describes only the “manifold” part of SageManifolds; the pure algebraic part is described in the section Tensors on free modules of finite rank.

More documentation (in particular example worksheets) can be found [here](#).



## TOPOLOGICAL MANIFOLDS

### 1.1 Topological Manifolds

Given a topological field  $K$  (in most applications,  $K = \mathbf{R}$  or  $K = \mathbf{C}$ ) and a non-negative integer  $n$ , a *topological manifold of dimension  $n$  over  $K$*  is a topological space  $M$  such that

- $M$  is a Hausdorff space,
- $M$  is second countable,
- every point in  $M$  has a neighborhood homeomorphic to  $K^n$ .

Topological manifolds are implemented via the class `TopologicalManifold`. Open subsets of topological manifolds are also implemented via `TopologicalManifold`, since they are topological manifolds by themselves.

In the current setting, topological manifolds are mostly described by means of charts (see `Chart`).

`TopologicalManifold` serves as a base class for more specific manifold classes.

The user interface is provided by the generic function `Manifold()`, with with the argument `structure` set to `'topological'`.

#### Example 1: the 2-sphere as a topological manifold of dimension 2 over $\mathbf{R}$

One starts by declaring  $S^2$  as a 2-dimensional topological manifold:

```
sage: M = Manifold(2, 'S^2', structure='topological')
sage: M
2-dimensional topological manifold S^2
```

Since the base topological field has not been specified in the argument list of `Manifold`,  $\mathbf{R}$  is assumed:

```
sage: M.base_field()
Real Field with 53 bits of precision
sage: dim(M)
2
```

Let us consider the complement of a point, the “North pole” say; this is an open subset of  $S^2$ , which we call  $U$ :

```
sage: U = M.open_subset('U'); U
Open subset U of the 2-dimensional topological manifold S^2
```

A standard chart on  $U$  is provided by the stereographic projection from the North pole to the equatorial plane:

```
sage: stereoN.<x,y> = U.chart(); stereoN
Chart (U, (x, y))
```

Thanks to the operator  $\langle x, y \rangle$  on the left-hand side, the coordinates declared in a chart (here  $x$  and  $y$ ), are accessible by their names; they are Sage's symbolic variables:

```
sage: y
y
sage: type(y)
<type 'sage.symbolic.expression.Expression'>
```

The South pole is the point of coordinates  $(x, y) = (0, 0)$  in the above chart:

```
sage: S = U.point((0,0), chart=stereoN, name='S'); S
Point S on the 2-dimensional topological manifold S^2
```

Let us call  $V$  the open subset that is the complement of the South pole and let us introduce on it the chart induced by the stereographic projection from the South pole to the equatorial plane:

```
sage: V = M.open_subset('V'); V
Open subset V of the 2-dimensional topological manifold S^2
sage: stereoS.<u,v> = V.chart(); stereoS
Chart (V, (u, v))
```

The North pole is the point of coordinates  $(u, v) = (0, 0)$  in this chart:

```
sage: N = V.point((0,0), chart=stereoS, name='N'); N
Point N on the 2-dimensional topological manifold S^2
```

To fully construct the manifold, we declare that it is the union of  $U$  and  $V$ :

```
sage: M.declare_union(U,V)
```

and we provide the transition map between the charts  $\text{stereoN} = (U, (x, y))$  and  $\text{stereoS} = (V, (u, v))$ , denoting by  $W$  the intersection of  $U$  and  $V$  ( $W$  is the subset of  $U$  defined by  $x^2 + y^2 \neq 0$ , as well as the subset of  $V$  defined by  $u^2 + v^2 \neq 0$ ):

```
sage: stereoN_to_S = stereoN.transition_map(stereoS, [x/(x^2+y^2), y/(x^2+y^2)],
.....:                                     intersection_name='W', restrictions1= x^2+y^2!=0,
.....:                                     restrictions2= u^2+v^2!=0)
sage: stereoN_to_S
Change of coordinates from Chart (W, (x, y)) to Chart (W, (u, v))
sage: stereoN_to_S.display()
u = x/(x^2 + y^2)
v = y/(x^2 + y^2)
```

We give the name  $W$  to the Python variable representing  $W = U \cap V$ :

```
sage: W = U.intersection(V)
```

The inverse of the transition map is computed by the method `sage.manifolds.chart.CoordChange.inverse()`:

```
sage: stereoN_to_S.inverse()
Change of coordinates from Chart (W, (u, v)) to Chart (W, (x, y))
sage: stereoN_to_S.inverse().display()
```

```
x = u/(u^2 + v^2)
y = v/(u^2 + v^2)
```

At this stage, we have four open subsets on  $S^2$ :

```
sage: M.list_of_subsets()
[2-dimensional topological manifold S^2,
 Open subset U of the 2-dimensional topological manifold S^2,
 Open subset V of the 2-dimensional topological manifold S^2,
 Open subset W of the 2-dimensional topological manifold S^2]
```

$W$  is the open subset that is the complement of the two poles:

```
sage: N in W or S in W
False
```

The North pole lies in  $V$  and the South pole in  $U$ :

```
sage: N in V, N in U
(True, False)
sage: S in U, S in V
(True, False)
```

The manifold's (user) atlas contains four charts, two of them being restrictions of charts to a smaller domain:

```
sage: M.atlas()
[Chart (U, (x, y)), Chart (V, (u, v)),
 Chart (W, (x, y)), Chart (W, (u, v))]
```

Let us consider the point of coordinates  $(1, 2)$  in the chart `stereoN`:

```
sage: p = M.point((1,2), chart=stereoN, name='p'); p
Point p on the 2-dimensional topological manifold S^2
sage: p.parent()
2-dimensional topological manifold S^2
sage: p in W
True
```

The coordinates of  $p$  in the chart `stereoS` are computed by letting the chart act on the point:

```
sage: stereoS(p)
(1/5, 2/5)
```

Given the definition of  $p$ , we have of course:

```
sage: stereoN(p)
(1, 2)
```

Similarly:

```
sage: stereoS(N)
(0, 0)
sage: stereoN(S)
(0, 0)
```

A continuous map  $S^2 \rightarrow \mathbf{R}$  (scalar field):

```

sage: f = M.scalar_field({stereoN: atan(x^2+y^2), stereoS: pi/2-atan(u^2+v^2)},
....:                      name='f')
sage: f
Scalar field f on the 2-dimensional topological manifold S^2
sage: f.display()
f: S^2 --> R
on U: (x, y) |--> arctan(x^2 + y^2)
on V: (u, v) |--> 1/2*pi - arctan(u^2 + v^2)
sage: f(p)
arctan(5)
sage: f(N)
1/2*pi
sage: f(S)
0
sage: f.parent()
Algebra of scalar fields on the 2-dimensional topological manifold S^2
sage: f.parent().category()
Category of commutative algebras over Symbolic Ring

```

## Example 2: the Riemann sphere as a topological manifold of dimension 1 over $\mathbb{C}$

We declare the Riemann sphere  $\mathbb{C}^*$  as a 1-dimensional topological manifold over  $\mathbb{C}$ :

```

sage: M = Manifold(1, 'C*', structure='topological', field='complex'); M
Complex 1-dimensional topological manifold C*

```

We introduce a first open subset, which is actually  $\mathbb{C} = \mathbb{C}^* \setminus \{\infty\}$  if we interpret  $\mathbb{C}^*$  as the Alexandroff one-point compactification of  $\mathbb{C}$ :

```

sage: U = M.open_subset('U')

```

A natural chart on  $U$  is then nothing but the identity map of  $\mathbb{C}$ , hence we denote the associated coordinate by  $z$ :

```

sage: Z.<z> = U.chart()

```

The origin of the complex plane is the point of coordinate  $z = 0$ :

```

sage: O = U.point((0,), chart=Z, name='O'); O
Point O on the Complex 1-dimensional topological manifold C*

```

Another open subset of  $\mathbb{C}^*$  is  $V = \mathbb{C}^* \setminus \{O\}$ :

```

sage: V = M.open_subset('V')

```

We define a chart on  $V$  such that the point at infinity is the point of coordinate 0 in this chart:

```

sage: W.<w> = V.chart(); W
Chart (V, (w,))
sage: inf = M.point((0,), chart=W, name='inf', latex_name=r'\infty')
sage: inf
Point inf on the Complex 1-dimensional topological manifold C*

```

To fully construct the Riemann sphere, we declare that it is the union of  $U$  and  $V$ :

```

sage: M.declare_union(U,V)

```

and we provide the transition map between the two charts as  $w = 1/z$  on  $A = U \cap V$ :

```
sage: Z_to_W = Z.transition_map(W, 1/z, intersection_name='A',
....:                          restrictions1= z!=0, restrictions2= w!=0)
sage: Z_to_W
Change of coordinates from Chart (A, (z,)) to Chart (A, (w,))
sage: Z_to_W.display()
w = 1/z
sage: Z_to_W.inverse()
Change of coordinates from Chart (A, (w,)) to Chart (A, (z,))
sage: Z_to_W.inverse().display()
z = 1/w
```

Let consider the complex number  $i$  as a point of the Riemann sphere:

```
sage: i = M((I,), chart=Z, name='i'); i
Point i on the Complex 1-dimensional topological manifold C*
```

Its coordinates w.r.t. the charts Z and W are:

```
sage: Z(i)
(I,)
sage: W(i)
(-I,)
```

and we have:

```
sage: i in U
True
sage: i in V
True
```

The following subsets and charts have been defined:

```
sage: M.list_of_subsets()
[Open subset A of the Complex 1-dimensional topological manifold C*,
Complex 1-dimensional topological manifold C*,
Open subset U of the Complex 1-dimensional topological manifold C*,
Open subset V of the Complex 1-dimensional topological manifold C*]
sage: M.atlas()
[Chart (U, (z,)), Chart (V, (w,)), Chart (A, (z,)), Chart (A, (w,))]
```

A constant map  $\mathbf{C}^* \rightarrow \mathbf{C}$ :

```
sage: f = M.constant_scalar_field(3+2*I, name='f'); f
Scalar field f on the Complex 1-dimensional topological manifold C*
sage: f.display()
f: C* --> C
on U: z |--> 2*I + 3
on V: w |--> 2*I + 3
sage: f(0)
2*I + 3
sage: f(i)
2*I + 3
sage: f(inf)
2*I + 3
sage: f.parent()
Algebra of scalar fields on the Complex 1-dimensional topological
```

```

manifold C*
sage: f.parent().category()
Category of commutative algebras over Symbolic Ring

```

## AUTHORS:

- Eric Gourgoulhon (2015): initial version
- Travis Scrimshaw (2015): structure described via *TopologicalStructure* or *RealTopologicalStructure*

## REFERENCES:

- [Lee2011]
- [Lee2013]
- [KN1963]
- [Huy2005]

```

sage.manifolds.manifold.Manifold(dim, name, latex_name=None, field='real', structure='smooth', start_index=0, **extra_kwds)

```

Construct a manifold of a given type over a topological field  $K$ .

Given a topological field  $K$  (in most applications,  $K = \mathbf{R}$  or  $K = \mathbf{C}$ ) and a non-negative integer  $n$ , a *topological manifold of dimension  $n$  over  $K$*  is a topological space  $M$  such that

- $M$  is a Hausdorff space,
- $M$  is second countable, and
- every point in  $M$  has a neighborhood homeomorphic to  $K^n$ .

A *real manifold* is a manifold over  $\mathbf{R}$ . A *differentiable* (resp. *smooth*, resp. *analytic*) is a real manifold such that all transition maps are *differentiable* (resp. *smooth*, resp. *analytic*).

## INPUT:

- `dim` – positive integer; dimension of the manifold
- `name` – string; name (symbol) given to the manifold
- `latex_name` – (default: None) string; LaTeX symbol to denote the manifold; if none are provided, it is set to `name`
- `field` – (default: 'real') field  $K$  on which the manifold is defined; allowed values are
  - 'real' or an object of type `RealField` (e.g. `RR`) for a manifold over  $\mathbf{R}$
  - 'complex' or an object of type `ComplexField` (e.g. `CC`) for a manifold over  $\mathbf{C}$
  - an object in the category of topological fields (see `Fields` and `TopologicalSpaces`) for other types of manifolds
- `structure` – (default: 'smooth') to specify the structure or type of manifold; allowed values are
  - 'topological' or 'top' for a topological manifold
  - 'differentiable' or 'diff' for a differentiable manifold
  - 'smooth' for a smooth manifold
  - 'analytic' for an analytic manifold
- `start_index` – (default: 0) integer; lower value of the range of indices used for “indexed objects” on the manifold, e.g. coordinates in a chart

- `extra_kwds` – keywords meaningful only for some specific types of manifolds

OUTPUT:

- a manifold of the specified type, as an instance of *TopologicalManifold* or one of its subclasses, e.g. *DifferentiableManifold*

EXAMPLES:

A 3-dimensional real topological manifold:

```
sage: M = Manifold(3, 'M', structure='topological'); M
3-dimensional topological manifold M
```

Given the default value of the parameter `field`, the above is equivalent to:

```
sage: M = Manifold(3, 'M', structure='topological', field='real'); M
3-dimensional topological manifold M
```

A complex topological manifold:

```
sage: M = Manifold(3, 'M', structure='topological', field='complex'); M
Complex 3-dimensional topological manifold M
```

A topological manifold over  $\mathbb{Q}$ :

```
sage: M = Manifold(3, 'M', structure='topological', field=QQ); M
3-dimensional topological manifold M over the Rational Field
```

A 3-dimensional real differentiable manifold of class  $C^4$ :

```
sage: M = Manifold(3, 'M', field='real', structure='differentiable',
....:               diff_degree=4); M
3-dimensional differentiable manifold M
```

Since the default value of the parameter `field` is `'real'`, the above is equivalent to:

```
sage: M = Manifold(3, 'M', structure='differentiable', diff_degree=4)
sage: M
3-dimensional differentiable manifold M
sage: M.base_field_type()
'real'
```

A 3-dimensional real smooth manifold:

```
sage: M = Manifold(3, 'M', structure='differentiable', diff_degree=+oo)
sage: M
3-dimensional differentiable manifold M
```

Instead of `structure='differentiable', diff_degree=+oo`, it suffices to use `structure='smooth'` to get the same result:

```
sage: M = Manifold(3, 'M', structure='smooth'); M
3-dimensional differentiable manifold M
sage: M.diff_degree()
+Infinity
```

Actually, since `'smooth'` is the default value of the parameter `structure`, the creation of a real smooth manifold can be shortened to:

```
sage: M = Manifold(3, 'M'); M
3-dimensional differentiable manifold M
sage: M.diff_degree()
+Infinity
```

For a complex smooth manifold, we have to set the parameter `field`:

```
sage: M = Manifold(3, 'M', field='complex'); M
3-dimensional complex manifold M
sage: M.diff_degree()
+Infinity
```

See the documentation of classes *TopologicalManifold* and *DifferentiableManifold* for more detailed examples.

### Uniqueness of manifold objects

Suppose we construct a manifold named  $M$ :

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
```

At some point, we change our mind and would like to restart with a new manifold, using the same name  $M$  and keeping the previous manifold for reference:

```
sage: M_old = M # for reference
sage: M = Manifold(2, 'M', structure='topological')
```

This results in a brand new object:

```
sage: M.atlas()
[]
```

The object `M_old` is intact:

```
sage: M_old.atlas()
[Chart (M, (x, y))]
```

Both objects have the same display:

```
sage: M
2-dimensional topological manifold M
sage: M_old
2-dimensional topological manifold M
```

but they are different:

```
sage: M != M_old
True
```

Let us introduce a chart on  $M$ , using the same coordinate symbols as for `M_old`:

```
sage: X.<x,y> = M.chart()
```

The charts are displayed in the same way:

```
sage: M.atlas()
[Chart (M, (x, y))]
sage: M_old.atlas()
[Chart (M, (x, y))]
```

but they are actually different:

```
sage: M.atlas()[0] != M_old.atlas()[0]
True
```

Moreover, the two manifolds `M` and `M_old` are still considered distinct:

```
sage: M != M_old
True
```

This reflects the fact that the equality of manifold objects holds only for identical objects, i.e. one has `M1 == M2` if, and only if, `M1` is `M2`. Actually, the manifold classes inherit from `WithEqualityById`:

```
sage: isinstance(M, sage.misc.fast_methods.WithEqualityById)
True
```

```
class sage.manifolds.manifold.TopologicalManifold(n, name, field, structure, ambient=None, latex_name=None,
start_index=0, category=None, unique_tag=None)
```

Bases: `sage.manifolds.subset.ManifoldSubset`

Topological manifold over a topological field  $K$ .

Given a topological field  $K$  (in most applications,  $K = \mathbf{R}$  or  $K = \mathbf{C}$ ) and a non-negative integer  $n$ , a *topological manifold of dimension  $n$  over  $K$*  is a topological space  $M$  such that

- $M$  is a Hausdorff space,
- $M$  is second countable, and
- every point in  $M$  has a neighborhood homeomorphic to  $K^n$ .

This is a Sage *parent* class, the corresponding *element* class being `ManifoldPoint`.

INPUT:

- `n` – positive integer; dimension of the manifold
- `name` – string; name (symbol) given to the manifold
- `field` – field  $K$  on which the manifold is defined; allowed values are
  - 'real' or an object of type `RealField` (e.g., `RR`) for a manifold over  $\mathbf{R}$
  - 'complex' or an object of type `ComplexField` (e.g., `CC`) for a manifold over  $\mathbf{C}$
  - an object in the category of topological fields (see `Fields` and `TopologicalSpaces`) for other types of manifolds
- `structure` – manifold structure (see `TopologicalStructure` or `RealTopologicalStructure`)
- `ambient` – (default: `None`) if not `None`, must be a topological manifold; the created object is then an open subset of `ambient`
- `latex_name` – (default: `None`) string; LaTeX symbol to denote the manifold; if none are provided, it is set to `name`

- `start_index` – (default: 0) integer; lower value of the range of indices used for “indexed objects” on the manifold, e.g., coordinates in a chart
- `category` – (default: None) to specify the category; if None, `Manifolds (field)` is assumed (see the category `Manifolds`)
- `unique_tag` – (default: None) tag used to force the construction of a new object when all the other arguments have been used previously (without `unique_tag`, the `UniqueRepresentation` behavior inherited from `ManifoldSubset` would return the previously constructed object corresponding to these arguments)

## EXAMPLES:

A 4-dimensional topological manifold (over  $\mathbf{R}$ ):

```
sage: M = Manifold(4, 'M', latex_name=r'\mathcal{M}', structure='topological')
sage: M
4-dimensional topological manifold M
sage: latex(M)
\mathcal{M}
sage: type(M)
<class 'sage.manifolds.manifold.TopologicalManifold_with_category'>
sage: M.base_field()
Real Field with 53 bits of precision
sage: dim(M)
4
```

The input parameter `start_index` defines the range of indices on the manifold:

```
sage: M = Manifold(4, 'M', structure='topological')
sage: list(M.irange())
[0, 1, 2, 3]
sage: M = Manifold(4, 'M', structure='topological', start_index=1)
sage: list(M.irange())
[1, 2, 3, 4]
sage: list(Manifold(4, 'M', structure='topological', start_index=-2).irange())
[-2, -1, 0, 1]
```

A complex manifold:

```
sage: N = Manifold(3, 'N', structure='topological', field='complex'); N
Complex 3-dimensional topological manifold N
```

A manifold over  $\mathbf{Q}$ :

```
sage: N = Manifold(6, 'N', structure='topological', field=QQ); N
6-dimensional topological manifold N over the Rational Field
```

A manifold over  $\mathbf{Q}_5$ , the field of 5-adic numbers:

```
sage: N = Manifold(2, 'N', structure='topological', field=Qp(5)); N
2-dimensional topological manifold N over the 5-adic Field with capped
relative precision 20
```

A manifold is a Sage *parent* object, in the category of topological manifolds over a given topological field (see `Manifolds`):

```
sage: isinstance(M, Parent)
True
```

```

sage: M.category()
Category of manifolds over Real Field with 53 bits of precision
sage: from sage.categories.manifolds import Manifolds
sage: M.category() is Manifolds(RR)
True
sage: M.category() is Manifolds(M.base_field())
True
sage: M in Manifolds(RR)
True
sage: N in Manifolds(Qp(5))
True

```

The corresponding Sage *elements* are points:

```

sage: X.<t, x, y, z> = M.chart()
sage: p = M.an_element(); p
Point on the 4-dimensional topological manifold M
sage: p.parent()
4-dimensional topological manifold M
sage: M.is_parent_of(p)
True
sage: p in M
True

```

The manifold's points are instances of class *ManifoldPoint*:

```

sage: isinstance(p, sage.manifolds.point.ManifoldPoint)
True

```

Since an open subset of a topological manifold  $M$  is itself a topological manifold, open subsets of  $M$  are instances of the class *TopologicalManifold*:

```

sage: U = M.open_subset('U'); U
Open subset U of the 4-dimensional topological manifold M
sage: isinstance(U, sage.manifolds.manifold.TopologicalManifold)
True
sage: U.base_field() == M.base_field()
True
sage: dim(U) == dim(M)
True
sage: U.category()
Join of Category of subobjects of sets and Category of manifolds over
Real Field with 53 bits of precision

```

The manifold passes all the tests of the test suite relative to its category:

```

sage: TestSuite(M).run()

```

**See also:**

`sage.manifolds.manifold`

**atlas()**

Return the list of charts that have been defined on the manifold.

EXAMPLES:

Let us consider  $\mathbf{R}^2$  as a 2-dimensional manifold:

```
sage: M = Manifold(2, 'R^2', structure='topological')
```

Immediately after the manifold creation, the atlas is empty, since no chart has been defined yet:

```
sage: M.atlas()
[]
```

Let us introduce the chart of Cartesian coordinates:

```
sage: c_cart.<x,y> = M.chart()
sage: M.atlas()
[Chart (R^2, (x, y))]
```

The complement of the half line  $\{y = 0, x \geq 0\}$ :

```
sage: U = M.open_subset('U', coord_def={c_cart: (y!=0,x<0)})
sage: U.atlas()
[Chart (U, (x, y))]
sage: M.atlas()
[Chart (R^2, (x, y)), Chart (U, (x, y))]
```

Spherical (polar) coordinates on U:

```
sage: c_spher.<r, ph> = U.chart(r'r:(0,+oo) ph:(0,2*pi):\phi')
sage: U.atlas()
[Chart (U, (x, y)), Chart (U, (r, ph))]
sage: M.atlas()
[Chart (R^2, (x, y)), Chart (U, (x, y)), Chart (U, (r, ph))]
```

See also:

`top_charts()`

#### `base_field()`

Return the field on which the manifold is defined.

OUTPUT:

- a topological field

EXAMPLES:

```
sage: M = Manifold(3, 'M', structure='topological')
sage: M.base_field()
Real Field with 53 bits of precision
sage: M = Manifold(3, 'M', structure='topological', field='complex')
sage: M.base_field()
Complex Field with 53 bits of precision
sage: M = Manifold(3, 'M', structure='topological', field=QQ)
sage: M.base_field()
Rational Field
```

#### `base_field_type()`

Return the type of topological field on which the manifold is defined.

OUTPUT:

- a string describing the field, with three possible values:
  - 'real' for the real field  $\mathbf{R}$

- 'complex' for the complex field  $\mathbf{C}$
- 'neither\_real\_nor\_complex' for a field different from  $\mathbf{R}$  and  $\mathbf{C}$

EXAMPLES:

```
sage: M = Manifold(3, 'M', structure='topological')
sage: M.base_field_type()
'real'
sage: M = Manifold(3, 'M', structure='topological', field='complex')
sage: M.base_field_type()
'complex'
sage: M = Manifold(3, 'M', structure='topological', field=QQ)
sage: M.base_field_type()
'neither_real_nor_complex'
```

**chart** (*coordinates='', names=None*)

Define a chart, the domain of which is the manifold.

A *chart* is a pair  $(U, \varphi)$ , where  $U$  is the current manifold and  $\varphi : U \rightarrow V \subset K^n$  is a homeomorphism from  $U$  to an open subset  $V$  of  $K^n$ ,  $K$  being the field on which the manifold is defined.

The components  $(x^1, \dots, x^n)$  of  $\varphi$ , defined by  $\varphi(p) = (x^1(p), \dots, x^n(p)) \in K^n$  for any point  $p \in U$ , are called the *coordinates* of the chart  $(U, \varphi)$ .

See [Chart](#) for a complete documentation.

INPUT:

- *coordinates* – (default: '' (empty string)) string defining the coordinate symbols and ranges, see below
- *names* – (default: None) unused argument, except if *coordinates* is not provided; it must then be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator `<, >` is used)

The coordinates declared in the string *coordinates* are separated by ' ' (whitespace) and each coordinate has at most three fields, separated by a colon (':'):

1. The coordinate symbol (a letter or a few letters).
2. (optional, only for manifolds over  $\mathbf{R}$ ) The interval  $I$  defining the coordinate range: if not provided, the coordinate is assumed to span all  $\mathbf{R}$ ; otherwise  $I$  must be provided in the form  $(a, b)$  (or equivalently  $]a, b[$ ) The bounds  $a$  and  $b$  can be  $\pm$ Infinity, Inf, infinity, inf or  $\infty$ . For *singular* coordinates, non-open intervals such as  $[a, b]$  and  $(a, b]$  (or equivalently  $]a, b]$ ) are allowed. Note that the interval declaration must not contain any space character.
3. (optional) The LaTeX spelling of the coordinate; if not provided the coordinate symbol given in the first field will be used.

The order of the fields 2 and 3 does not matter and each of them can be omitted. If it contains any LaTeX expression, the string *coordinates* must be declared with the prefix 'r' (for "raw") to allow for a proper treatment of the backslash character (see examples below). If no interval range and no LaTeX spelling is to be provided for any coordinate, the argument *coordinates* can be omitted when the shortcut operator `<, >` is used via Sage parser (see examples below).

OUTPUT:

- the created chart, as an instance of [Chart](#) or of the subclass [RealChart](#) for manifolds over  $\mathbf{R}$ .

EXAMPLES:

Chart on a 2-dimensional manifold:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: U = M.open_subset('U')
sage: X = U.chart('x y'); X
Chart (U, (x, y))
sage: X[0]
x
sage: X[1]
y
sage: X[:]
(x, y)

```

The declared coordinates are not known at the global level:

```

sage: y
Traceback (most recent call last):
...
NameError: name 'y' is not defined

```

They can be recovered by the operator `[ : ]` applied to the chart:

```

sage: (x, y) = X[:]
sage: y
y
sage: type(y)
<type 'sage.symbolic.expression.Expression'>

```

But a shorter way to proceed is to use the operator `<, >` in the left-hand side of the chart declaration (there is then no need to pass the string `'x y'` to `chart()`):

```

sage: M = Manifold(2, 'M', structure='topological')
sage: U = M.open_subset('U')
sage: X.<x,y> = U.chart(); X
Chart (U, (x, y))

```

Indeed, the declared coordinates are then known at the global level:

```

sage: y
y
sage: (x,y) == X[:]
True

```

Actually the instruction `X.<x,y> = U.chart()` is equivalent to the combination of the two instructions `X = U.chart('x y')` and `(x,y) = X[:]`.

See the documentation of class `Chart` for more examples, especially regarding the coordinates ranges and restrictions.

**constant\_scalar\_field** (*value*, *name=None*, *latex\_name=None*)

Define a constant scalar field on the manifold.

INPUT:

- *value* – constant value of the scalar field, either a numerical value or a symbolic expression not involving any chart coordinates
- *name* – (default: `None`) name given to the scalar field
- *latex\_name* – (default: `None`) LaTeX symbol to denote the scalar field; if `None`, the LaTeX symbol is set to *name*

OUTPUT:

- instance of *ScalarField* representing the scalar field whose constant value is value

EXAMPLES:

A constant scalar field on the 2-sphere:

```
sage: M = Manifold(2, 'M', structure='topological') # the 2-dimensional_
↳sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                               intersection_name='W',
....:                               restrictions1= x^2+y^2!=0,
....:                               restrictions2= u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: f = M.constant_scalar_field(-1) ; f
Scalar field on the 2-dimensional topological manifold M
sage: f.display()
M --> R
on U: (x, y) |--> -1
on V: (u, v) |--> -1
```

We have:

```
sage: f.restrict(U) == U.constant_scalar_field(-1)
True
sage: M.constant_scalar_field(0) is M.zero_scalar_field()
True
```

See also:

*zero\_scalar\_field()*, *one\_scalar\_field()*

**continuous\_map** (*codomain*, *coord\_functions=None*, *chart1=None*, *chart2=None*, *name=None*, *latex\_name=None*)

Define a continuous map from *self* to *codomain*.

INPUT:

- codomain* – *TopologicalManifold*; the map's codomain
- coord\_functions* – (default: *None*) if not *None*, must be either
  - (i) a dictionary of the coordinate expressions (as lists (or tuples) of the coordinates of the image expressed in terms of the coordinates of the considered point) with the pairs of charts (*chart1*, *chart2*) as keys (*chart1* being a chart on *self* and *chart2* a chart on *codomain*);
  - (ii) a single coordinate expression in a given pair of charts, the latter being provided by the arguments *chart1* and *chart2*;

in both cases, if the dimension of the codomain is 1, a single coordinate expression can be passed instead of a tuple with a single element
- chart1* – (default: *None*; used only in case (ii) above) chart on *self* defining the start coordinates involved in *coord\_functions* for case (ii); if *None*, the coordinates are assumed to refer to the default chart of *self*

- `chart2` – (default: None; used only in case (ii) above) chart on `codomain` defining the target coordinates involved in `coord_functions` for case (ii); if None, the coordinates are assumed to refer to the default chart of `codomain`
- `name` – (default: None) name given to the continuous map
- `latex_name` – (default: None) LaTeX symbol to denote the continuous map; if None, the LaTeX symbol is set to `name`

OUTPUT:

- the continuous map as an instance of *ContinuousMap*

EXAMPLES:

A continuous map between an open subset of  $S^2$  covered by regular spherical coordinates and  $\mathbb{R}^3$ :

```
sage: M = Manifold(2, 'S^2', structure='topological')
sage: U = M.open_subset('U')
sage: c_spher.<th,ph> = U.chart(r'th:(0,pi):\theta ph:(0,2*pi):\phi')
sage: N = Manifold(3, 'R^3', latex_name=r'\RR^3', structure='topological')
sage: c_cart.<x,y,z> = N.chart() # Cartesian coord. on R^3
sage: Phi = U.continuous_map(N, (sin(th)*cos(ph), sin(th)*sin(ph), cos(th)),
....:                          name='Phi', latex_name=r'\Phi')
sage: Phi
Continuous map Phi from the Open subset U of the 2-dimensional topological
↳ manifold S^2 to the 3-dimensional topological manifold R^3
```

The same definition, but with a dictionary with pairs of charts as keys (case (i) above):

```
sage: Phi1 = U.continuous_map(N,
....:   {(c_spher, c_cart): (sin(th)*cos(ph), sin(th)*sin(ph), cos(th))},
....:   name='Phi', latex_name=r'\Phi')
sage: Phi1 == Phi
True
```

The continuous map acting on a point:

```
sage: p = U.point((pi/2, pi)) ; p
Point on the 2-dimensional topological manifold S^2
sage: Phi(p)
Point on the 3-dimensional topological manifold R^3
sage: Phi(p).coord(c_cart)
(-1, 0, 0)
sage: Phi1(p) == Phi(p)
True
```

**See also:**

See *ContinuousMap* for the complete documentation and more examples.

---

**Todo**

Allow the construction of continuous maps from `self` to the base field (considered as a trivial 1-dimensional manifold).

---

**coord\_change** (*chart1*, *chart2*)

Return the change of coordinates (transition map) between two charts defined on the manifold.

The change of coordinates must have been defined previously, for instance by the method `transition_map()`.

INPUT:

- chart1 – chart 1
- chart2 – chart 2

OUTPUT:

- instance of `CoordChange` representing the transition map from chart 1 to chart 2

EXAMPLES:

Change of coordinates on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: c_uv.<u,v> = M.chart()
sage: c_xy.transition_map(c_uv, (x+y, x-y)) # defines the coord. change
Change of coordinates from Chart (M, (x, y)) to Chart (M, (u, v))
sage: M.coord_change(c_xy, c_uv) # returns the coord. change defined above
Change of coordinates from Chart (M, (x, y)) to Chart (M, (u, v))
```

`coord_changes()`

Return the changes of coordinates (transition maps) defined on subsets of the manifold.

OUTPUT:

- dictionary of changes of coordinates, with pairs of charts as keys

EXAMPLES:

Various changes of coordinates on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: c_uv.<u,v> = M.chart()
sage: xy_to_uv = c_xy.transition_map(c_uv, [x+y, x-y])
sage: M.coord_changes()
{(Chart (M, (x, y)),
  Chart (M, (u, v))): Change of coordinates from Chart (M, (x, y)) to Chart
↔(M, (u, v))}
sage: uv_to_xy = xy_to_uv.inverse()
sage: M.coord_changes() # random (dictionary output)
{(Chart (M, (u, v)),
  Chart (M, (x, y))): Change of coordinates from Chart (M, (u, v)) to Chart
↔(M, (x, y)),
 (Chart (M, (x, y)),
  Chart (M, (u, v))): Change of coordinates from Chart (M, (x, y)) to Chart
↔(M, (u, v))}
sage: c_rs.<r,s> = M.chart()
sage: uv_to_rs = c_uv.transition_map(c_rs, [-u+2*v, 3*u-v])
sage: M.coord_changes() # random (dictionary output)
{(Chart (M, (u, v)),
  Chart (M, (r, s))): Change of coordinates from Chart (M, (u, v)) to Chart
↔(M, (r, s)),
 (Chart (M, (u, v)),
  Chart (M, (x, y))): Change of coordinates from Chart (M, (u, v)) to Chart
↔(M, (x, y)),
 (Chart (M, (x, y)),
  Chart (M, (x, y))): Change of coordinates from Chart (M, (x, y)) to Chart
↔(M, (x, y))}
```

```

Chart (M, (u, v)): Change of coordinates from Chart (M, (x, y)) to Chart
↪(M, (u, v))
sage: xy_to_rs = uv_to_rs * xy_to_uv
sage: M.coord_changes() # random (dictionary output)
{(Chart (M, (u, v)),
 Chart (M, (r, s))): Change of coordinates from Chart (M, (u, v)) to Chart
↪(M, (r, s)),
 (Chart (M, (u, v)),
 Chart (M, (x, y))): Change of coordinates from Chart (M, (u, v)) to Chart
↪(M, (x, y)),
 (Chart (M, (x, y)),
 Chart (M, (u, v))): Change of coordinates from Chart (M, (x, y)) to Chart
↪(M, (u, v)),
 (Chart (M, (x, y)),
 Chart (M, (r, s))): Change of coordinates from Chart (M, (x, y)) to Chart
↪(M, (r, s))}

```

**default\_chart()**

Return the default chart defined on the manifold.

Unless changed via `set_default_chart()`, the *default chart* is the first one defined on a subset of the manifold (possibly itself).

OUTPUT:

- instance of `Chart` representing the default chart

EXAMPLES:

Default chart on a 2-dimensional manifold and on some subsets:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: M.chart('x y')
Chart (M, (x, y))
sage: M.chart('u v')
Chart (M, (u, v))
sage: M.default_chart()
Chart (M, (x, y))
sage: A = M.open_subset('A')
sage: A.chart('t z')
Chart (A, (t, z))
sage: A.default_chart()
Chart (A, (t, z))

```

**dim()**

Return the dimension of the manifold over its base field.

EXAMPLES:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: M.dimension()
2

```

A shortcut is `dim()`:

```

sage: M.dim()
2

```

The Sage global function `dim` can also be used:

```
sage: dim(M)
2
```

**dimension()**

Return the dimension of the manifold over its base field.

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: M.dimension()
2
```

A shortcut is `dim()`:

```
sage: M.dim()
2
```

The Sage global function `dim` can also be used:

```
sage: dim(M)
2
```

**get\_chart** (*coordinates, domain=None*)

Get a chart from its coordinates.

The chart must have been previously created by the method `chart()`.

INPUT:

- `coordinates` – single string composed of the coordinate symbols separated by a space
- `domain` – (default: `None`) string containing the name of the chart's domain, which must be a subset of the current manifold; if `None`, the current manifold is assumed

OUTPUT:

- instance of `Chart` (or of the subclass `RealChart`) representing the chart corresponding to the above specifications

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x, y> = M.chart()
sage: M.get_chart('x y')
Chart (M, (x, y))
sage: M.get_chart('x y') is X
True
sage: U = M.open_subset('U', coord_def={X: (y!=0, x<0)})
sage: Y.<r, ph> = U.chart(r'r:(0,+oo) ph:(0,2*pi):\phi')
sage: M.atlas()
[Chart (M, (x, y)), Chart (U, (x, y)), Chart (U, (r, ph))]
sage: M.get_chart('x y', domain='U')
Chart (U, (x, y))
sage: M.get_chart('x y', domain='U') is X.restrict(U)
True
sage: U.get_chart('r ph')
Chart (U, (r, ph))
sage: M.get_chart('r ph', domain='U')
Chart (U, (r, ph))
```

```
sage: M.get_chart('r ph', domain='U') is Y
True
```

**global\_options** (\*args, \*\*kws)

Deprecated: Use `options()` instead. See [trac ticket #18555](#) for details.

**homeomorphism**(codomain, coord\_functions=None, chart1=None, chart2=None, name=None, latex\_name=None)

Define a homeomorphism between the current manifold and another one.

See *ContinuousMap* for a complete documentation.

INPUT:

- `codomain` – *TopologicalManifold*; codomain of the homeomorphism
- `coord_functions` – (default: None) if not None, must be either
  - (i) a dictionary of the coordinate expressions (as lists (or tuples) of the coordinates of the image expressed in terms of the coordinates of the considered point) with the pairs of charts (`chart1`, `chart2`) as keys (`chart1` being a chart on `self` and `chart2` a chart on `codomain`);
  - (ii) a single coordinate expression in a given pair of charts, the latter being provided by the arguments `chart1` and `chart2`;
 in both cases, if the dimension of the codomain is 1, a single coordinate expression can be passed instead of a tuple with a single element
- `chart1` – (default: None; used only in case (ii) above) chart on `self` defining the start coordinates involved in `coord_functions` for case (ii); if None, the coordinates are assumed to refer to the default chart of `self`
- `chart2` – (default: None; used only in case (ii) above) chart on `codomain` defining the target coordinates involved in `coord_functions` for case (ii); if None, the coordinates are assumed to refer to the default chart of `codomain`
- `name` – (default: None) name given to the homeomorphism
- `latex_name` – (default: None) LaTeX symbol to denote the homeomorphism; if None, the LaTeX symbol is set to `name`

OUTPUT:

- the homeomorphism, as an instance of *ContinuousMap*

EXAMPLES:

Homeomorphism between the open unit disk in  $\mathbf{R}^2$  and  $\mathbf{R}^2$ :

```
sage: forget() # for doctests only
sage: M = Manifold(2, 'M', structure='topological') # the open unit disk
sage: c_xy.<x,y> = M.chart('x: (-1,1) y: (-1,1)') # Cartesian coord on M
sage: c_xy.add_restrictions(x^2+y^2<1)
sage: N = Manifold(2, 'N', structure='topological') # R^2
sage: c_XY.<X,Y> = N.chart() # canonical coordinates on R^2
sage: Phi = M.homeomorphism(N, [x/sqrt(1-x^2-y^2), y/sqrt(1-x^2-y^2)],
....:                          name='Phi', latex_name=r'\Phi')
sage: Phi
Homeomorphism Phi from the 2-dimensional topological manifold M to
the 2-dimensional topological manifold N
sage: Phi.display()
Phi: M --> N
(x, y) |--> (X, Y) = (x/sqrt(-x^2 - y^2 + 1), y/sqrt(-x^2 - y^2 + 1))
```

The inverse homeomorphism:

```
sage: Phi^(-1)
Homeomorphism Phi^(-1) from the 2-dimensional topological
manifold N to the 2-dimensional topological manifold M
sage: (Phi^(-1)).display()
Phi^(-1): N --> M
      (X, Y) |--> (x, y) = (X/sqrt(X^2 + Y^2 + 1), Y/sqrt(X^2 + Y^2 + 1))
```

See the documentation of *ContinuousMap* for more examples.

### **identity\_map()**

Identity map of self.

The identity map of a topological manifold  $M$  is the trivial homeomorphism:

$$\begin{array}{ccc} \text{Id}_M : M & \longrightarrow & M \\ p & \longmapsto & p \end{array}$$

OUTPUT:

- the identity map as an instance of *ContinuousMap*

EXAMPLES:

Identity map of a complex manifold:

```
sage: M = Manifold(2, 'M', structure='topological', field='complex')
sage: X.<x,y> = M.chart()
sage: id = M.identity_map(); id
Identity map Id_M of the Complex 2-dimensional topological manifold M
sage: id.parent()
Set of Morphisms from Complex 2-dimensional topological manifold M
to Complex 2-dimensional topological manifold M in Category of
manifolds over Complex Field with 53 bits of precision
sage: id.display()
Id_M: M --> M
      (x, y) |--> (x, y)
```

The identity map acting on a point:

```
sage: p = M((1+I, 3-I), name='p'); p
Point p on the Complex 2-dimensional topological manifold M
sage: id(p)
Point p on the Complex 2-dimensional topological manifold M
sage: id(p) == p
True
```

**See also:**

See *ContinuousMap* for the complete documentation.

### **index\_generator**(*nb\_indices*)

Generator of index series.

INPUT:

- nb\_indices* – number of indices in a series

OUTPUT:

- an iterable index series for a generic component with the specified number of indices

EXAMPLES:

Indices on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological', start_index=1)
sage: list(M.index_generator(2))
[(1, 1), (1, 2), (2, 1), (2, 2)]
```

Loops can be nested:

```
sage: for ind1 in M.index_generator(2):
...:     print("{} : {}".format(ind1, list(M.index_generator(2))))
(1, 1) : [(1, 1), (1, 2), (2, 1), (2, 2)]
(1, 2) : [(1, 1), (1, 2), (2, 1), (2, 2)]
(2, 1) : [(1, 1), (1, 2), (2, 1), (2, 2)]
(2, 2) : [(1, 1), (1, 2), (2, 1), (2, 2)]
```

**irange** (*start=None*)

Single index generator.

INPUT:

- *start* – (default: None) initial value  $i_0$  of the index; if none are provided, the value returned by `start_index()` is assumed

OUTPUT:

- an iterable index, starting from  $i_0$  and ending at  $i_0 + n - 1$ , where  $n$  is the manifold's dimension

EXAMPLES:

Index range on a 4-dimensional manifold:

```
sage: M = Manifold(4, 'M', structure='topological')
sage: list(M.irange())
[0, 1, 2, 3]
sage: list(M.irange(2))
[2, 3]
```

Index range on a 4-dimensional manifold with starting index=1:

```
sage: M = Manifold(4, 'M', structure='topological', start_index=1)
sage: list(M.irange())
[1, 2, 3, 4]
sage: list(M.irange(2))
[2, 3, 4]
```

In general, one has always:

```
sage: next(M.irange()) == M.start_index()
True
```

**is\_manifestly\_coordinate\_domain** ()

Return True if the manifold is known to be the domain of some coordinate chart and False otherwise.

If False is returned, either the manifold cannot be the domain of some coordinate chart or no such chart has been declared yet.

EXAMPLES:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: U = M.open_subset('U')
sage: X.<x,y> = U.chart()
sage: U.is_manifestly_coordinate_domain()
True
sage: M.is_manifestly_coordinate_domain()
False
sage: Y.<u,v> = M.chart()
sage: M.is_manifestly_coordinate_domain()
True

```

**is\_open()**

Return if `self` is an open set.

In the present case (manifold or open subset of it), always return `True`.

**one\_scalar\_field()**

Return the constant scalar field with value the unit element of the base field of `self`.

OUTPUT:

- a *ScalarField* representing the constant scalar field with value the unit element of the base field of `self`

EXAMPLES:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.one_scalar_field(); f
Scalar field 1 on the 2-dimensional topological manifold M
sage: f.display()
1: M --> R
   (x, y) |--> 1
sage: f.parent()
Algebra of scalar fields on the 2-dimensional topological manifold M
sage: f is M.scalar_field_algebra().one()
True

```

**open\_subset** (*name*, *latex\_name=None*, *coord\_def={}*)

Create an open subset of the manifold.

An open subset is a set that is (i) included in the manifold and (ii) open with respect to the manifold's topology. It is a topological manifold by itself. Hence the returned object is an instance of *TopologicalManifold*.

INPUT:

- *name* – name given to the open subset
- *latex\_name* – (default: `None`) LaTeX symbol to denote the subset; if none are provided, it is set to *name*
- *coord\_def* – (default: `{}`) definition of the subset in terms of coordinates; *coord\_def* must be a dictionary with keys *charts* on the manifold and values the symbolic expressions formed by the coordinates to define the subset

OUTPUT:

- the open subset, as an instance of *TopologicalManifold*

EXAMPLES:

Creating an open subset of a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: A = M.open_subset('A'); A
Open subset A of the 2-dimensional topological manifold M
```

As an open subset of a topological manifold, A is itself a topological manifold, on the same topological field and of the same dimension as M:

```
sage: isinstance(A, sage.manifolds.manifold.TopologicalManifold)
True
sage: A.base_field() == M.base_field()
True
sage: dim(A) == dim(M)
True
sage: A.category() is M.category().Subobjects()
True
```

Creating an open subset of A:

```
sage: B = A.open_subset('B'); B
Open subset B of the 2-dimensional topological manifold M
```

We have then:

```
sage: A.subsets() # random (set output)
{Open subset B of the 2-dimensional topological manifold M,
 Open subset A of the 2-dimensional topological manifold M}
sage: B.is_subset(A)
True
sage: B.is_subset(M)
True
```

Defining an open subset by some coordinate restrictions: the open unit disk in  $\mathbb{R}^2$ :

```
sage: M = Manifold(2, 'R^2', structure='topological')
sage: c_cart.<x,y> = M.chart() # Cartesian coordinates on R^2
sage: U = M.open_subset('U', coord_def={c_cart: x^2+y^2<1}); U
Open subset U of the 2-dimensional topological manifold R^2
```

Since the argument `coord_def` has been set, U is automatically provided with a chart, which is the restriction of the Cartesian one to U:

```
sage: U.atlas()
[Chart (U, (x, y))]
```

Therefore, one can immediately check whether a point belongs to U:

```
sage: M.point((0,0)) in U
True
sage: M.point((1/2,1/3)) in U
True
sage: M.point((1,2)) in U
False
```

**options** (*\*get\_value*, *\*\*set\_value*)

Sets and displays the options for manifolds. If no parameters are set, then the function returns a copy of the options dictionary.

The options to manifolds can be accessed as the method `Manifold.options`.

OPTIONS:

- `omit_function_arguments` – (default: `False`) Determine if the arguments of symbolic functions are printed
- `textbook_output` – (default: `True`) textbook-like output instead of the Pynac output for derivatives

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: g = function('g')(x, y)
```

For coordinate functions, the display is more “textbook” like:

```
sage: f = X.function(diff(g, x) + diff(g, y))
sage: f
d(g)/dx + d(g)/dy
sage: latex(f)
\frac{\partial g}{\partial x} + \frac{\partial g}{\partial y}
```

One can switch to Pynac notation by changing `textbook_output` to `False`:

```
sage: Manifold.options.textbook_output=False
sage: f
D[0](g)(x, y) + D[1](g)(x, y)
sage: latex(f)
D[0]\left(g\right)\left(x, y\right) + D[1]\left(g\right)\left(x, y\right)
sage: Manifold.options._reset()
```

If there is a clear understanding that  $u$  and  $v$  are functions of  $(x, y)$ , the explicit mention of the latter can be cumbersome in lengthy tensor expressions:

```
sage: f = X.function(function('u')(x, y) * function('v')(x, y))
sage: f
u(x, y)*v(x, y)
```

We can switch it off by:

```
sage: M.options.omit_function_arguments=True
sage: f
u*v
sage: M.options._reset()
```

See `GlobalOptions` for more features of these options.

**scalar\_field** (*coord\_expression=None, chart=None, name=None, latex\_name=None*)

Define a scalar field on the manifold.

See `ScalarField` (or `DiffScalarField` if the manifold is differentiable) for a complete documentation.

INPUT:

- `coord_expression` – (default: `None`) coordinate expression(s) of the scalar field; this can be either

- a single coordinate expression; if the argument `chart` is 'all', this expression is set to all the charts defined on the open set; otherwise, the expression is set in the specific chart provided by the argument `chart`
- a dictionary of coordinate expressions, with the charts as keys
- `chart` – (default: None) chart defining the coordinates used in `coord_expression` when the latter is a single coordinate expression; if None, the default chart of the open set is assumed; if `chart=='all'`, `coord_expression` is assumed to be independent of the chart (constant scalar field)
- `name` – (default: None) name given to the scalar field
- `latex_name` – (default: None) LaTeX symbol to denote the scalar field; if None, the LaTeX symbol is set to `name`

If `coord_expression` is None or does not fully specified the scalar field, other coordinate expressions can be added subsequently by means of the methods `add_expr()`, `add_expr_by_continuation()`, or `set_expr()`

OUTPUT:

- instance of `ScalarField` (or of the subclass `DiffScalarField` if the manifold is differentiable) representing the defined scalar field

EXAMPLES:

A scalar field defined by its coordinate expression in the open set's default chart:

```
sage: M = Manifold(3, 'M', structure='topological')
sage: U = M.open_subset('U')
sage: c_xyz.<x,y,z> = U.chart()
sage: f = U.scalar_field(sin(x)*cos(y) + z, name='F'); f
Scalar field F on the Open subset U of the 3-dimensional topological manifold
↳M
sage: f.display()
F: U --> R
(x, y, z) |--> cos(y)*sin(x) + z
sage: f.parent()
Algebra of scalar fields on the Open subset U of the 3-dimensional
↳topological manifold M
sage: f in U.scalar_field_algebra()
True
```

Equivalent definition with the chart specified:

```
sage: f = U.scalar_field(sin(x)*cos(y) + z, chart=c_xyz, name='F')
sage: f.display()
F: U --> R
(x, y, z) |--> cos(y)*sin(x) + z
```

Equivalent definition with a dictionary of coordinate expression(s):

```
sage: f = U.scalar_field({c_xyz: sin(x)*cos(y) + z}, name='F')
sage: f.display()
F: U --> R
(x, y, z) |--> cos(y)*sin(x) + z
```

See the documentation of class `ScalarField` for more examples.

See also:

```
constant_scalar_field(), zero_scalar_field(), one_scalar_field()
```

### **scalar\_field\_algebra()**

Return the algebra of scalar fields defined the manifold.

See *ScalarFieldAlgebra* for a complete documentation.

OUTPUT:

- instance of *ScalarFieldAlgebra* representing the algebra  $C^0(U)$  of all scalar fields defined on  $U = \text{self}$

EXAMPLES:

Scalar algebra of a 3-dimensional open subset:

```
sage: M = Manifold(3, 'M', structure='topological')
sage: U = M.open_subset('U')
sage: CU = U.scalar_field_algebra() ; CU
Algebra of scalar fields on the Open subset U of the 3-dimensional_
↳topological manifold M
sage: CU.category()
Category of commutative algebras over Symbolic Ring
sage: CU.zero()
Scalar field zero on the Open subset U of the 3-dimensional topological_
↳manifold M
```

The output is cached:

```
sage: U.scalar_field_algebra() is CU
True
```

### **set\_default\_chart(chart)**

Changing the default chart on self.

INPUT:

- chart – a chart (must be defined on some subset self)

EXAMPLES:

Charts on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: c_uv.<u,v> = M.chart()
sage: M.default_chart()
Chart (M, (x, y))
sage: M.set_default_chart(c_uv)
sage: M.default_chart()
Chart (M, (u, v))
```

### **start\_index()**

Return the first value of the index range used on the manifold.

This is the parameter `start_index` passed at the construction of the manifold.

OUTPUT:

- the integer  $i_0$  such that all indices of indexed objects on the manifold range from  $i_0$  to  $i_0 + n - 1$ , where  $n$  is the manifold's dimension

EXAMPLES:

```

sage: M = Manifold(3, 'M', structure='topological')
sage: M.start_index()
0
sage: M = Manifold(3, 'M', structure='topological', start_index=1)
sage: M.start_index()
1

```

**top\_charts()**

Return the list of charts defined on subsets of the current manifold that are not subcharts of charts on larger subsets.

OUTPUT:

- list of charts defined on open subsets of the manifold but not on larger subsets

EXAMPLES:

Charts on a 2-dimensional manifold:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: U = M.open_subset('U', coord_def={X: x>0})
sage: Y.<u,v> = U.chart()
sage: M.top_charts()
[Chart (M, (x, y)), Chart (U, (u, v))]

```

Note that the (user) atlas contains one more chart:  $(U, (x, y))$ , which is not a “top” chart:

```

sage: M.atlas()
[Chart (M, (x, y)), Chart (U, (x, y)), Chart (U, (u, v))]

```

**See also:**

`atlas()` for the complete list of charts defined on the manifold.

**zero\_scalar\_field()**

Return the zero scalar field defined on `self`.

OUTPUT:

- a *ScalarField* representing the constant scalar field with value 0

EXAMPLES:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.zero_scalar_field() ; f
Scalar field zero on the 2-dimensional topological manifold M
sage: f.display()
zero: M --> R
      (x, y) |--> 0
sage: f.parent()
Algebra of scalar fields on the 2-dimensional topological manifold M
sage: f is M.scalar_field_algebra().zero()
True

```

## 1.2 Subsets of Topological Manifolds

The class *ManifoldSubset* implements generic subsets of a topological manifold. Open subsets are implemented by the class *TopologicalManifold* (since an open subset of a manifold is a manifold by itself), which inherits from *ManifoldSubset*.

AUTHORS:

- Ericourgoulhon, Michal Bejger (2013-2015): initial version
- Travis Scrimshaw (2015): review tweaks; removal of facade parents

REFERENCES:

- [Lee2011]

EXAMPLES:

Two subsets on a manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: a = M.subset('A'); a
Subset A of the 2-dimensional topological manifold M
sage: b = M.subset('B'); b
Subset B of the 2-dimensional topological manifold M
sage: M.list_of_subsets()
[Subset A of the 2-dimensional topological manifold M,
Subset B of the 2-dimensional topological manifold M,
2-dimensional topological manifold M]
```

The intersection of the two subsets:

```
sage: c = a.intersection(b); c
Subset A_inter_B of the 2-dimensional topological manifold M
```

Their union:

```
sage: d = a.union(b); d
Subset A_union_B of the 2-dimensional topological manifold M
```

Lists of subsets after the above operations:

```
sage: M.list_of_subsets()
[Subset A of the 2-dimensional topological manifold M,
Subset A_inter_B of the 2-dimensional topological manifold M,
Subset A_union_B of the 2-dimensional topological manifold M,
Subset B of the 2-dimensional topological manifold M,
2-dimensional topological manifold M]
sage: a.list_of_subsets()
[Subset A of the 2-dimensional topological manifold M,
Subset A_inter_B of the 2-dimensional topological manifold M]
sage: c.list_of_subsets()
[Subset A_inter_B of the 2-dimensional topological manifold M]
sage: d.list_of_subsets()
[Subset A of the 2-dimensional topological manifold M,
Subset A_inter_B of the 2-dimensional topological manifold M,
Subset A_union_B of the 2-dimensional topological manifold M,
Subset B of the 2-dimensional topological manifold M]
```

```
class sage.manifolds.subset.ManifoldSubset (manifold, name, latex_name=None, category=None)
    Bases: sage.structure.unique_representation.UniqueRepresentation, sage.structure.parent.Parent
```

Subset of a topological manifold.

The class *ManifoldSubset* inherits from the generic class *Parent*. The corresponding element class is *ManifoldPoint*.

Note that open subsets are not implemented directly by this class, but by the derived class *TopologicalManifold* (an open subset of a topological manifold being itself a topological manifold).

INPUT:

- *manifold* – topological manifold on which the subset is defined
- *name* – string; name (symbol) given to the subset
- *latex\_name* – (default: None) string; LaTeX symbol to denote the subset; if none are provided, it is set to *name*
- *category* – (default: None) to specify the category; if None, the category for generic subsets is used

EXAMPLES:

A subset of a manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: from sage.manifolds.subset import ManifoldSubset
sage: A = ManifoldSubset(M, 'A', latex_name=r'\mathcal{A}')
sage: A
Subset A of the 2-dimensional topological manifold M
sage: latex(A)
\mathcal{A}
sage: A.is_subset(M)
True
```

Instead of importing *ManifoldSubset* in the global namespace, it is recommended to use the method *subset()* to create a new subset:

```
sage: B = M.subset('B', latex_name=r'\mathcal{B}'); B
Subset B of the 2-dimensional topological manifold M
sage: M.list_of_subsets()
[Subset A of the 2-dimensional topological manifold M,
Subset B of the 2-dimensional topological manifold M,
2-dimensional topological manifold M]
```

The manifold is itself a subset:

```
sage: isinstance(M, ManifoldSubset)
True
sage: M in M.subsets()
True
```

Instances of *ManifoldSubset* are parents:

```
sage: isinstance(A, Parent)
True
sage: A.category()
Category of subobjects of sets
sage: p = A.an_element(); p
```

```

Point on the 2-dimensional topological manifold M
sage: p.parent()
Subset A of the 2-dimensional topological manifold M
sage: p in A
True
sage: p in M
True

```

**Element**

alias of `ManifoldPoint`

**ambient()**

Return the ambient manifold of `self`.

EXAMPLES:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: A = M.subset('A')
sage: A.manifold()
2-dimensional topological manifold M
sage: A.manifold() is M
True
sage: B = A.subset('B')
sage: B.manifold() is M
True

```

An alias is `ambient`:

```

sage: A.ambient() is A.manifold()
True

```

**declare\_union(*dom1*, *dom2*)**

Declare that the current subset is the union of two subsets.

Suppose  $U$  is the current subset, then this method declares that  $U$

$$U = U_1 \cup U_2,$$

where  $U_1 \subset U$  and  $U_2 \subset U$ .

INPUT:

- `dom1` – the subset  $U_1$
- `dom2` – the subset  $U_2$

EXAMPLES:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: A = M.subset('A')
sage: B = M.subset('B')
sage: M.declare_union(A, B)
sage: A.union(B)
2-dimensional topological manifold M

```

**get\_subset(*name*)**

Get a subset by its name.

The subset must have been previously created by the method `subset()` (or `open_subset()`)

INPUT:

- name – (string) name of the subset

OUTPUT:

- instance of *ManifoldSubset* (or of the derived class *TopologicalManifold* for an open subset) representing the subset whose name is name

EXAMPLES:

```
sage: M = Manifold(4, 'M', structure='topological')
sage: A = M.subset('A')
sage: B = A.subset('B')
sage: U = M.open_subset('U')
sage: M.list_of_subsets()
[Subset A of the 4-dimensional topological manifold M,
 Subset B of the 4-dimensional topological manifold M,
 4-dimensional topological manifold M,
 Open subset U of the 4-dimensional topological manifold M]
sage: M.get_subset('A')
Subset A of the 4-dimensional topological manifold M
sage: M.get_subset('A') is A
True
sage: M.get_subset('B') is B
True
sage: A.get_subset('B') is B
True
sage: M.get_subset('U')
Open subset U of the 4-dimensional topological manifold M
sage: M.get_subset('U') is U
True
```

**intersection** (*other*, *name=None*, *latex\_name=None*)

Return the intersection of the current subset with another subset.

INPUT:

- other – another subset of the same manifold
- name – (default: None) name given to the intersection in the case the latter has to be created; the default is `self._name inter other._name`
- latex\_name – (default: None) LaTeX symbol to denote the intersection in the case the latter has to be created; the default is built upon the symbol  $\cap$

OUTPUT:

- instance of *ManifoldSubset* representing the subset that is the intersection of the current subset with other

EXAMPLES:

Intersection of two subsets:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: a = M.subset('A')
sage: b = M.subset('B')
sage: c = a.intersection(b); c
Subset A_inter_B of the 2-dimensional topological manifold M
sage: a.list_of_subsets()
[Subset A of the 2-dimensional topological manifold M,
 Subset A_inter_B of the 2-dimensional topological manifold M]
sage: b.list_of_subsets()
```

```
[Subset A_inter_B of the 2-dimensional topological manifold M,
Subset B of the 2-dimensional topological manifold M]
sage: c._supersets # random (set output)
{Subset B of the 2-dimensional topological manifold M,
Subset A_inter_B of the 2-dimensional topological manifold M,
Subset A of the 2-dimensional topological manifold M,
2-dimensional topological manifold M}
```

Some checks:

```
sage: (a.intersection(b)).is_subset(a)
True
sage: (a.intersection(b)).is_subset(a)
True
sage: a.intersection(b) is b.intersection(a)
True
sage: a.intersection(a.intersection(b)) is a.intersection(b)
True
sage: (a.intersection(b)).intersection(a) is a.intersection(b)
True
sage: M.intersection(a) is a
True
sage: a.intersection(M) is a
True
```

### `is_open()`

Return if `self` is an open set.

This method always returns `False`, since open subsets must be constructed as instances of the subclass `TopologicalManifold` (which redefines `is_open`)

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: A = M.subset('A')
sage: A.is_open()
False
```

### `is_subset(other)`

Return `True` if and only if `self` is included in `other`.

EXAMPLES:

Subsets on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: a = M.subset('A')
sage: b = a.subset('B')
sage: c = M.subset('C')
sage: a.is_subset(M)
True
sage: b.is_subset(a)
True
sage: b.is_subset(M)
True
sage: a.is_subset(b)
False
sage: c.is_subset(a)
False
```

**lift** (*p*)

Return the lift of *p* to the ambient manifold of *self*.

INPUT:

- *p* – point of the subset

OUTPUT:

- the same point, considered as a point of the ambient manifold

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: A = M.open_subset('A', coord_def={X: x>0})
sage: p = A((1, -2)); p
Point on the 2-dimensional topological manifold M
sage: p.parent()
Open subset A of the 2-dimensional topological manifold M
sage: q = A.lift(p); q
Point on the 2-dimensional topological manifold M
sage: q.parent()
2-dimensional topological manifold M
sage: q.coord()
(1, -2)
sage: (p == q) and (q == p)
True
```

**list\_of\_subsets** ()

Return the list of subsets that have been defined on the current subset.

The list is sorted by the alphabetical names of the subsets.

OUTPUT:

- a list containing all the subsets that have been defined on the current subset

---

**Note:** To get the subsets as a Python set, used the method `subsets()` instead.

---

EXAMPLES:

Subsets of a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: U = M.open_subset('U')
sage: V = M.subset('V')
sage: M.list_of_subsets()
[2-dimensional topological manifold M,
 Open subset U of the 2-dimensional topological manifold M,
 Subset V of the 2-dimensional topological manifold M]
```

The method `subsets()` returns a set instead of a list:

```
sage: M.subsets() # random (set output)
{Subset V of the 2-dimensional topological manifold M,
 2-dimensional topological manifold M,
 Open subset U of the 2-dimensional topological manifold M}
```

**manifold()**

Return the ambient manifold of `self`.

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: A = M.subset('A')
sage: A.manifold()
2-dimensional topological manifold M
sage: A.manifold() is M
True
sage: B = A.subset('B')
sage: B.manifold() is M
True
```

An alias is `ambient`:

```
sage: A.ambient() is A.manifold()
True
```

**open\_covers()**

Return the list of open covers of the current subset.

If the current subset,  $A$  say, is a subset of the manifold  $M$ , an *open cover* of  $A$  is list (indexed set)  $(U_i)_{i \in I}$  of open subsets of  $M$  such that

$$A \subset \bigcup_{i \in I} U_i.$$

If  $A$  is open, we ask that the above inclusion is actually an identity:

$$A = \bigcup_{i \in I} U_i.$$

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: M.open_covers()
[[2-dimensional topological manifold M]]
sage: U = M.open_subset('U')
sage: U.open_covers()
[[Open subset U of the 2-dimensional topological manifold M]]
sage: A = U.open_subset('A')
sage: B = U.open_subset('B')
sage: U.declare_union(A,B)
sage: U.open_covers()
[[Open subset U of the 2-dimensional topological manifold M],
 [Open subset A of the 2-dimensional topological manifold M,
  Open subset B of the 2-dimensional topological manifold M]]
sage: V = M.open_subset('V')
sage: M.declare_union(U,V)
sage: M.open_covers()
[[2-dimensional topological manifold M],
 [Open subset U of the 2-dimensional topological manifold M,
  Open subset V of the 2-dimensional topological manifold M],
 [Open subset A of the 2-dimensional topological manifold M,
  Open subset B of the 2-dimensional topological manifold M,
  Open subset V of the 2-dimensional topological manifold M]]
```

**point** (*coords=None, chart=None, name=None, latex\_name=None*)

Define a point in self.

See *ManifoldPoint* for a complete documentation.

INPUT:

- *coords* – the point coordinates (as a tuple or a list) in the chart specified by *chart*
- *chart* – (default: None) chart in which the point coordinates are given; if None, the coordinates are assumed to refer to the default chart of the current subset
- *name* – (default: None) name given to the point
- *latex\_name* – (default: None) LaTeX symbol to denote the point; if None, the LaTeX symbol is set to *name*

OUTPUT:

- the declared point, as an instance of *ManifoldPoint*

EXAMPLES:

Points on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: p = M.point((1,2), name='p'); p
Point p on the 2-dimensional topological manifold M
sage: p in M
True
sage: a = M.open_subset('A')
sage: c_uv.<u,v> = a.chart()
sage: q = a.point((-1,0), name='q'); q
Point q on the 2-dimensional topological manifold M
sage: q in a
True
sage: p._coordinates
{Chart (M, (x, y)): (1, 2)}
sage: q._coordinates
{Chart (A, (u, v)): (-1, 0)}
```

**retract** (*p*)

Return the retract of *p* to self.

INPUT:

- *p* – point of the ambient manifold

OUTPUT:

- the same point, considered as a point of the subset

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: A = M.open_subset('A', coord_def={X: x>0})
sage: p = M((1, -2)); p
Point on the 2-dimensional topological manifold M
sage: p.parent()
2-dimensional topological manifold M
sage: q = A.retract(p); q
Point on the 2-dimensional topological manifold M
```

```

sage: q.parent()
Open subset A of the 2-dimensional topological manifold M
sage: q.coord()
(1, -2)
sage: (q == p) and (p == q)
True

```

Of course, if the point does not belong to A, the retract method fails:

```

sage: p = M((-1, 3)) # x < 0, so that p is not in A
sage: q = A.retract(p)
Traceback (most recent call last):
...
ValueError: the Point on the 2-dimensional topological manifold M
is not in Open subset A of the 2-dimensional topological manifold M

```

**subset** (*name*, *latex\_name=None*, *is\_open=False*)

Create a subset of the current subset.

INPUT:

- *name* – name given to the subset
- *latex\_name* – (default: None) LaTeX symbol to denote the subset; if none are provided, it is set to *name*
- *is\_open* – (default: False) if True, the created subset is assumed to be open with respect to the manifold's topology

OUTPUT:

- the subset, as an instance of *ManifoldSubset*, or of the derived class *TopologicalManifold* if *is\_open* is True

EXAMPLES:

Creating a subset of a manifold:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: a = M.subset('A'); a
Subset A of the 2-dimensional topological manifold M

```

Creating a subset of A:

```

sage: b = a.subset('B', latex_name=r'\mathcal{B}'); b
Subset B of the 2-dimensional topological manifold M
sage: latex(b)
\mathcal{B}

```

We have then:

```

sage: b.is_subset(a)
True
sage: b in a.subsets()
True

```

**subsets** ()

Return the set of subsets that have been defined on the current subset.

OUTPUT:

- a Python set containing all the subsets that have been defined on the current subset

---

**Note:** To get the subsets as a list, used the method `list_of_subsets()` instead.

---

EXAMPLES:

Subsets of a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: U = M.open_subset('U')
sage: V = M.subset('V')
sage: M.subsets() # random (set output)
{Subset V of the 2-dimensional topological manifold M,
 2-dimensional topological manifold M,
 Open subset U of the 2-dimensional topological manifold M}
sage: type(M.subsets())
<... 'frozenset'>
sage: U in M.subsets()
True
```

The method `list_of_subsets()` returns a list (sorted alphabetically by the subset names) instead of a set:

```
sage: M.list_of_subsets()
[2-dimensional topological manifold M,
 Open subset U of the 2-dimensional topological manifold M,
 Subset V of the 2-dimensional topological manifold M]
```

**superset** (*name*, *latex\_name=None*, *is\_open=False*)

Create a superset of the current subset.

A *superset* is a manifold subset in which the current subset is included.

INPUT:

- *name* – name given to the superset
- *latex\_name* – (default: None) LaTeX symbol to denote the superset; if none are provided, it is set to *name*
- *is\_open* – (default: False) if True, the created subset is assumed to be open with respect to the manifold's topology

OUTPUT:

- the superset, as an instance of `ManifoldSubset` or of the derived class `TopologicalManifold` if *is\_open* is True

EXAMPLES:

Creating some superset of a given subset:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: a = M.subset('A')
sage: b = a.superset('B'); b
Subset B of the 2-dimensional topological manifold M
sage: b.list_of_subsets()
[Subset A of the 2-dimensional topological manifold M,
 Subset B of the 2-dimensional topological manifold M]
sage: a._supersets # random (set output)
```

```
{Subset B of the 2-dimensional topological manifold M,
 Subset A of the 2-dimensional topological manifold M,
 2-dimensional topological manifold M}
```

The superset of the whole manifold is itself:

```
sage: M.superset('SM') is M
True
```

Two supersets of a given subset are a priori different:

```
sage: c = a.superset('C')
sage: c == b
False
```

**union** (*other*, *name=None*, *latex\_name=None*)

Return the union of the current subset with another subset.

INPUT:

- *other* – another subset of the same manifold
- *name* – (default: None) name given to the union in the case the latter has to be created; the default is `self._name union other._name`
- *latex\_name* – (default: None) LaTeX symbol to denote the union in the case the latter has to be created; the default is built upon the symbol  $\cup$

OUTPUT:

- instance of *ManifoldSubset* representing the subset that is the union of the current subset with *other*

EXAMPLES:

Union of two subsets:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: a = M.subset('A')
sage: b = M.subset('B')
sage: c = a.union(b); c
Subset A_union_B of the 2-dimensional topological manifold M
sage: a._supersets # random (set output)
set([subset 'A_union_B' of the 2-dimensional manifold 'M',
      2-dimensional manifold 'M',
      subset 'A' of the 2-dimensional manifold 'M'])
sage: b._supersets # random (set output)
set([subset 'B' of the 2-dimensional manifold 'M',
      2-dimensional manifold 'M',
      subset 'A_union_B' of the 2-dimensional manifold 'M'])
sage: c._subsets # random (set output)
set([subset 'A_union_B' of the 2-dimensional manifold 'M',
      subset 'A' of the 2-dimensional manifold 'M',
      subset 'B' of the 2-dimensional manifold 'M'])
```

Some checks:

```
sage: a.is_subset(a.union(b))
True
sage: b.is_subset(a.union(b))
```

```

True
sage: a.union(b) is b.union(a)
True
sage: a.union(a.union(b)) is a.union(b)
True
sage: (a.union(b)).union(a) is a.union(b)
True
sage: a.union(M) is M
True
sage: M.union(a) is M
True

```

## 1.3 Manifold Structures

These classes encode the structure of a manifold.

AUTHORS:

- Travis Scrimshaw (2015-11-25): Initial version
- Eric Gourgoulhon (2015): add *DifferentialStructure* and *RealDifferentialStructure*

**class** `sage.manifolds.structure.DifferentialStructure`

Bases: `sage.misc.fast_methods.Singleton`

The structure of a differentiable manifold over a general topological field.

**chart**

alias of `DiffChart`

**homset**

alias of `DifferentiableManifoldHomset`

**scalar\_field\_algebra**

alias of `DiffScalarFieldAlgebra`

**subcategory** (*cat*)

Return the subcategory of *cat* corresponding to the structure of self.

EXAMPLES:

```

sage: from sage.manifolds.structure import DifferentialStructure
sage: from sage.categories.manifolds import Manifolds
sage: DifferentialStructure().subcategory(Manifolds(RR))
Category of manifolds over Real Field with 53 bits of precision

```

**class** `sage.manifolds.structure.RealDifferentialStructure`

Bases: `sage.misc.fast_methods.Singleton`

The structure of a differentiable manifold over  $\mathbf{R}$ .

**chart**

alias of `RealDiffChart`

**homset**

alias of `DifferentiableManifoldHomset`

**scalar\_field\_algebra**

alias of `DiffScalarFieldAlgebra`

**subcategory** (*cat*)

Return the subcategory of *cat* corresponding to the structure of self.

EXAMPLES:

```
sage: from sage.manifolds.structure import DifferentialStructure
sage: from sage.categories.manifolds import Manifolds
sage: DifferentialStructure().subcategory(Manifolds(RR))
Category of manifolds over Real Field with 53 bits of precision
```

**class** `sage.manifolds.structure.RealTopologicalStructure`

Bases: `sage.misc.fast_methods.Singleton`

The structure of a topological manifold over  $\mathbf{R}$ .

**chart**

alias of `RealChart`

**homset**

alias of `TopologicalManifoldHomset`

**scalar\_field\_algebra**

alias of `ScalarFieldAlgebra`

**subcategory** (*cat*)

Return the subcategory of *cat* corresponding to the structure of self.

EXAMPLES:

```
sage: from sage.manifolds.structure import RealTopologicalStructure
sage: from sage.categories.manifolds import Manifolds
sage: RealTopologicalStructure().subcategory(Manifolds(RR))
Category of manifolds over Real Field with 53 bits of precision
```

**class** `sage.manifolds.structure.TopologicalStructure`

Bases: `sage.misc.fast_methods.Singleton`

The structure of a topological manifold over a general topological field.

**chart**

alias of `Chart`

**homset**

alias of `TopologicalManifoldHomset`

**scalar\_field\_algebra**

alias of `ScalarFieldAlgebra`

**subcategory** (*cat*)

Return the subcategory of *cat* corresponding to the structure of self.

EXAMPLES:

```
sage: from sage.manifolds.structure import TopologicalStructure
sage: from sage.categories.manifolds import Manifolds
sage: TopologicalStructure().subcategory(Manifolds(RR))
Category of manifolds over Real Field with 53 bits of precision
```

## 1.4 Points of Topological Manifolds

The class `ManifoldPoint` implements points of a topological manifold.

A `ManifoldPoint` object can have coordinates in various charts defined on the manifold. Two points are declared equal if they have the same coordinates in the same chart.

AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2013-2015) : initial version

REFERENCES:

- [Lee2011]
- [Lee2013]

EXAMPLES:

Defining a point in  $\mathbb{R}^3$  by its spherical coordinates:

```
sage: M = Manifold(3, 'R^3', structure='topological')
sage: U = M.open_subset('U') # the complement of the half-plane (y=0, x>=0)
sage: c_spher.<r,th,ph> = U.chart(r'r:(0,+oo) th:(0,pi):\theta ph:(0,2*pi):\phi')
```

We construct the point in the coordinates in the default chart of `U` (`c_spher`):

```
sage: p = U((1, pi/2, pi), name='P')
sage: p
Point P on the 3-dimensional topological manifold R^3
sage: latex(p)
P
sage: p in U
True
sage: p.parent()
Open subset U of the 3-dimensional topological manifold R^3
sage: c_spher(p)
(1, 1/2*pi, pi)
sage: p.coordinates(c_spher) # equivalent to above
(1, 1/2*pi, pi)
```

Computing the coordinates of `p` in a new chart:

```
sage: c_cart.<x,y,z> = U.chart() # Cartesian coordinates on U
sage: spher_to_cart = c_spher.transition_map(c_cart,
.....: [r*sin(th)*cos(ph), r*sin(th)*sin(ph), r*cos(th)])
sage: c_cart(p) # evaluate P's Cartesian coordinates
(-1, 0, 0)
```

Points can be compared:

```
sage: p1 = U((1, pi/2, pi))
sage: p == p1
True
sage: q = U((1,2,3), chart=c_cart, name='Q') # point defined by its Cartesian_
↪coordinates
sage: p == q
False
```

**class** `sage.manifolds.point.ManifoldPoint` (*parent*, *coords=None*, *chart=None*, *name=None*, *latex\_name=None*, *check\_coords=True*)

Bases: `sage.structure.element.Element`

Point of a topological manifold.

This is a Sage *element* class, the corresponding *parent* class being *TopologicalManifold* or *ManifoldSubset*.

INPUT:

- *parent* – the manifold subset to which the point belongs
- *coords* – (default: `None`) the point coordinates (as a tuple or a list) in the chart *chart*
- *chart* – (default: `None`) chart in which the coordinates are given; if `None`, the coordinates are assumed to refer to the default chart of *parent*
- *name* – (default: `None`) name given to the point
- *latex\_name* – (default: `None`) LaTeX symbol to denote the point; if `None`, the LaTeX symbol is set to *name*
- *check\_coords* – (default: `True`) determines whether *coords* are valid coordinates for the chart *chart*; for symbolic coordinates, it is recommended to set *check\_coords* to `False`

EXAMPLES:

A point on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: (a, b) = var('a b') # generic coordinates for the point
sage: p = M.point((a, b), name='P'); p
Point P on the 2-dimensional topological manifold M
sage: p.coordinates() # coordinates of P in the subset's default chart
(a, b)
```

Since points are Sage *elements*, the *parent* of which being the subset on which they are defined, it is equivalent to write:

```
sage: p = M((a, b), name='P'); p
Point P on the 2-dimensional topological manifold M
```

A point is an element of the manifold subset in which it has been defined:

```
sage: p in M
True
sage: p.parent()
2-dimensional topological manifold M
sage: U = M.open_subset('U', coord_def={c_xy: x>0})
sage: q = U.point((2,1), name='q')
sage: q.parent()
Open subset U of the 2-dimensional topological manifold M
sage: q in U
True
sage: q in M
True
```

By default, the LaTeX symbol of the point is deduced from its name:

```
sage: latex(p)
P
```

But it can be set to any value:

```
sage: p = M.point((a, b), name='P', latex_name=r'\mathcal{P}')
sage: latex(p)
\mathcal{P}
```

Points can be drawn in 2D or 3D graphics thanks to the method `plot()`.

**add\_coord** (*coords*, *chart=None*)

Adds some coordinates in the specified chart.

The previous coordinates with respect to other charts are kept. To clear them, use `set_coord()` instead.

INPUT:

- *coords* – the point coordinates (as a tuple or a list)
- *chart* – (default: None) chart in which the coordinates are given; if none are provided, the coordinates are assumed to refer to the subset's default chart

**Warning:** If the point has already coordinates in other charts, it is the user's responsibility to make sure that the coordinates to be added are consistent with them.

EXAMPLES:

Setting coordinates to a point on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: p = M.point()
```

We give the point some coordinates in the manifold's default chart:

```
sage: p.add_coordinates((2,-3))
sage: p.coordinates()
(2, -3)
sage: X(p)
(2, -3)
```

A shortcut for `add_coordinates` is `add_coord`:

```
sage: p.add_coord((2,-3))
sage: p.coord()
(2, -3)
```

Let us introduce a second chart on the manifold:

```
sage: Y.<u,v> = M.chart()
sage: X_to_Y = X.transition_map(Y, [x+y, x-y])
```

If we add coordinates for *p* in chart *Y*, those in chart *X* are kept:

```
sage: p.add_coordinates((-1,5), chart=Y)
sage: p._coordinates # random (dictionary output)
{Chart (M, (u, v)): (-1, 5), Chart (M, (x, y)): (2, -3)}
```

On the contrary, with the method `set_coordinates()`, the coordinates in charts different from  $Y$  would be lost:

```
sage: p.set_coordinates((-1,5), chart=Y)
sage: p._coordinates
{Chart (M, (u, v)): (-1, 5)}
```

#### `add_coordinates` (*coords*, *chart=None*)

Adds some coordinates in the specified chart.

The previous coordinates with respect to other charts are kept. To clear them, use `set_coord()` instead.

INPUT:

- `coords` – the point coordinates (as a tuple or a list)
- `chart` – (default: `None`) chart in which the coordinates are given; if none are provided, the coordinates are assumed to refer to the subset's default chart

**Warning:** If the point has already coordinates in other charts, it is the user's responsibility to make sure that the coordinates to be added are consistent with them.

EXAMPLES:

Setting coordinates to a point on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: p = M.point()
```

We give the point some coordinates in the manifold's default chart:

```
sage: p.add_coordinates((2,-3))
sage: p.coordinates()
(2, -3)
sage: X(p)
(2, -3)
```

A shortcut for `add_coordinates` is `add_coord`:

```
sage: p.add_coord((2,-3))
sage: p.coord()
(2, -3)
```

Let us introduce a second chart on the manifold:

```
sage: Y.<u,v> = M.chart()
sage: X_to_Y = X.transition_map(Y, [x+y, x-y])
```

If we add coordinates for  $p$  in chart  $Y$ , those in chart  $X$  are kept:

```
sage: p.add_coordinates((-1,5), chart=Y)
sage: p._coordinates # random (dictionary output)
{Chart (M, (u, v)): (-1, 5), Chart (M, (x, y)): (2, -3)}
```

On the contrary, with the method `set_coordinates()`, the coordinates in charts different from  $Y$  would be lost:

```
sage: p.set_coordinates((-1,5), chart=Y)
sage: p._coordinates
{Chart (M, (u, v)): (-1, 5)}
```

**coord** (*chart=None, old\_chart=None*)

Return the point coordinates in the specified chart.

If these coordinates are not already known, they are computed from known ones by means of change-of-chart formulas.

An equivalent way to get the coordinates of a point is to let the chart acting on the point, i.e. if  $X$  is a chart and  $p$  a point, one has `p.coordinates(chart=X) == X(p)`.

INPUT:

- `chart` – (default: None) chart in which the coordinates are given; if none are provided, the coordinates are assumed to refer to the subset's default chart
- `old_chart` – (default: None) chart from which the coordinates in `chart` are to be computed; if None, a chart in which the point's coordinates are already known will be picked, privileging the subset's default chart

EXAMPLES:

Spherical coordinates of a point on  $\mathbf{R}^3$ :

```
sage: M = Manifold(3, 'M', structure='topological')
sage: c_spher.<r,th,ph> = M.chart(r'r:(0,+oo) th:(0,pi):\theta ph:(0,
↪2*pi):\phi') # spherical coordinates
sage: p = M.point((1, pi/2, pi))
sage: p.coordinates() # coordinates in the manifold's default chart
(1, 1/2*pi, pi)
```

Since the default chart of  $M$  is `c_spher`, it is equivalent to write:

```
sage: p.coordinates(c_spher)
(1, 1/2*pi, pi)
```

An alternative way to get the coordinates is to let the chart act on the point (from the very definition of a chart):

```
sage: c_spher(p)
(1, 1/2*pi, pi)
```

A shortcut for `coordinates` is `coord`:

```
sage: p.coord()
(1, 1/2*pi, pi)
```

Computing the Cartesian coordinates from the spherical ones:

```
sage: c_cart.<x,y,z> = M.chart() # Cartesian coordinates
sage: c_spher.transition_map(c_cart, [r*sin(th)*cos(ph),
....:                               r*sin(th)*sin(ph), r*cos(th)])
Change of coordinates from Chart (M, (r, th, ph)) to Chart (M, (x, y, z))
```

The computation is performed by means of the above change of coordinates:

```
sage: p.coord(c_cart)
(-1, 0, 0)
sage: p.coord(c_cart) == c_cart(p)
True
```

Coordinates of a point on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: (a, b) = var('a b') # generic coordinates for the point
sage: P = M.point((a, b), name='P')
```

Coordinates of P in the manifold's default chart:

```
sage: P.coord()
(a, b)
```

Coordinates of P in a new chart:

```
sage: c_uv.<u,v> = M.chart()
sage: ch_xy_uv = c_xy.transition_map(c_uv, [x-y, x+y])
sage: P.coord(c_uv)
(a - b, a + b)
```

Coordinates of P in a third chart:

```
sage: c_wz.<w,z> = M.chart()
sage: ch_uv_wz = c_uv.transition_map(c_wz, [u^3, v^3])
sage: P.coord(c_wz, old_chart=c_uv)
(a^3 - 3*a^2*b + 3*a*b^2 - b^3, a^3 + 3*a^2*b + 3*a*b^2 + b^3)
```

Actually, in the present case, it is not necessary to specify `old_chart='uv'`. Note that the first command erases all the coordinates except those in the chart `c_uv`:

```
sage: P.set_coord((a-b, a+b), c_uv)
sage: P._coordinates
{Chart (M, (u, v)): (a - b, a + b)}
sage: P.coord(c_wz)
(a^3 - 3*a^2*b + 3*a*b^2 - b^3, a^3 + 3*a^2*b + 3*a*b^2 + b^3)
sage: P._coordinates # random (dictionary output)
{Chart (M, (u, v)): (a - b, a + b),
 Chart (M, (w, z)): (a^3 - 3*a^2*b + 3*a*b^2 - b^3,
                   a^3 + 3*a^2*b + 3*a*b^2 + b^3)}
```

**coordinates** (*chart=None, old\_chart=None*)

Return the point coordinates in the specified chart.

If these coordinates are not already known, they are computed from known ones by means of change-of-chart formulas.

An equivalent way to get the coordinates of a point is to let the chart acting on the point, i.e. if `X` is a chart and `p` a point, one has `p.coordinates(chart=X) == X(p)`.

INPUT:

- `chart` – (default: `None`) chart in which the coordinates are given; if none are provided, the coordinates are assumed to refer to the subset's default chart

- `old_chart` – (default: `None`) chart from which the coordinates in `chart` are to be computed; if `None`, a chart in which the point's coordinates are already known will be picked, privileging the subset's default chart

EXAMPLES:

Spherical coordinates of a point on  $\mathbf{R}^3$ :

```
sage: M = Manifold(3, 'M', structure='topological')
sage: c_spher.<r,th,ph> = M.chart(r'r:(0,+oo) th:(0,pi):\theta ph:(0,
↔2*pi):\phi') # spherical coordinates
sage: p = M.point((1, pi/2, pi))
sage: p.coordinates() # coordinates in the manifold's default chart
(1, 1/2*pi, pi)
```

Since the default chart of  $M$  is `c_spher`, it is equivalent to write:

```
sage: p.coordinates(c_spher)
(1, 1/2*pi, pi)
```

An alternative way to get the coordinates is to let the chart act on the point (from the very definition of a chart):

```
sage: c_spher(p)
(1, 1/2*pi, pi)
```

A shortcut for coordinates is `coord`:

```
sage: p.coord()
(1, 1/2*pi, pi)
```

Computing the Cartesian coordinates from the spherical ones:

```
sage: c_cart.<x,y,z> = M.chart() # Cartesian coordinates
sage: c_spher.transition_map(c_cart, [r*sin(th)*cos(ph),
....:                               r*sin(th)*sin(ph), r*cos(th)])
Change of coordinates from Chart (M, (r, th, ph)) to Chart (M, (x, y, z))
```

The computation is performed by means of the above change of coordinates:

```
sage: p.coord(c_cart)
(-1, 0, 0)
sage: p.coord(c_cart) == c_cart(p)
True
```

Coordinates of a point on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: (a, b) = var('a b') # generic coordinates for the point
sage: P = M.point((a, b), name='P')
```

Coordinates of  $P$  in the manifold's default chart:

```
sage: P.coord()
(a, b)
```

Coordinates of  $P$  in a new chart:

```

sage: c_uv.<u,v> = M.chart()
sage: ch_xy_uv = c_uv.transition_map(c_uv, [x-y, x+y])
sage: P.coord(c_uv)
(a - b, a + b)

```

Coordinates of  $P$  in a third chart:

```

sage: c_wz.<w,z> = M.chart()
sage: ch_uv_wz = c_uv.transition_map(c_wz, [u^3, v^3])
sage: P.coord(c_wz, old_chart=c_uv)
(a^3 - 3*a^2*b + 3*a*b^2 - b^3, a^3 + 3*a^2*b + 3*a*b^2 + b^3)

```

Actually, in the present case, it is not necessary to specify `old_chart='uv'`. Note that the first command erases all the coordinates except those in the chart `c_uv`:

```

sage: P.set_coord((a-b, a+b), c_uv)
sage: P._coordinates
{Chart (M, (u, v)): (a - b, a + b)}
sage: P.coord(c_wz)
(a^3 - 3*a^2*b + 3*a*b^2 - b^3, a^3 + 3*a^2*b + 3*a*b^2 + b^3)
sage: P._coordinates # random (dictionary output)
{Chart (M, (u, v)): (a - b, a + b),
 Chart (M, (w, z)): (a^3 - 3*a^2*b + 3*a*b^2 - b^3,
                    a^3 + 3*a^2*b + 3*a*b^2 + b^3)}

```

**plot** (*chart=None, ambient\_coords=None, mapping=None, label=None, parameters=None, color='black', label\_offset=0.1, fontsize=10, label\_color=None, size=10, \*\*kws*)  
 For real manifolds, plot `self` in a Cartesian graph based on the coordinates of some ambient chart.

The point is drawn in terms of two (2D graphics) or three (3D graphics) coordinates of a given chart, called hereafter the *ambient chart*. The domain of the ambient chart must contain the point, or its image by a continuous manifold map  $\Phi$ .

INPUT:

- `chart` – (default: `None`) the ambient chart (see above); if `None`, the ambient chart is set the default chart of `self.parent()`
- `ambient_coords` – (default: `None`) tuple containing the 2 or 3 coordinates of the ambient chart in terms of which the plot is performed; if `None`, all the coordinates of the ambient chart are considered
- `mapping` – (default: `None`) *ContinuousMap*; continuous manifold map  $\Phi$  providing the link between the current point  $p$  and the ambient chart `chart`: the domain of `chart` must contain  $\Phi(p)$ ; if `None`, the identity map is assumed
- `label` – (default: `None`) label printed next to the point; if `None`, the point's name is used
- `parameters` – (default: `None`) dictionary giving the numerical values of the parameters that may appear in the point coordinates
- `size` – (default: 10) size of the point once drawn as a small disk or sphere
- `color` – (default: 'black') color of the point
- `label_color` – (default: `None`) color to print the label; if `None`, the value of `color` is used
- `fontsize` – (default: 10) size of the font used to print the label
- `label_offset` – (default: 0.1) determines the separation between the point and its label

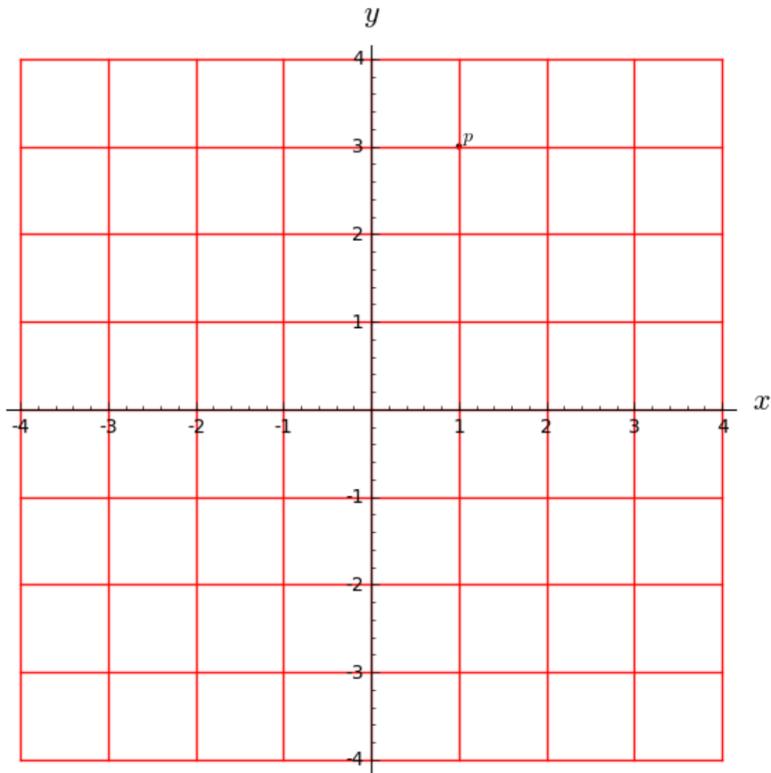
OUTPUT:

- a graphic object, either an instance of `Graphics` for a 2D plot (i.e. based on 2 coordinates of the ambient chart) or an instance of `Graphics3d` for a 3D plot (i.e. based on 3 coordinates of the ambient chart)

EXAMPLES:

Drawing a point on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: p = M.point((1,3), name='p')
sage: g = p.plot(X)
sage: print(g)
Graphics object consisting of 2 graphics primitives
sage: gX = X.plot(max_range=4) # plot of the coordinate grid
sage: g + gX # display of the point atop the coordinate grid
Graphics object consisting of 20 graphics primitives
```



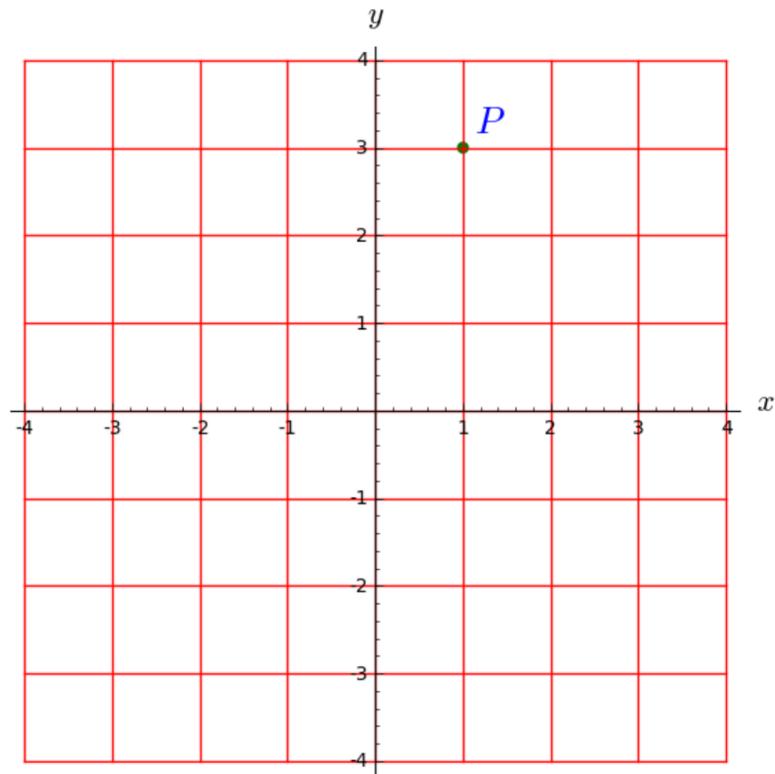
Actually, since `X` is the default chart of the open set in which `p` has been defined, it can be skipped in the arguments of `plot`:

```
sage: g = p.plot()
sage: g + gX
Graphics object consisting of 20 graphics primitives
```

Call with some options:

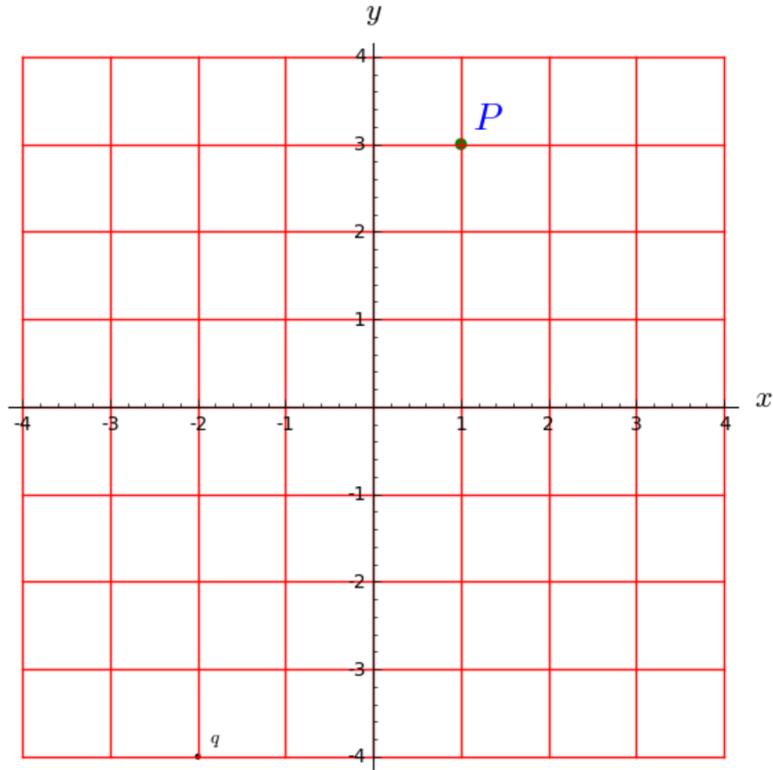
```
sage: g = p.plot(chart=X, size=40, color='green', label='$P$',
....:             label_color='blue', fontsize=20, label_offset=0.3)
```

```
sage: g + gX
Graphics object consisting of 20 graphics primitives
```



Use of the parameters option to set a numerical value of some symbolic variable:

```
sage: a = var('a')
sage: q = M.point((a, 2*a), name='q')
sage: qq = q.plot(parameters={a:-2}, label_offset=0.2)
sage: g + gX + qq
Graphics object consisting of 22 graphics primitives
```



The numerical value is used only for the plot:

```
sage: q.coord()
(a, 2*a)
```

Drawing a point on a 3-dimensional manifold:

```
sage: M = Manifold(3, 'M', structure='topological')
sage: X.<x,y,z> = M.chart()
sage: p = M.point((2,1,3), name='p')
sage: g = p.plot()
sage: print(g)
Graphics3d Object
sage: gX = X.plot(number_values=5) # coordinate mesh cube
sage: g + gX # display of the point atop the coordinate mesh
Graphics3d Object
```

Call with some options:

```
sage: g = p.plot(chart=X, size=40, color='green', label='P_1',
....:           label_color='blue', fontsize=20, label_offset=0.3)
sage: g + gX
Graphics3d Object
```

An example of plot via a mapping: plot of a point on a 2-sphere viewed in the 3-dimensional space M:

```
sage: S2 = Manifold(2, 'S^2', structure='topological')
sage: U = S2.open_subset('U') # the open set covered by spherical coord.
sage: XS.<th,ph> = U.chart(r'th:(0,pi):\theta ph:(0,2*pi):\phi')
```

```

sage: p = U.point((pi/4, pi/8), name='p')
sage: F = S2.continuous_map(M, {(XS, X): [sin(th)*cos(ph),
....:                               sin(th)*sin(ph), cos(th)]}, name='F')
sage: F.display()
F: S^2 --> M
on U: (th, ph) |--> (x, y, z) = (cos(ph)*sin(th), sin(ph)*sin(th), cos(th))
sage: g = p.plot(chart=X, mapping=F)
sage: gS2 = XS.plot(chart=X, mapping=F, number_values=9)
sage: g + gS2
Graphics3d Object

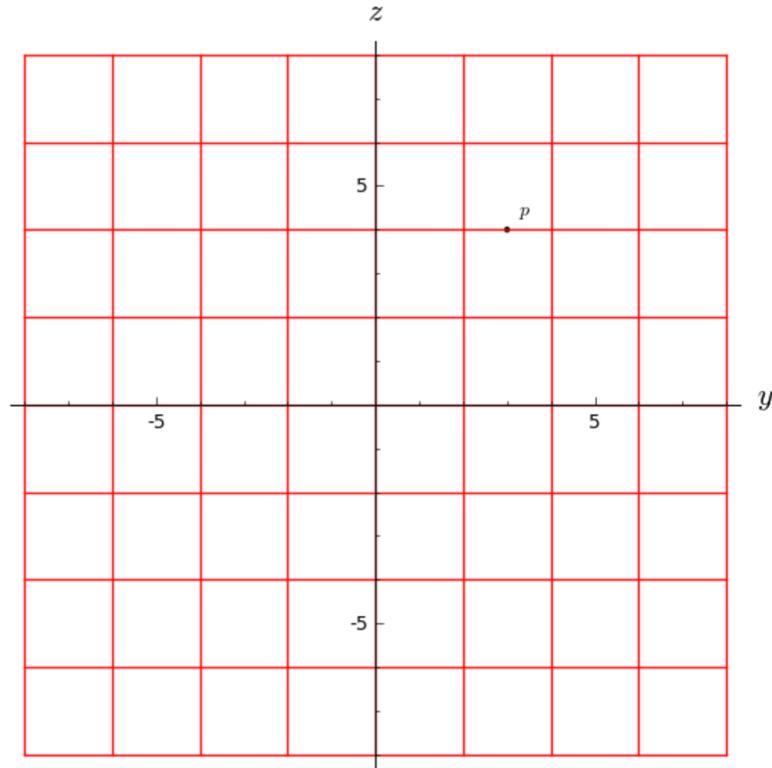
```

Use of the option `ambient_coords` for plots on a 4-dimensional manifold:

```

sage: M = Manifold(4, 'M', structure='topological')
sage: X.<t,x,y,z> = M.chart()
sage: p = M.point((1,2,3,4), name='p')
sage: g = p.plot(X, ambient_coords=(t,x,y), label_offset=0.4) # the_
↳coordinate z is skipped
sage: gX = X.plot(X, ambient_coords=(t,x,y), number_values=5) # long time
sage: g + gX # 3D plot # long time
Graphics3d Object
sage: g = p.plot(X, ambient_coords=(t,y,z), label_offset=0.4) # the_
↳coordinate x is skipped
sage: gX = X.plot(X, ambient_coords=(t,y,z), number_values=5) # long time
sage: g + gX # 3D plot # long time
Graphics3d Object
sage: g = p.plot(X, ambient_coords=(y,z), label_offset=0.4) # the_
↳coordinates t and x are skipped
sage: gX = X.plot(X, ambient_coords=(y,z))
sage: g + gX # 2D plot
Graphics object consisting of 20 graphics primitives

```



**set\_coord**(*coords*, *chart=None*)

Sets the point coordinates in the specified chart.

Coordinates with respect to other charts are deleted, in order to avoid any inconsistency. To keep them, use the method `add_coord()` instead.

INPUT:

- *coords* – the point coordinates (as a tuple or a list)
- *chart* – (default: `None`) chart in which the coordinates are given; if none are provided, the coordinates are assumed to refer to the subset's default chart

EXAMPLES:

Setting coordinates to a point on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: p = M.point()
```

We set the coordinates in the manifold's default chart:

```
sage: p.set_coordinates((2,-3))
sage: p.coordinates()
(2, -3)
sage: X(p)
(2, -3)
```

A shortcut for `set_coordinates` is `set_coord`:

```
sage: p.set_coord((2, -3))
sage: p.coord()
(2, -3)
```

Let us introduce a second chart on the manifold:

```
sage: Y.<u,v> = M.chart()
sage: X_to_Y = X.transition_map(Y, [x+y, x-y])
```

If we set the coordinates of  $p$  in chart  $Y$ , those in chart  $X$  are lost:

```
sage: Y(p)
(-1, 5)
sage: p.set_coord(Y(p), chart=Y)
sage: p._coordinates
{Chart (M, (u, v)): (-1, 5)}
```

#### **set\_coordinates** (*coords*, *chart=None*)

Sets the point coordinates in the specified chart.

Coordinates with respect to other charts are deleted, in order to avoid any inconsistency. To keep them, use the method `add_coord()` instead.

INPUT:

- *coords* – the point coordinates (as a tuple or a list)
- *chart* – (default: `None`) chart in which the coordinates are given; if none are provided, the coordinates are assumed to refer to the subset's default chart

EXAMPLES:

Setting coordinates to a point on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: p = M.point()
```

We set the coordinates in the manifold's default chart:

```
sage: p.set_coordinates((2, -3))
sage: p.coordinates()
(2, -3)
sage: X(p)
(2, -3)
```

A shortcut for `set_coordinates` is `set_coord`:

```
sage: p.set_coord((2, -3))
sage: p.coord()
(2, -3)
```

Let us introduce a second chart on the manifold:

```
sage: Y.<u,v> = M.chart()
sage: X_to_Y = X.transition_map(Y, [x+y, x-y])
```

If we set the coordinates of  $p$  in chart  $Y$ , those in chart  $X$  are lost:

```
sage: Y(p)
(-1, 5)
sage: p.set_coord(Y(p), chart=Y)
sage: p._coordinates
{Chart (M, (u, v)): (-1, 5)}
```

## 1.5 Coordinate Charts

### 1.5.1 Coordinate Charts

The class `Chart` implements coordinate charts on a topological manifold over a topological field  $K$ . The subclass `RealChart` is devoted to the case  $K = \mathbf{R}$ , for which the concept of coordinate range is meaningful. Moreover, `RealChart` is endowed with some plotting capabilities (cf. method `plot()`).

Transition maps between charts are implemented via the class `CoordChange`.

AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2013-2015) : initial version
- Travis Scrimshaw (2015): review tweaks

REFERENCES:

- Chap. 2 of [Lee2011]
- Chap. 1 of [Lee2013]

**class** `sage.manifolds.chart.Chart` (*domain, coordinates='', names=None*)

Bases: `sage.structure.unique_representation.UniqueRepresentation`, `sage.structure.sage_object.SageObject`

Chart on a topological manifold.

Given a topological manifold  $M$  of dimension  $n$  over a topological field  $K$ , a *chart* on  $M$  is a pair  $(U, \varphi)$ , where  $U$  is an open subset of  $M$  and  $\varphi : U \rightarrow V \subset K^n$  is a homeomorphism from  $U$  to an open subset  $V$  of  $K^n$ .

The components  $(x^1, \dots, x^n)$  of  $\varphi$ , defined by  $\varphi(p) = (x^1(p), \dots, x^n(p)) \in K^n$  for any point  $p \in U$ , are called the *coordinates* of the chart  $(U, \varphi)$ .

INPUT:

- `domain` – open subset  $U$  on which the chart is defined (must be an instance of `TopologicalManifold`)
- `coordinates` – (default: '' (empty string)) the string defining the coordinate symbols, see below
- `names` – (default: None) unused argument, except if `coordinates` is not provided; it must then be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator `<, >` is used)

The string `coordinates` has the space ' ' as a separator and each item has at most two fields, separated by a colon (:):

1. the coordinate symbol (a letter or a few letters);
2. (optional) the LaTeX spelling of the coordinate, if not provided the coordinate symbol given in the first field will be used.

If it contains any LaTeX expression, the string `coordinates` must be declared with the prefix 'r' (for "raw") to allow for a proper treatment of LaTeX's backslash character (see examples below). If no LaTeX spelling is to

be set for any coordinate, the argument `coordinates` can be omitted when the shortcut operator `<, >` is used via Sage preparser (see examples below).

EXAMPLES:

A chart on a complex 2-dimensional topological manifold:

```
sage: M = Manifold(2, 'M', field='complex', structure='topological')
sage: X = M.chart('x y'); X
Chart (M, (x, y))
sage: latex(X)
\left(M, (x, y)\right)
sage: type(X)
<class 'sage.manifolds.chart.Chart'>
```

To manipulate the coordinates  $(x, y)$  as global variables, one has to set:

```
sage: x, y = X[:]
```

However, a shortcut is to use the declarator `<x, y>` in the left-hand side of the chart declaration (there is then no need to pass the string `'x y'` to `chart()`):

```
sage: M = Manifold(2, 'M', field='complex', structure='topological')
sage: X.<x,y> = M.chart(); X
Chart (M, (x, y))
```

The coordinates are then immediately accessible:

```
sage: y
y
sage: x is X[0] and y is X[1]
True
```

Note that `x` and `y` declared in `<x, y>` are mere Python variable names and do not have to coincide with the coordinate symbols; for instance, one may write:

```
sage: M = Manifold(2, 'M', field='complex', structure='topological')
sage: X.<x1,y1> = M.chart('x y'); X
Chart (M, (x, y))
```

Then `y` is not known as a global Python variable and the coordinate  $y$  is accessible only through the global variable `y1`:

```
sage: y1
y
sage: latex(y1)
y
sage: y1 is X[1]
True
```

However, having the name of the Python variable coincide with the coordinate symbol is quite convenient; so it is recommended to declare:

```
sage: M = Manifold(2, 'M', field='complex', structure='topological')
sage: X.<x,y> = M.chart()
```

In the above example, the chart `X` covers entirely the manifold `M`:

```
sage: X.domain()
Complex 2-dimensional topological manifold M
```

Of course, one may declare a chart only on an open subset of M:

```
sage: U = M.open_subset('U')
sage: Y.<z1, z2> = U.chart(r'z1:\zeta_1 z2:\zeta_2'); Y
Chart (U, (z1, z2))
sage: Y.domain()
Open subset U of the Complex 2-dimensional topological manifold M
```

In the above declaration, we have also specified some LaTeX writing of the coordinates different from the text one:

```
sage: latex(z1)
{\zeta_1}
```

Note the prefix `r` in front of the string `r'z1:\zeta_1 z2:\zeta_2'`; it makes sure that the backslash character is treated as an ordinary character, to be passed to the LaTeX interpreter.

Coordinates are Sage symbolic variables (see `sage.symbolic.expression`):

```
sage: type(z1)
<type 'sage.symbolic.expression.Expression'>
```

In addition to the Python variable name provided in the operator `<., .>`, the coordinates are accessible by their indices:

```
sage: Y[0], Y[1]
(z1, z2)
```

The index range is that declared during the creation of the manifold. By default, it starts at 0, but this can be changed via the parameter `start_index`:

```
sage: M1 = Manifold(2, 'M_1', field='complex', structure='topological',
....:                 start_index=1)
sage: Z.<u,v> = M1.chart()
sage: Z[1], Z[2]
(u, v)
```

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

```
sage: Y[: ]
(z1, z2)
```

Some partial sets of coordinates:

```
sage: Y[:1]
(z1,)
sage: Y[1:]
(z2,)
```

Each constructed chart is automatically added to the manifold's user atlas:

```
sage: M.atlas()
[Chart (M, (x, y)), Chart (U, (z1, z2))]
```

and to the atlas of the chart's domain:

```
sage: U.atlas()
[Chart (U, (z1, z2))]
```

Manifold subsets have a *default chart*, which, unless changed via the method `set_default_chart()`, is the first defined chart on the subset (or on an open subset of it):

```
sage: M.default_chart()
Chart (M, (x, y))
sage: U.default_chart()
Chart (U, (z1, z2))
```

The default charts are not privileged charts on the manifold, but rather charts whose name can be skipped in the argument list of functions having an optional `chart=` argument.

The chart map  $\varphi$  acting on a point is obtained by passing it as an input to the map:

```
sage: p = M.point((1+i, 2), chart=X); p
Point on the Complex 2-dimensional topological manifold M
sage: X(p)
(I + 1, 2)
sage: X(p) == p.coord(X)
True
```

**See also:**

`sage.manifolds.chart.RealChart` for charts on topological manifolds over  $\mathbf{R}$ .

**add\_restrictions** (*restrictions*)

Add some restrictions on the coordinates.

INPUT:

- *restrictions* – list of restrictions on the coordinates, in addition to the ranges declared by the intervals specified in the chart constructor

A restriction can be any symbolic equality or inequality involving the coordinates, such as  $x > y$  or  $x^2 + y^2 \neq 0$ . The items of the list *restrictions* are combined with the `and` operator; if some restrictions are to be combined with the `or` operator instead, they have to be passed as a tuple in some single item of the list *restrictions*. For example:

```
restrictions = [x > y, (x != 0, y != 0), z^2 < x]
```

means  $(x > y)$  and  $((x \neq 0) \text{ or } (y \neq 0))$  and  $(z^2 < x)$ . If the list *restrictions* contains only one item, this item can be passed as such, i.e. writing  $x > y$  instead of the single element list  $[x > y]$ .

EXAMPLES:

```
sage: M = Manifold(2, 'M', field='complex', structure='topological')
sage: X.<x,y> = M.chart()
sage: X.add_restrictions(abs(x) > 1)
sage: X.valid_coordinates(2+i, 1)
True
sage: X.valid_coordinates(i, 1)
False
```

**domain** ()

Return the open subset on which the chart is defined.

EXAMPLES:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: X.domain()
2-dimensional topological manifold M
sage: U = M.open_subset('U')
sage: Y.<u,v> = U.chart()
sage: Y.domain()
Open subset U of the 2-dimensional topological manifold M

```

**function** (*expression*)

Define a coordinate function to the base field.

If the current chart belongs to the atlas of a  $n$ -dimensional manifold over a topological field  $K$ , a *coordinate function* is a map

$$f: \begin{array}{ccc} V \subset K^n & \longrightarrow & K \\ (x^1, \dots, x^n) & \longmapsto & f(x^1, \dots, x^n), \end{array}$$

where  $V$  is the chart codomain and  $(x^1, \dots, x^n)$  are the chart coordinates.

The coordinate function can be either a symbolic one or a numerical one, depending on the parameter expression (see below).

See *CoordFunction* and *CoordFunctionSymb* for a complete documentation.

INPUT:

- *expression* – material defining the coordinate function; it can be either:
  - a symbolic expression involving the chart coordinates, to represent  $f(x^1, \dots, x^n)$
  - a string representing the name of a file where the data to construct a numerical coordinate function is stored

OUTPUT:

- instance of a subclass of the base class *CoordFunction* representing the coordinate function  $f$ ; this is *CoordFunctionSymb* if *expression* is a symbolic expression.

EXAMPLES:

A symbolic coordinate function:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(sin(x*y))
sage: f
sin(x*y)
sage: type(f)
<class 'sage.manifolds.coord_func_symb.CoordFunctionSymbRing_with_category.
  element_class'>
sage: f.display()
(x, y) |--> sin(x*y)
sage: f(2,3)
sin(6)

```

**function\_ring** ()

Return the ring of coordinate functions on self.

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: X.function_ring()
Ring of coordinate functions on Chart (M, (x, y))
```

**manifold()**

Return the manifold on which the chart is defined.

## EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: U = M.open_subset('U')
sage: X.<x,y> = U.chart()
sage: X.manifold()
2-dimensional topological manifold M
sage: X.domain()
Open subset U of the 2-dimensional topological manifold M
```

**multifunction(\*expressions)**

Define a coordinate function to some Cartesian power of the base field.

If  $n$  and  $m$  are two positive integers and  $(U, \varphi)$  is a chart on a topological manifold  $M$  of dimension  $n$  over a topological field  $K$ , a *multi-coordinate function* associated to  $(U, \varphi)$  is a map

$$f: V \subset K^n \longrightarrow K^m \\ (x^1, \dots, x^n) \longmapsto (f_1(x^1, \dots, x^n), \dots, f_m(x^1, \dots, x^n)),$$

where  $V$  is the codomain of  $\varphi$ . In other words,  $f$  is a  $K^m$ -valued function of the coordinates associated to the chart  $(U, \varphi)$ .

See [MultiCoordFunction](#) for a complete documentation.

## INPUT:

- `expressions` – list (or tuple) of  $m$  elements to construct the coordinate functions  $f_i$  ( $1 \leq i \leq m$ ); for symbolic coordinate functions, this must be symbolic expressions involving the chart coordinates, while for numerical coordinate functions, this must be data file names

## OUTPUT:

- a [MultiCoordFunction](#) representing  $f$

## EXAMPLES:

Function of two coordinates with values in  $\mathbf{R}^3$ :

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.multifunction(x+y, sin(x*y), x^2 + 3*y); f
Coordinate functions (x + y, sin(x*y), x^2 + 3*y) on the Chart (M, (x, y))
sage: f(2,3)
(5, sin(6), 13)
```

**one\_function()**

Return the constant function of the coordinates equal to one.

If the current chart belongs to the atlas of a  $n$ -dimensional manifold over a topological field  $K$ , the “one” coordinate function is the map

$$f: V \subset K^n \longrightarrow K \\ (x^1, \dots, x^n) \longmapsto 1,$$

where  $V$  is the chart codomain.

See class `CoordFunctionSymb` for a complete documentation.

OUTPUT:

- a `CoordFunctionSymb` representing the one coordinate function  $f$

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: X.one_function()
1
sage: X.one_function().display()
(x, y) |--> 1
sage: type(X.one_function())
<class 'sage.manifolds.coord_func_symb.CoordFunctionSymbRing_with_category.
↪element_class'>
```

The result is cached:

```
sage: X.one_function() is X.one_function()
True
```

One function on a p-adic manifold:

```
sage: M = Manifold(2, 'M', structure='topological', field=Qp(5)); M
2-dimensional topological manifold M over the 5-adic Field with
capped relative precision 20
sage: X.<x,y> = M.chart()
sage: X.one_function()
1 + O(5^20)
sage: X.one_function().display()
(x, y) |--> 1 + O(5^20)
```

**restrict** (*subset*, *restrictions=None*)

Return the restriction of `self` to some open subset of its domain.

If the current chart is  $(U, \varphi)$ , a *restriction* (or *subchart*) is a chart  $(V, \psi)$  such that  $V \subset U$  and  $\psi = \varphi|_V$ .

If such subchart has not been defined yet, it is constructed here.

The coordinates of the subchart bare the same names as the coordinates of the current chart.

INPUT:

- *subset* – open subset  $V$  of the chart domain  $U$  (must be an instance of `TopologicalManifold`)
- *restrictions* – (default: `None`) list of coordinate restrictions defining the subset  $V$

A restriction can be any symbolic equality or inequality involving the coordinates, such as  $x > y$  or  $x^2 + y^2 \neq 0$ . The items of the list `restrictions` are combined with the `and` operator; if some restrictions are to be combined with the `or` operator instead, they have to be passed as a tuple in some single item of the list `restrictions`. For example:

```
restrictions = [x > y, (x != 0, y != 0), z^2 < x]
```

means  $(x > y)$  and  $((x \neq 0) \text{ or } (y \neq 0))$  and  $(z^2 < x)$ . If the list `restrictions` contains only one item, this item can be passed as such, i.e. writing  $x > y$  instead of the single element list  $[x > y]$ .

OUTPUT:

- `chart` ( $V, \psi$ ) as a *Chart*

EXAMPLES:

Coordinates on the unit open ball of  $\mathbf{C}^2$  as a subchart of the global coordinates of  $\mathbf{C}^2$ :

```
sage: M = Manifold(2, 'C^2', field='complex', structure='topological')
sage: X.<z1, z2> = M.chart()
sage: B = M.open_subset('B')
sage: X_B = X.restrict(B, abs(z1)^2 + abs(z2)^2 < 1); X_B
Chart (B, (z1, z2))
```

**transition\_map** (*other*, *transformations*, *intersection\_name=None*, *restrictions1=None*, *restrictions2=None*)

Construct the transition map between the current chart,  $(U, \varphi)$  say, and another one,  $(V, \psi)$  say.

If  $n$  is the manifold's dimension, the *transition map* is the map

$$\psi \circ \varphi^{-1} : \varphi(U \cap V) \subset K^n \rightarrow \psi(U \cap V) \subset K^n,$$

where  $K$  is the manifold's base field. In other words, the transition map expresses the coordinates  $(y^1, \dots, y^n)$  of  $(V, \psi)$  in terms of the coordinates  $(x^1, \dots, x^n)$  of  $(U, \varphi)$  on the open subset where the two charts intersect, i.e. on  $U \cap V$ .

INPUT:

- other* – the chart  $(V, \psi)$
- transformations* – tuple (or list)  $(Y_1, \dots, Y_n)$ , where  $Y_i$  is the symbolic expression of the coordinate  $y^i$  in terms of the coordinates  $(x^1, \dots, x^n)$
- intersection\_name* – (default: None) name to be given to the subset  $U \cap V$  if the latter differs from  $U$  or  $V$
- restrictions1* – (default: None) list of conditions on the coordinates of the current chart that define  $U \cap V$  if the latter differs from  $U$
- restrictions2* – (default: None) list of conditions on the coordinates of the chart  $(V, \psi)$  that define  $U \cap V$  if the latter differs from  $V$

A restriction can be any symbolic equality or inequality involving the coordinates, such as  $x > y$  or  $x^2 + y^2 \neq 0$ . The items of the list *restrictions* are combined with the `and` operator; if some restrictions are to be combined with the `or` operator instead, they have to be passed as a tuple in some single item of the list *restrictions*. For example:

```
restrictions = [x > y, (x != 0, y != 0), z^2 < x]
```

means  $(x > y)$  and  $((x \neq 0) \text{ or } (y \neq 0))$  and  $(z^2 < x)$ . If the list *restrictions* contains only one item, this item can be passed as such, i.e. writing  $x > y$  instead of the single element list  $[x > y]$ .

OUTPUT:

- the transition map  $\psi \circ \varphi^{-1}$  defined on  $U \cap V$  as a *CoordChange*

EXAMPLES:

Transition map between two stereographic charts on the circle  $S^1$ :

```
sage: M = Manifold(1, 'S^1', structure='topological')
sage: U = M.open_subset('U') # Complement of the North pole
sage: cU.<x> = U.chart() # Stereographic chart from the North pole
sage: V = M.open_subset('V') # Complement of the South pole
```

```

sage: cV.<y> = V.chart() # Stereographic chart from the South pole
sage: M.declare_union(U,V) # S^1 is the union of U and V
sage: trans = cU.transition_map(cV, 1/x, intersection_name='W',
....:                          restrictions1= x!=0, restrictions2 = y!=0)
sage: trans
Change of coordinates from Chart (W, (x,)) to Chart (W, (y,))
sage: trans.display()
y = 1/x

```

The subset  $W$ , intersection of  $U$  and  $V$ , has been created by `transition_map()`:

```

sage: M.list_of_subsets()
[1-dimensional topological manifold S^1,
 Open subset U of the 1-dimensional topological manifold S^1,
 Open subset V of the 1-dimensional topological manifold S^1,
 Open subset W of the 1-dimensional topological manifold S^1]
sage: W = M.list_of_subsets()[3]
sage: W is U.intersection(V)
True
sage: M.atlas()
[Chart (U, (x,)), Chart (V, (y,)), Chart (W, (x,)), Chart (W, (y,))]

```

Transition map between the spherical chart and the Cartesian one on  $\mathbf{R}^2$ :

```

sage: M = Manifold(2, 'R^2', structure='topological')
sage: c_cart.<x,y> = M.chart()
sage: U = M.open_subset('U') # the complement of the half line {y=0, x >= 0}
sage: c_spher.<r,phi> = U.chart(r'r:(0,+oo) phi:(0,2*pi):\phi')
sage: trans = c_spher.transition_map(c_cart, (r*cos(phi), r*sin(phi)),
....:                               restrictions2=(y!=0, x<0))
sage: trans
Change of coordinates from Chart (U, (r, phi)) to Chart (U, (x, y))
sage: trans.display()
x = r*cos(phi)
y = r*sin(phi)

```

In this case, no new subset has been created since  $U \cap M = U$ :

```

sage: M.list_of_subsets()
[2-dimensional topological manifold R^2,
 Open subset U of the 2-dimensional topological manifold R^2]

```

but a new chart has been created:  $(U, (x, y))$ :

```

sage: M.atlas()
[Chart (R^2, (x, y)), Chart (U, (r, phi)), Chart (U, (x, y))]

```

### **valid\_coordinates** (\*coordinates, \*\*kwds)

Check whether a tuple of coordinates can be the coordinates of a point in the chart domain.

INPUT:

- \*coordinates – coordinate values
- \*\*kwds – options:
  - parameters=None, dictionary to set numerical values to some parameters (see example below)

OUTPUT:

- True if the coordinate values are admissible in the chart image, False otherwise

EXAMPLES:

```
sage: M = Manifold(2, 'M', field='complex', structure='topological')
sage: X.<x,y> = M.chart()
sage: X.add_restrictions([abs(x)<1, y!=0])
sage: X.valid_coordinates(0, i)
True
sage: X.valid_coordinates(i, 1)
False
sage: X.valid_coordinates(i/2, 1)
True
sage: X.valid_coordinates(i/2, 0)
False
sage: X.valid_coordinates(2, 0)
False
```

Example of use with the keyword parameters to set a specific value to a parameter appearing in the coordinate restrictions:

```
sage: var('a') # the parameter is a symbolic variable
a
sage: Y.<u,v> = M.chart()
sage: Y.add_restrictions(abs(v)<a)
sage: Y.valid_coordinates(1, i, parameters={a: 2}) # setting a=2
True
sage: Y.valid_coordinates(1, 2*i, parameters={a: 2})
False
```

### **zero\_function()**

Return the zero function of the coordinates.

If the current chart belongs to the atlas of a  $n$ -dimensional manifold over a topological field  $K$ , the zero coordinate function is the map

$$f: \begin{array}{l} V \subset K^n \longrightarrow K \\ (x^1, \dots, x^n) \longmapsto 0, \end{array}$$

where  $V$  is the chart codomain.

See class `CoordFunctionSymb` for a complete documentation.

OUTPUT:

- a `CoordFunctionSymb` representing the zero coordinate function  $f$

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: X.zero_function()
0
sage: X.zero_function().display()
(x, y) |--> 0
sage: type(X.zero_function())
<class 'sage.manifolds.coord_func_symb.CoordFunctionSymbRing_with_category.
  element_class'>
```

The result is cached:

```
sage: X.zero_function() is X.zero_function()
True
```

Zero function on a p-adic manifold:

```
sage: M = Manifold(2, 'M', structure='topological', field=Qp(5)); M
2-dimensional topological manifold M over the 5-adic Field with
capped relative precision 20
sage: X.<x,y> = M.chart()
sage: X.zero_function()
0
sage: X.zero_function().display()
(x, y) |--> 0
```

**class** sage.manifolds.chart.CoordChange (chart1, chart2, \*transformations)

Bases: sage.structure.sage\_object.SageObject

Transition map between two charts of a topological manifold.

Giving two coordinate charts  $(U, \varphi)$  and  $(V, \psi)$  on a topological manifold  $M$  of dimension  $n$  over a topological field  $K$ , the *transition map from  $(U, \varphi)$  to  $(V, \psi)$*  is the map

$$\psi \circ \varphi^{-1} : \varphi(U \cap V) \subset K^n \rightarrow \psi(U \cap V) \subset K^n.$$

In other words, the transition map  $\psi \circ \varphi^{-1}$  expresses the coordinates  $(y^1, \dots, y^n)$  of  $(V, \psi)$  in terms of the coordinates  $(x^1, \dots, x^n)$  of  $(U, \varphi)$  on the open subset where the two charts intersect, i.e. on  $U \cap V$ .

INPUT:

- chart1 – chart  $(U, \varphi)$
- chart2 – chart  $(V, \psi)$
- transformations – tuple (or list)  $(Y_1, \dots, Y_2)$ , where  $Y_i$  is the symbolic expression of the coordinate  $y^i$  in terms of the coordinates  $(x^1, \dots, x^n)$

EXAMPLES:

Transition map on a 2-dimensional topological manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: Y.<u,v> = M.chart()
sage: X_to_Y = X.transition_map(Y, [x+y, x-y])
sage: X_to_Y
Change of coordinates from Chart (M, (x, y)) to Chart (M, (u, v))
sage: type(X_to_Y)
<class 'sage.manifolds.chart.CoordChange'>
sage: X_to_Y.display()
u = x + y
v = x - y
```

**disp()**

Display of the coordinate transformation.

The output is either text-formatted (console mode) or LaTeX-formatted (notebook mode).

EXAMPLES:

From spherical coordinates to Cartesian ones in the plane:

```

sage: M = Manifold(2, 'R^2', structure='topological')
sage: U = M.open_subset('U') # the complement of the half line {y=0, x>= 0}
sage: c_cart.<x,y> = U.chart()
sage: c_spher.<r,ph> = U.chart(r'r:(0,+oo) ph:(0,2*pi):\phi')
sage: spher_to_cart = c_spher.transition_map(c_cart, [r*cos(ph), r*sin(ph)])
sage: spher_to_cart.display()
x = r*cos(ph)
y = r*sin(ph)
sage: latex(spher_to_cart.display())
\left\{\begin{array}{lcl} x & = & r \cos\left(\phi\right) \\ y & = & r \sin\left(\phi\right) \end{array}\right.

```

A shortcut is `disp()`:

```

sage: spher_to_cart.disp()
x = r*cos(ph)
y = r*sin(ph)

```

### **display()**

Display of the coordinate transformation.

The output is either text-formatted (console mode) or LaTeX-formatted (notebook mode).

EXAMPLES:

From spherical coordinates to Cartesian ones in the plane:

```

sage: M = Manifold(2, 'R^2', structure='topological')
sage: U = M.open_subset('U') # the complement of the half line {y=0, x>= 0}
sage: c_cart.<x,y> = U.chart()
sage: c_spher.<r,ph> = U.chart(r'r:(0,+oo) ph:(0,2*pi):\phi')
sage: spher_to_cart = c_spher.transition_map(c_cart, [r*cos(ph), r*sin(ph)])
sage: spher_to_cart.display()
x = r*cos(ph)
y = r*sin(ph)
sage: latex(spher_to_cart.display())
\left\{\begin{array}{lcl} x & = & r \cos\left(\phi\right) \\ y & = & r \sin\left(\phi\right) \end{array}\right.

```

A shortcut is `disp()`:

```

sage: spher_to_cart.disp()
x = r*cos(ph)
y = r*sin(ph)

```

### **inverse()**

Compute the inverse coordinate transformation.

OUTPUT:

- an instance of *CoordChange* representing the inverse of the current coordinate transformation

EXAMPLES:

Inverse of a coordinate transformation corresponding to a  $\pi/3$ -rotation in the plane:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: c_uv.<u,v> = M.chart()
sage: xy_to_uv = c_xy.transition_map(c_uv, ((x - sqrt(3)*y)/2, (sqrt(3)*x +
↪ y)/2))

```

```

sage: M.coord_changes()
{(Chart (M, (x, y)),
 Chart (M, (u, v))): Change of coordinates from Chart (M, (x, y)) to Chart
↳(M, (u, v))}
sage: uv_to_xy = xy_to_uv.inverse(); uv_to_xy
Change of coordinates from Chart (M, (u, v)) to Chart (M, (x, y))
sage: uv_to_xy.display()
x = 1/2*sqrt(3)*v + 1/2*u
y = -1/2*sqrt(3)*u + 1/2*v
sage: M.coord_changes() # random (dictionary output)
{(Chart (M, (u, v)),
 Chart (M, (x, y))): Change of coordinates from Chart (M, (u, v)) to Chart
↳(M, (x, y)),
 (Chart (M, (x, y)),
 Chart (M, (u, v))): Change of coordinates from Chart (M, (x, y)) to Chart
↳(M, (u, v))}

```

**restrict** (*dom1*, *dom2=None*)

Restriction to subsets.

INPUT:

- *dom1* – open subset of the domain of *chart1*
- *dom2* – (default: None) open subset of the domain of *chart2*; if None, *dom1* is assumed

OUTPUT:

- the transition map between the charts restricted to the specified subsets

EXAMPLES:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: Y.<u,v> = M.chart()
sage: X_to_Y = X.transition_map(Y, [x+y, x-y])
sage: U = M.open_subset('U', coord_def={X: x>0, Y: u+v>0})
sage: X_to_Y_U = X_to_Y.restrict(U); X_to_Y_U
Change of coordinates from Chart (U, (x, y)) to Chart (U, (u, v))
sage: X_to_Y_U.display()
u = x + y
v = x - y

```

The result is cached:

```

sage: X_to_Y.restrict(U) is X_to_Y_U
True

```

**set\_inverse** (*\*transformations*, *\*\*kws*)

Sets the inverse of the coordinate transformation.

This is useful when the automatic computation via *inverse()* fails.

INPUT:

- *transformations* – the inverse transformations expressed as a list of the expressions of the “old” coordinates in terms of the “new” ones
- *kws* – keyword arguments: only *verbose=True* or *verbose=False* (default) are meaningful; it determines whether the provided transformations are checked to be indeed the inverse coordinate transformations

## EXAMPLES:

From spherical coordinates to Cartesian ones in the plane:

```
sage: M = Manifold(2, 'R^2', structure='topological')
sage: U = M.open_subset('U') # the complement of the half line {y=0, x>= 0}
sage: c_cart.<x,y> = U.chart()
sage: c_spher.<r,ph> = U.chart(r'r:(0,+oo) ph:(0,2*pi):\phi')
sage: spher_to_cart = c_spher.transition_map(c_cart, [r*cos(ph), r*sin(ph)])
sage: spher_to_cart.set_inverse(sqrt(x^2+y^2), atan2(y,x))
sage: spher_to_cart.inverse()
Change of coordinates from Chart (U, (x, y)) to Chart (U, (r, ph))
sage: spher_to_cart.inverse().display()
r = sqrt(x^2 + y^2)
ph = arctan2(y, x)
sage: M.coord_changes() # random (dictionary output)
{(Chart (U, (r, ph)),
  Chart (U, (x, y))): Change of coordinates from Chart (U, (r, ph)) to Chart_
↔(U, (x, y)),
 (Chart (U, (x, y)),
  Chart (U, (r, ph))): Change of coordinates from Chart (U, (x, y)) to Chart_
↔(U, (r, ph))}
```

Introducing a wrong inverse transformation (note the  $x^3$  typo) is revealed by setting `verbose` to `True`:

```
sage: spher_to_cart.set_inverse(sqrt(x^3+y^2), atan2(y,x), verbose=True)
Check of the inverse coordinate transformation:
r == sqrt(r*cos(ph)^3 + sin(ph)^2)*r
ph == arctan2(r*sin(ph), r*cos(ph))
x == sqrt(x^3 + y^2)*x/sqrt(x^2 + y^2)
y == sqrt(x^3 + y^2)*y/sqrt(x^2 + y^2)
```

**class** `sage.manifolds.chart.RealChart` (*domain, coordinates='', names=None*)

Bases: `sage.manifolds.chart.Chart`

Chart on a topological manifold over  $\mathbf{R}$ .

Given a topological manifold  $M$  of dimension  $n$  over  $\mathbf{R}$ , a *chart* on  $M$  is a pair  $(U, \varphi)$ , where  $U$  is an open subset of  $M$  and  $\varphi : U \rightarrow V \subset \mathbf{R}^n$  is a homeomorphism from  $U$  to an open subset  $V$  of  $\mathbf{R}^n$ .

The components  $(x^1, \dots, x^n)$  of  $\varphi$ , defined by  $\varphi(p) = (x^1(p), \dots, x^n(p)) \in \mathbf{R}^n$  for any point  $p \in U$ , are called the *coordinates* of the chart  $(U, \varphi)$ .

INPUT:

- `domain` – open subset  $U$  on which the chart is defined
- `coordinates` – (default: `''` (empty string)) string defining the coordinate symbols and ranges, see below
- `names` – (default: `None`) unused argument, except if `coordinates` is not provided; it must then be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator `<, >` is used)

The string `coordinates` has the space `' '` as a separator and each item has at most three fields, separated by a colon `(:)`:

1. The coordinate symbol (a letter or a few letters).
2. (optional) The interval  $I$  defining the coordinate range: if not provided, the coordinate is assumed to span all  $\mathbf{R}$ ; otherwise  $I$  must be provided in the form  $(a, b)$  (or equivalently  $]a, b[$ ). The bounds  $a$  and  $b$  can be `+/-Infinity`, `Inf`, `infinity`, `inf` or `oo`. For *singular* coordinates, non-open intervals such

as  $[a, b]$  and  $(a, b]$  (or equivalently  $]a, b[$ ) are allowed. Note that the interval declaration must not contain any whitespace.

- 3.(optional) The LaTeX spelling of the coordinate; if not provided the coordinate symbol given in the first field will be used.

The order of the fields 2 and 3 does not matter and each of them can be omitted. If it contains any LaTeX expression, the string `coordinates` must be declared with the prefix ‘r’ (for “raw”) to allow for a proper treatment of LaTeX backslash characters (see examples below). If no interval range and no LaTeX spelling is to be set for any coordinate, the argument `coordinates` can be omitted when the shortcut operator `<, >` is used via Sage parser (see examples below).

#### EXAMPLES:

Cartesian coordinates on  $\mathbf{R}^3$ :

```
sage: M = Manifold(3, 'R^3', r'\RR^3', structure='topological',
....:             start_index=1)
sage: c_cart = M.chart('x y z'); c_cart
Chart (R^3, (x, y, z))
sage: type(c_cart)
<class 'sage.manifolds.chart.RealChart'>
```

To have the coordinates accessible as global variables, one has to set:

```
sage: (x, y, z) = c_cart[:]
```

However, a shortcut is to use the declarator `<x, y, z>` in the left-hand side of the chart declaration (there is then no need to pass the string `'x y z'` to `chart()`):

```
sage: M = Manifold(3, 'R^3', r'\RR^3', structure='topological',
....:             start_index=1)
sage: c_cart.<x,y,z> = M.chart(); c_cart
Chart (R^3, (x, y, z))
```

The coordinates are then immediately accessible:

```
sage: y
y
sage: y is c_cart[2]
True
```

Note that `x`, `y`, `z` declared in `<x,y,z>` are mere Python variable names and do not have to coincide with the coordinate symbols; for instance, one may write:

```
sage: M = Manifold(3, 'R^3', r'\RR^3', structure='topological', start_index=1)
sage: c_cart.<x1,y1,z1> = M.chart('x y z'); c_cart
Chart (R^3, (x, y, z))
```

Then `y` is not known as a global variable and the coordinate `y` is accessible only through the global variable `y1`:

```
sage: y1
y
sage: y1 is c_cart[2]
True
```

However, having the name of the Python variable coincide with the coordinate symbol is quite convenient; so it is recommended to declare:

```

sage: forget() # for doctests only
sage: M = Manifold(3, 'R^3', r'\RR^3', structure='topological', start_index=1)
sage: c_cart.<x,y,z> = M.chart()

```

Spherical coordinates on the subset  $U$  of  $\mathbf{R}^3$  that is the complement of the half-plane  $\{y = 0, x \geq 0\}$ :

```

sage: U = M.open_subset('U')
sage: c_spher.<r,th,ph> = U.chart(r'r:(0,+oo) th:(0,pi):\theta ph:(0,2*pi):\phi')
sage: c_spher
Chart (U, (r, th, ph))

```

Note the prefix 'r' for the string defining the coordinates in the arguments of chart.

Coordinates are Sage symbolic variables (see `sage.symbolic.expression`):

```

sage: type(th)
<type 'sage.symbolic.expression.Expression'>
sage: latex(th)
{\theta}
sage: assumptions(th)
[th is real, th > 0, th < pi]

```

Coordinate are also accessible by their indices:

```

sage: x1 = c_spher[1]; x2 = c_spher[2]; x3 = c_spher[3]
sage: [x1, x2, x3]
[r, th, ph]
sage: (x1, x2, x3) == (r, th, ph)
True

```

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

```

sage: c_cart[: ]
(x, y, z)
sage: c_spher[: ]
(r, th, ph)

```

Let us check that the declared coordinate ranges have been taken into account:

```

sage: c_cart.coord_range()
x: (-oo, +oo); y: (-oo, +oo); z: (-oo, +oo)
sage: c_spher.coord_range()
r: (0, +oo); th: (0, pi); ph: (0, 2*pi)
sage: bool(th>0 and th<pi)
True
sage: assumptions() # list all current symbolic assumptions
[x is real, y is real, z is real, r is real, r > 0, th is real,
 th > 0, th < pi, ph is real, ph > 0, ph < 2*pi]

```

The coordinate ranges are used for simplifications:

```

sage: simplify(abs(r)) # r has been declared to lie in the interval (0,+oo)
r
sage: simplify(abs(x)) # no positive range has been declared for x
abs(x)

```

Each constructed chart is automatically added to the manifold's user atlas:

```
sage: M.atlas()
[Chart (R^3, (x, y, z)), Chart (U, (r, th, ph))]
```

and to the atlas of its domain:

```
sage: U.atlas()
[Chart (U, (r, th, ph))]
```

Manifold subsets have a *default chart*, which, unless changed via the method `set_default_chart()`, is the first defined chart on the subset (or on an open subset of it):

```
sage: M.default_chart()
Chart (R^3, (x, y, z))
sage: U.default_chart()
Chart (U, (r, th, ph))
```

The default charts are not privileged charts on the manifold, but rather charts whose name can be skipped in the argument list of functions having an optional `chart=` argument.

The chart map  $\varphi$  acting on a point is obtained by means of the call operator, i.e. the operator `()`:

```
sage: p = M.point((1,0,-2)); p
Point on the 3-dimensional topological manifold R^3
sage: c_cart(p)
(1, 0, -2)
sage: c_cart(p) == p.coord(c_cart)
True
sage: q = M.point((2,pi/2,pi/3), chart=c_spher) # point defined by its spherical_
↪coordinates
sage: c_spher(q)
(2, 1/2*pi, 1/3*pi)
sage: c_spher(q) == q.coord(c_spher)
True
sage: a = U.point((1,pi/2,pi)) # the default coordinates on U are the spherical_
↪ones
sage: c_spher(a)
(1, 1/2*pi, pi)
sage: c_spher(a) == a.coord(c_spher)
True
```

Cartesian coordinates on  $U$  as an example of chart construction with coordinate restrictions: since  $U$  is the complement of the half-plane  $\{y = 0, x \geq 0\}$ , we must have  $y \neq 0$  or  $x < 0$  on  $U$ . Accordingly, we set:

```
sage: c_cartU.<x,y,z> = U.chart()
sage: c_cartU.add_restrictions((y!=0, x<0))
sage: U.atlas()
[Chart (U, (r, th, ph)), Chart (U, (x, y, z))]
sage: M.atlas()
[Chart (R^3, (x, y, z)), Chart (U, (r, th, ph)), Chart (U, (x, y, z))]
sage: c_cartU.valid_coordinates(-1,0,2)
True
sage: c_cartU.valid_coordinates(1,0,2)
False
sage: c_cart.valid_coordinates(1,0,2)
True
```

Note that, as an example, the following would have meant  $y \neq 0$  and  $x < 0$ :

```
c_cartU.add_restrictions([y!=0, x<0])
```

Chart grids can be drawn in 2D or 3D graphics thanks to the method `plot()`.

#### **add\_restrictions** (*restrictions*)

Add some restrictions on the coordinates.

INPUT:

- *restrictions* – list of restrictions on the coordinates, in addition to the ranges declared by the intervals specified in the chart constructor

A restriction can be any symbolic equality or inequality involving the coordinates, such as  $x > y$  or  $x^2 + y^2 \neq 0$ . The items of the list *restrictions* are combined with the `and` operator; if some restrictions are to be combined with the `or` operator instead, they have to be passed as a tuple in some single item of the list *restrictions*. For example:

```
restrictions = [x > y, (x != 0, y != 0), z^2 < x]
```

means  $(x > y)$  and  $((x \neq 0) \text{ or } (y \neq 0))$  and  $(z^2 < x)$ . If the list *restrictions* contains only one item, this item can be passed as such, i.e. writing  $x > y$  instead of the single element list  $[x > y]$ .

EXAMPLES:

Cartesian coordinates on the open unit disc in  $\mathbf{R}^2$ :

```
sage: M = Manifold(2, 'M', structure='topological') # the open unit disc
sage: X.<x,y> = M.chart()
sage: X.add_restrictions(x^2+y^2<1)
sage: X.valid_coordinates(0,2)
False
sage: X.valid_coordinates(0,1/3)
True
```

The restrictions are transmitted to subcharts:

```
sage: A = M.open_subset('A') # annulus 1/2 < r < 1
sage: X_A = X.restrict(A, x^2+y^2 > 1/4)
sage: X_A._restrictions
[x^2 + y^2 < 1, x^2 + y^2 > (1/4)]
sage: X_A.valid_coordinates(0,1/3)
False
sage: X_A.valid_coordinates(2/3,1/3)
True
```

If appropriate, the restrictions are transformed into bounds on the coordinate ranges:

```
sage: U = M.open_subset('U')
sage: X_U = X.restrict(U)
sage: X_U.coord_range()
x: (-oo, +oo); y: (-oo, +oo)
sage: X_U.add_restrictions([x<0, y>1/2])
sage: X_U.coord_range()
x: (-oo, 0); y: (1/2, +oo)
```

#### **coord\_bounds** (*i=None*)

Return the lower and upper bounds of the range of a coordinate.

For a nicely formatted output, use `coord_range()` instead.

INPUT:

- `i` – (default: `None`) index of the coordinate; if `None`, the bounds of all the coordinates are returned

OUTPUT:

- the coordinate bounds as the tuple `((xmin, min_included), (xmax, max_included))` where

– `xmin` is the coordinate lower bound

– `min_included` is a boolean, indicating whether the coordinate can take the value `xmin`, i.e. `xmin` is a strict lower bound iff `min_included` is `False`

– `xmax` is the coordinate upper bound

– `max_included` is a boolean, indicating whether the coordinate can take the value `xmax`, i.e. `xmax` is a strict upper bound iff `max_included` is `False`

EXAMPLES:

Some coordinate bounds on a 2-dimensional manifold:

```
sage: forget() # for doctests only
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart('x y:[0,1]')
sage: c_xy.coord_bounds(0) # x in (-oo,+oo) (the default)
((-Infinity, False), (+Infinity, False))
sage: c_xy.coord_bounds(1) # y in [0,1]
((0, True), (1, False))
sage: c_xy.coord_bounds()
((-Infinity, False), (+Infinity, False)), ((0, True), (1, False))
sage: c_xy.coord_bounds() == (c_xy.coord_bounds(0), c_xy.coord_bounds(1))
True
```

The coordinate bounds can also be recovered via the method `coord_range()`:

```
sage: c_xy.coord_range()
x: (-oo, +oo); y: [0, 1]
sage: c_xy.coord_range(y)
y: [0, 1]
```

or via Sage's function `sage.symbolic.assumptions.assumptions()`:

```
sage: assumptions(x)
[x is real]
sage: assumptions(y)
[y is real, y >= 0, y < 1]
```

**coord\_range** (*xx=None*)

Display the range of a coordinate (or all coordinates), as an interval.

INPUT:

- `xx` – (default: `None`) symbolic expression corresponding to a coordinate of the current chart; if `None`, the ranges of all coordinates are displayed

EXAMPLES:

Ranges of coordinates on a 2-dimensional manifold:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: X.coord_range()
x: (-oo, +oo); y: (-oo, +oo)
sage: X.coord_range(x)
x: (-oo, +oo)
sage: U = M.open_subset('U', coord_def={X: [x>1, y<pi]})
sage: XU = X.restrict(U) # restriction of chart X to U
sage: XU.coord_range()
x: (1, +oo); y: (-oo, pi)
sage: XU.coord_range(x)
x: (1, +oo)
sage: XU.coord_range(y)
y: (-oo, pi)

```

The output is LaTeX-formatted for the notebook:

```

sage: latex(XU.coord_range(y))
y : \ \left( -\infty, \pi \right)

```

**plot** (*chart=None, ambient\_coords=None, mapping=None, fixed\_coords=None, ranges=None, number\_values=None, steps=None, parameters=None, max\_range=8, style='-', label\_axes=True, color='red', plot\_points=75, thickness=1, \*\*kws*)

Plot self as a grid in a Cartesian graph based on the coordinates of some ambient chart.

The grid is formed by curves along which a chart coordinate varies, the other coordinates being kept fixed. It is drawn in terms of two (2D graphics) or three (3D graphics) coordinates of another chart, called hereafter the *ambient chart*.

The ambient chart is related to the current chart either by a transition map if both charts are defined on the same manifold, or by the coordinate expression of some continuous map (typically an immersion). In the latter case, the two charts may be defined on two different manifolds.

INPUT:

- *chart* – (default: None) the ambient chart (see above); if None, the ambient chart is set to the current chart
- *ambient\_coords* – (default: None) tuple containing the 2 or 3 coordinates of the ambient chart in terms of which the plot is performed; if None, all the coordinates of the ambient chart are considered
- *mapping* – (default: None) *ContinuousMap*; continuous manifold map providing the link between the current chart and the ambient chart (cf. above); if None, both charts are supposed to be defined on the same manifold and related by some transition map (see *transition\_map()*)
- *fixed\_coords* – (default: None) dictionary with keys the chart coordinates that are not drawn and with values the fixed value of these coordinates; if None, all the coordinates of the current chart are drawn
- *ranges* – (default: None) dictionary with keys the coordinates to be drawn and values tuples (*x\_min*, *x\_max*) specifying the coordinate range for the plot; if None, the entire coordinate range declared during the chart construction is considered (with *-Infinity* replaced by *-max\_range* and *+Infinity* by *max\_range*)
- *number\_values* – (default: None) either an integer or a dictionary with keys the coordinates to be drawn and values the number of constant values of the coordinate to be considered; if *number\_values* is a single integer, it represents the number of constant values for all coordinates; if *number\_values* is None, it is set to 9 for a 2D plot and to 5 for a 3D plot

- `steps` – (default: `None`) dictionary with keys the coordinates to be drawn and values the step between each constant value of the coordinate; if `None`, the step is computed from the coordinate range (specified in `ranges`) and `number_values`. On the contrary if the step is provided for some coordinate, the corresponding number of constant values is deduced from it and the coordinate range.
- `parameters` – (default: `None`) dictionary giving the numerical values of the parameters that may appear in the relation between the two coordinate systems
- `max_range` – (default: `8`) numerical value substituted to `+Infinity` if the latter is the upper bound of the range of a coordinate for which the plot is performed over the entire coordinate range (i.e. for which no specific plot range has been set in `ranges`); similarly `-max_range` is the numerical valued substituted for `-Infinity`
- `color` – (default: `'red'`) either a single color or a dictionary of colors, with keys the coordinates to be drawn, representing the colors of the lines along which the coordinate varies, the other being kept constant; if `color` is a single color, it is used for all coordinate lines
- `style` – (default: `'-'`) either a single line style or a dictionary of line styles, with keys the coordinates to be drawn, representing the style of the lines along which the coordinate varies, the other being kept constant; if `style` is a single style, it is used for all coordinate lines; NB: `style` is effective only for 2D plots
- `thickness` – (default: `1`) either a single line thickness or a dictionary of line thicknesses, with keys the coordinates to be drawn, representing the thickness of the lines along which the coordinate varies, the other being kept constant; if `thickness` is a single value, it is used for all coordinate lines
- `plot_points` – (default: `75`) either a single number of points or a dictionary of integers, with keys the coordinates to be drawn, representing the number of points to plot the lines along which the coordinate varies, the other being kept constant; if `plot_points` is a single integer, it is used for all coordinate lines
- `label_axes` – (default: `True`) boolean determining whether the labels of the ambient coordinate axes shall be added to the graph; can be set to `False` if the graph is 3D and must be superposed with another graph

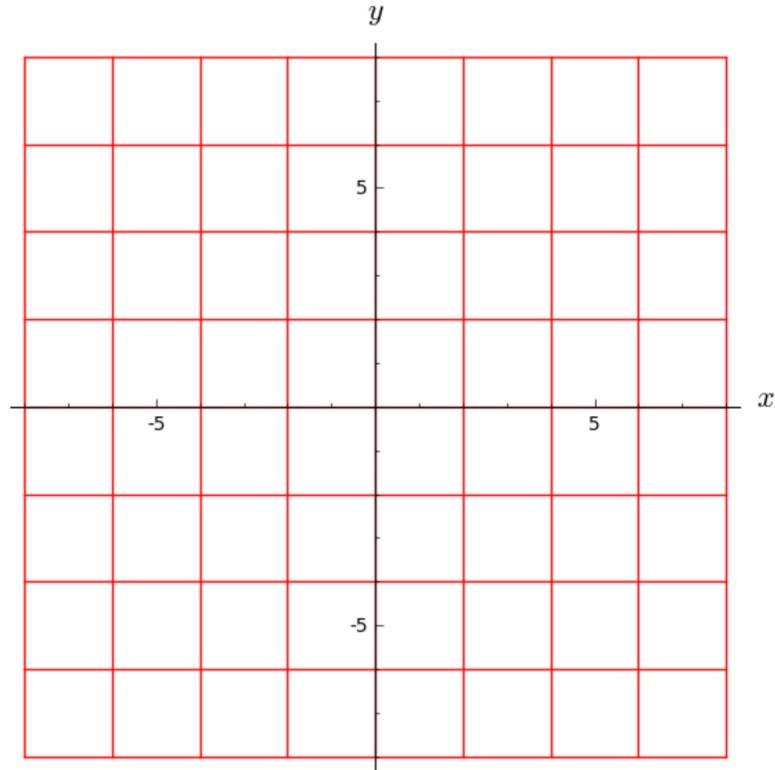
OUTPUT:

- a graphic object, either a `Graphics` for a 2D plot (i.e. based on 2 coordinates of the ambient chart) or a `Graphics3d` for a 3D plot (i.e. based on 3 coordinates of the ambient chart)

EXAMPLES:

A 2-dimensional chart plotted in terms of itself results in a rectangular grid:

```
sage: R2 = Manifold(2, 'R^2', structure='topological') # the Euclidean plane
sage: c_cart.<x,y> = R2.chart() # Cartesian coordinates
sage: g = c_cart.plot() # equivalent to c_cart.plot(c_cart)
sage: g
Graphics object consisting of 18 graphics primitives
```

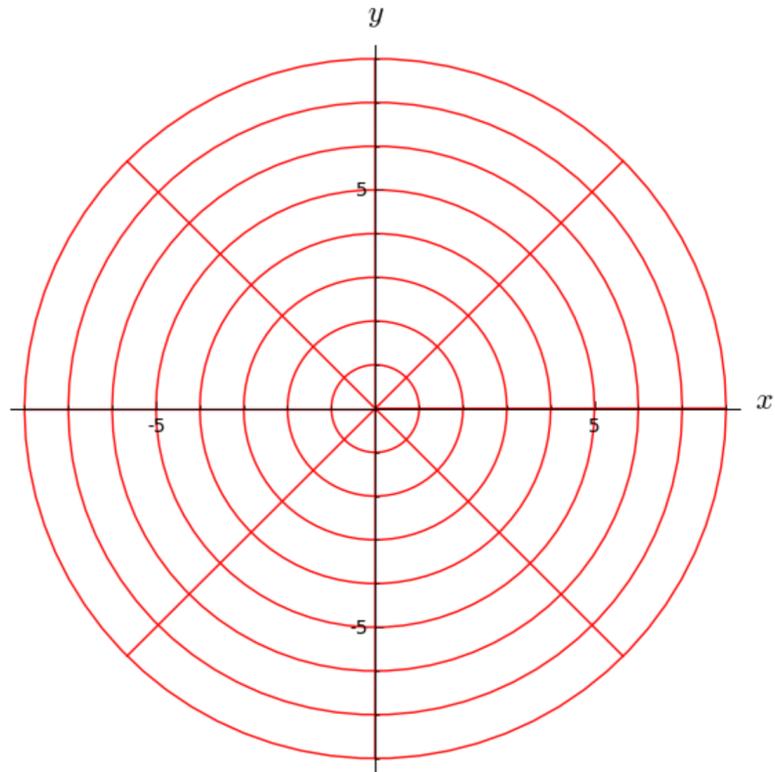


Grid of polar coordinates in terms of Cartesian coordinates in the Euclidean plane:

```

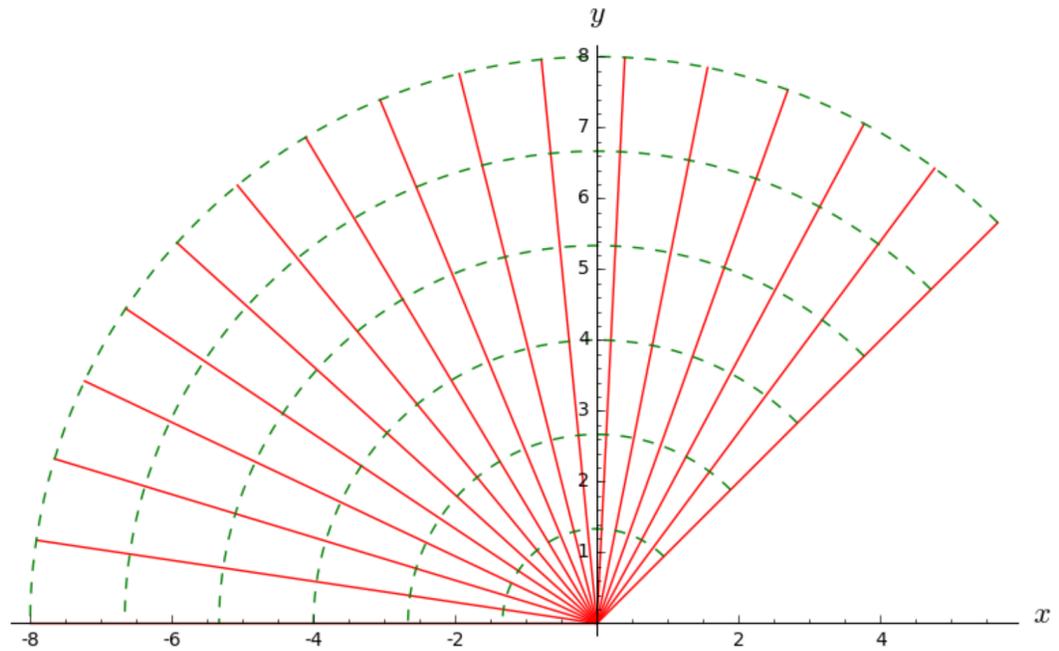
sage: U = R2.open_subset('U', coord_def={c_cart: (y!=0, x<0)}) # the_
↳complement of the segment y=0 and x>0
sage: c_pol.<r,ph> = U.chart(r'r:(0,+oo) ph:(0,2*pi):\phi') # polar_
↳coordinates on U
sage: pol_to_cart = c_pol.transition_map(c_cart, [r*cos(ph), r*sin(ph)])
sage: g = c_pol.plot(c_cart)
sage: g
Graphics object consisting of 18 graphics primitives

```



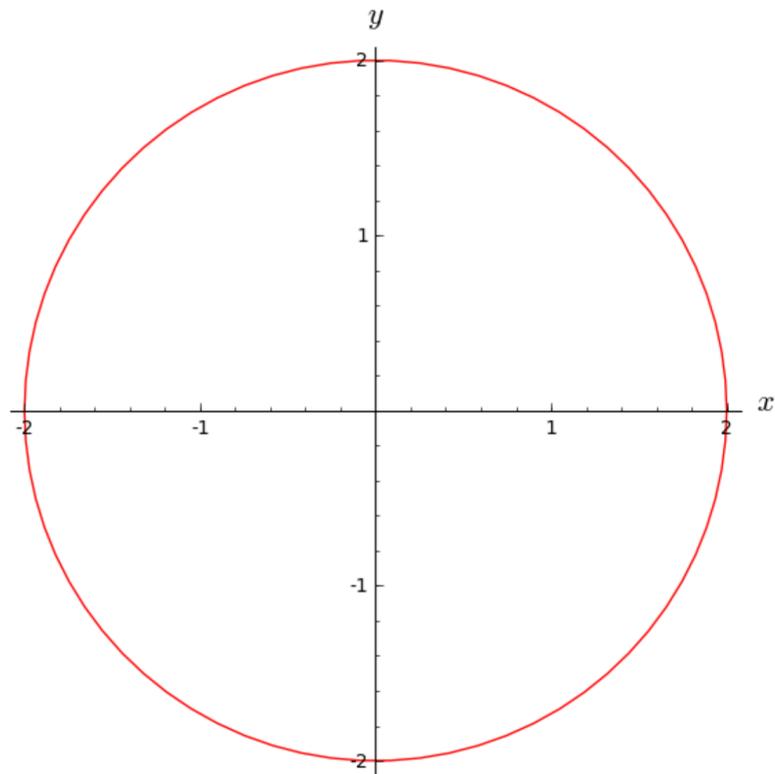
Call with non-default values:

```
sage: g = c_pol.plot(c_cart, ranges={ph:(pi/4,pi)},
....:               number_values={r:7, ph:17},
....:               color={r:'red', ph:'green'},
....:               style={r:'-', ph:'--'})
```

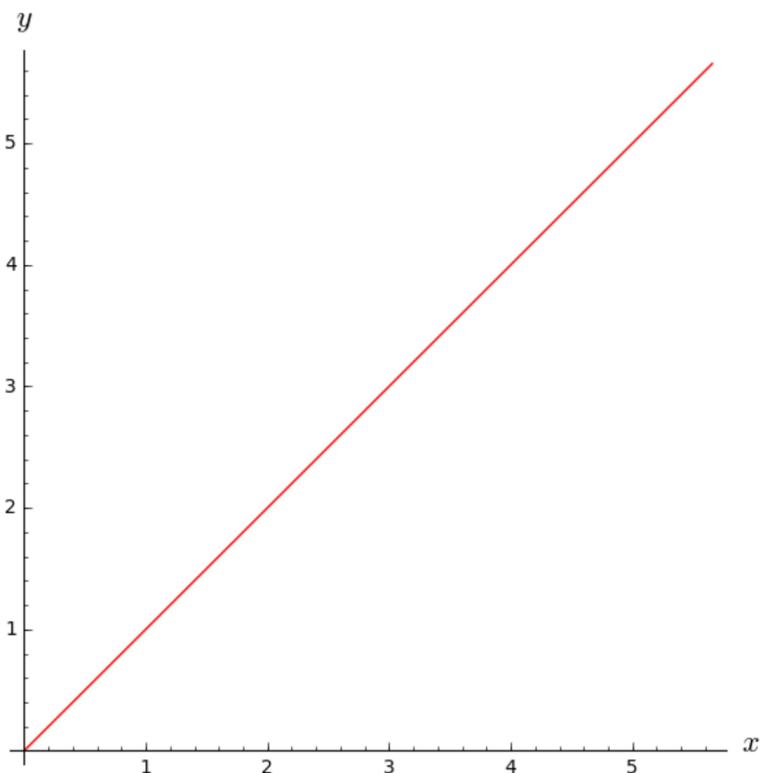


A single coordinate line can be drawn:

```
sage: g = c_pol.plot(c_cart, fixed_coords={r: 2}) # draw a circle of radius  $r=2$ 
```



```
sage: g = c_pol.plot(c_cart, fixed_coords={ph: pi/4}) # draw a segment at  $\varphi$   
↪ phi=pi/4
```

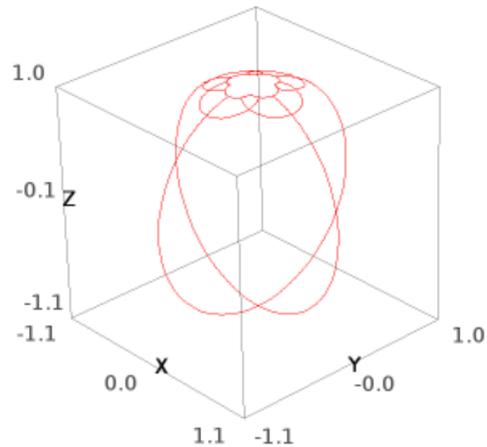


An example with the ambient chart lying in another manifold (the plot is then performed via some manifold map passed as the argument mapping): 3D plot of the stereographic charts on the 2-sphere:

```

sage: S2 = Manifold(2, 'S^2', structure='topological') # the 2-sphere
sage: U = S2.open_subset('U') ; V = S2.open_subset('V') # complement of the
↳ North and South pole, respectively
sage: S2.declare_union(U,V)
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                               intersection_name='W', restrictions1= x^2+y^2!=0,
....:                               restrictions2= u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: R3 = Manifold(3, 'R^3', structure='topological') # the Euclidean space
↳ R^3
sage: c_cart.<X,Y,Z> = R3.chart() # Cartesian coordinates on R^3
sage: Phi = S2.continuous_map(R3, {(c_xy, c_cart): [2*x/(1+x^2+y^2),
....:                               2*y/(1+x^2+y^2), (x^2+y^2-1)/(1+x^2+y^2)],
....:                               (c_uv, c_cart): [2*u/(1+u^2+v^2),
....:                               2*v/(1+u^2+v^2), (1-u^2-v^2)/(1+u^2+v^2)]},
....:                               name='Phi', latex_name=r'\Phi') # Embedding of
↳ S^2 in R^3
sage: g = c_xy.plot(c_cart, mapping=Phi)
sage: g
Graphics3d Object

```



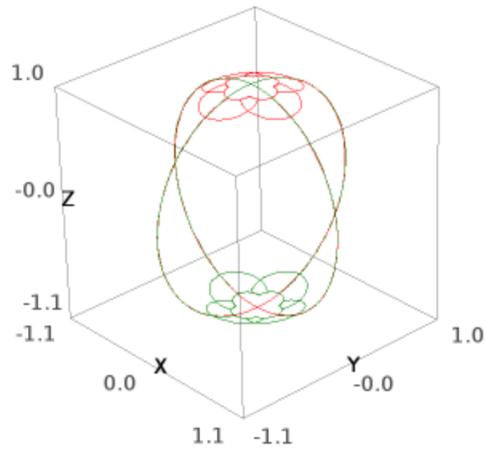
NB: to get a better coverage of the whole sphere, one should increase the coordinate sampling via the argument `number_values` or the argument `steps` (only the default value, `number_values = 5`, is used here, which is pretty low).

The same plot without the  $(X, Y, Z)$  axes labels:

```
sage: g = c_xy.plot(c_cart, mapping=Phi, label_axes=False)
```

The North and South stereographic charts on the same plot:

```
sage: g2 = c_uv.plot(c_cart, mapping=Phi, color='green')
sage: g + g2
Graphics3d Object
```

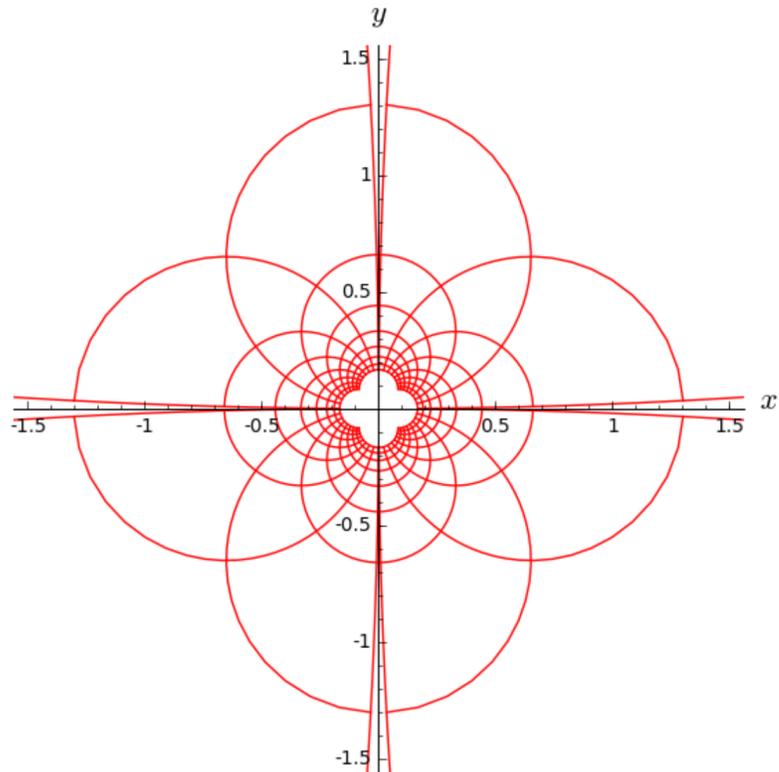


South stereographic chart drawn in terms of the North one (we split the plot in four parts to avoid the singularity at  $(u, v) = (0, 0)$ ):

```

sage: W = U.intersection(V) # the subset common to both charts
sage: c_uvW = c_uv.restrict(W) # chart (W, (u,v))
sage: gSN1 = c_uvW.plot(c_xy, ranges={u: [-6., -0.02], v: [-6., -0.02]}) # long_
↪time
sage: gSN2 = c_uvW.plot(c_xy, ranges={u: [-6., -0.02], v: [0.02, 6.]}) # long_
↪time
sage: gSN3 = c_uvW.plot(c_xy, ranges={u: [0.02, 6.], v: [-6., -0.02]}) # long_
↪time
sage: gSN4 = c_uvW.plot(c_xy, ranges={u: [0.02, 6.], v: [0.02, 6.]}) # long time
sage: show(gSN1+gSN2+gSN3+gSN4, xmin=-1.5, xmax=1.5, ymin=-1.5, ymax=1.5) #_
↪long time

```

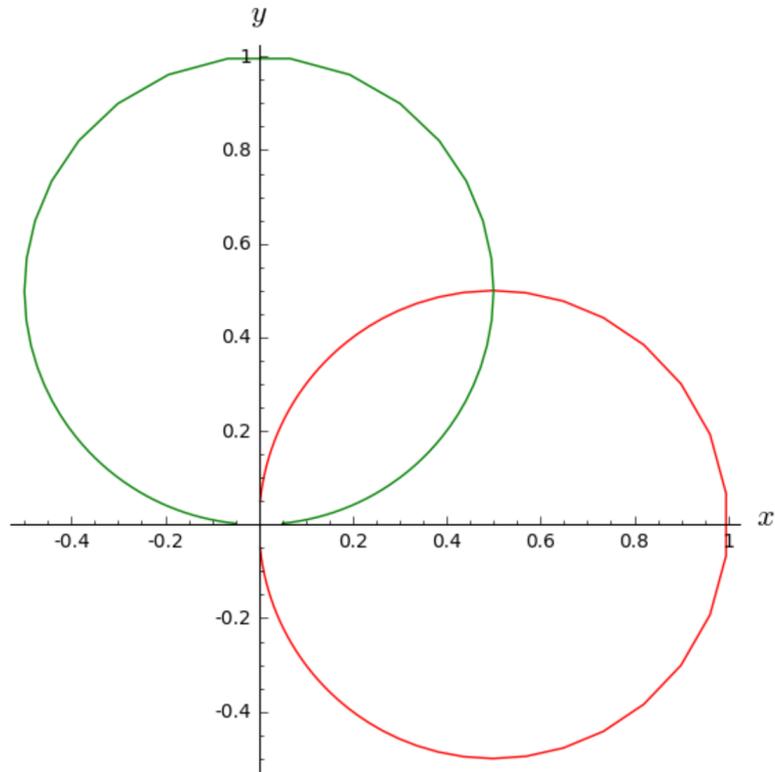


The coordinate line  $u = 1$  (red) and the coordinate line  $v = 1$  (green) on the same plot:

```

sage: gul = c_uvW.plot(c_xy, fixed_coords={u: 1}, max_range=20, plot_
↳points=300) # long time
sage: gv1 = c_uvW.plot(c_xy, fixed_coords={v: 1}, max_range=20, plot_
↳points=300, color='green') # long time
sage: gul + gv1 # long time
Graphics object consisting of 2 graphics primitives

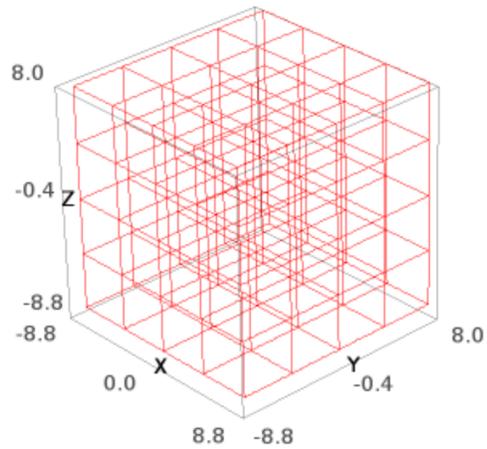
```



Note that we have set `max_range=20` to have a wider range for the coordinates  $u$  and  $v$ , i.e. to have  $[-20, 20]$  instead of the default  $[-8, 8]$ .

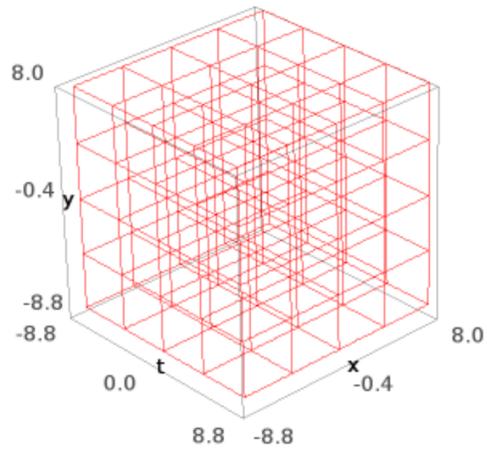
A 3-dimensional chart plotted in terms of itself results in a 3D rectangular grid:

```
sage: g = c_cart.plot() # equivalent to c_cart.plot(c_cart) # long time
sage: g # long time
Graphics3d Object
```

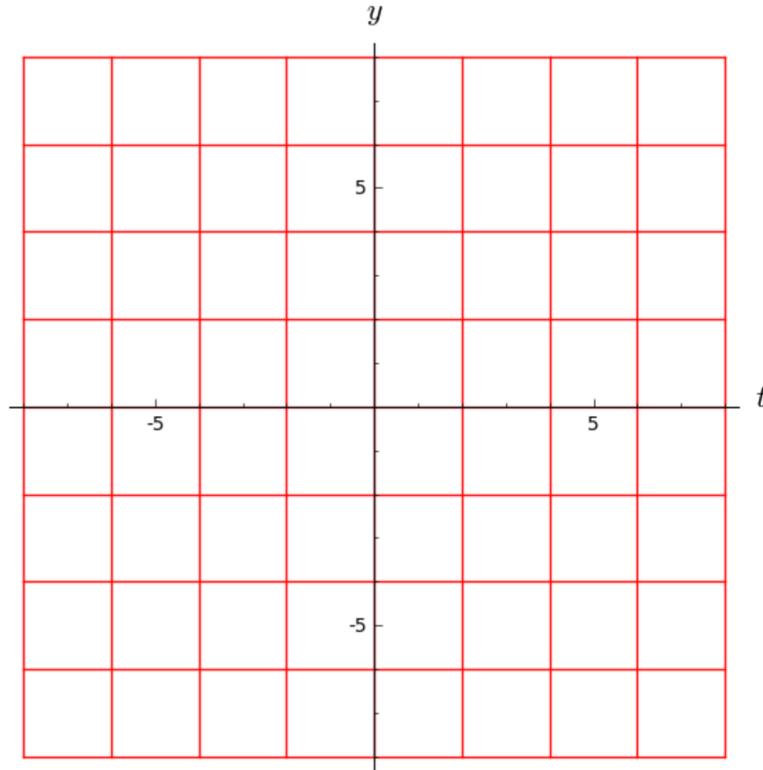


A 4-dimensional chart plotted in terms of itself (the plot is performed for at most 3 coordinates, which must be specified via the argument `ambient_coords`):

```
sage: M = Manifold(4, 'M', structure='topological')
sage: X.<t,x,y,z> = M.chart()
sage: g = X.plot(ambient_coords=(t,x,y)) # the coordinate z is not depicted
↔ # long time
sage: g # long time
Graphics3d Object
```



```
sage: g = X.plot(ambient_coords=(t,y)) # the coordinates x and z are not_
↳depicted
sage: g
Graphics object consisting of 18 graphics primitives
```



Note that the default values of some arguments of the method `plot` are stored in the dictionary `plot.options`:

```
sage: X.plot.options # random (dictionary output)
{'color': 'red', 'label_axes': True, 'max_range': 8,
 'plot_points': 75, 'style': '-', 'thickness': 1}
```

so that they can be adjusted by the user:

```
sage: X.plot.options['color'] = 'blue'
```

From now on, all chart plots will use blue as the default color. To restore the original default options, it suffices to type:

```
sage: X.plot.reset()
```

**restrict** (*subset*, *restrictions=None*)

Return the restriction of the chart to some open subset of its domain.

If the current chart is  $(U, \varphi)$ , a *restriction* (or *subchart*) is a chart  $(V, \psi)$  such that  $V \subset U$  and  $\psi = \varphi|_V$ .

If such subchart has not been defined yet, it is constructed here.

The coordinates of the subchart bare the same names as the coordinates of the current chart.

INPUT:

- *subset* – open subset  $V$  of the chart domain  $U$  (must be an instance of *TopologicalManifold*)
- *restrictions* – (default: `None`) list of coordinate restrictions defining the subset  $V$

A restriction can be any symbolic equality or inequality involving the coordinates, such as  $x > y$  or  $x^2 + y^2 \neq 0$ . The items of the list `restrictions` are combined with the `and` operator; if some restrictions are to be combined with the `or` operator instead, they have to be passed as a tuple in some single item of the list `restrictions`. For example:

```
restrictions = [x > y, (x != 0, y != 0), z^2 < x]
```

means  $(x > y)$  and  $((x \neq 0) \text{ or } (y \neq 0))$  and  $(z^2 < x)$ . If the list `restrictions` contains only one item, this item can be passed as such, i.e. writing  $x > y$  instead of the single element list  $[x > y]$ .

OUTPUT:

- the chart  $(V, \psi)$  as a *RealChart*

EXAMPLES:

Cartesian coordinates on the unit open disc in  $\mathbf{R}^2$  as a subchart of the global Cartesian coordinates:

```
sage: M = Manifold(2, 'R^2', structure='topological')
sage: c_cart.<x,y> = M.chart() # Cartesian coordinates on R^2
sage: D = M.open_subset('D') # the unit open disc
sage: c_cart_D = c_cart.restrict(D, x^2+y^2<1)
sage: p = M.point((1/2, 0))
sage: p in D
True
sage: q = M.point((1, 2))
sage: q in D
False
```

Cartesian coordinates on the annulus  $1 < \sqrt{x^2 + y^2} < 2$ :

```
sage: A = M.open_subset('A')
sage: c_cart_A = c_cart.restrict(A, [x^2+y^2>1, x^2+y^2<4])
sage: p in A, q in A
(False, False)
sage: a = M.point((3/2, 0))
sage: a in A
True
```

**valid\_coordinates** (\*coordinates, \*\*kwds)

Check whether a tuple of coordinates can be the coordinates of a point in the chart domain.

INPUT:

- \*coordinates – coordinate values
- \*\*kwds – options:
  - tolerance=0, to set the absolute tolerance in the test of coordinate ranges
  - parameters=None, to set some numerical values to parameters

OUTPUT:

- True if the coordinate values are admissible in the chart range and False otherwise

EXAMPLES:

Cartesian coordinates on a square interior:

```

sage: forget() # for doctest only
sage: M = Manifold(2, 'M', structure='topological') # the square interior
sage: X.<x,y> = M.chart('x: (-2,2) y: (-2,2)')
sage: X.valid_coordinates(0,1)
True
sage: X.valid_coordinates(-3/2,5/4)
True
sage: X.valid_coordinates(0,3)
False

```

The unit open disk inside the square:

```

sage: D = M.open_subset('D', coord_def={X: x^2+y^2<1})
sage: XD = X.restrict(D)
sage: XD.valid_coordinates(0,1)
False
sage: XD.valid_coordinates(-3/2,5/4)
False
sage: XD.valid_coordinates(-1/2,1/2)
True
sage: XD.valid_coordinates(0,0)
True

```

Another open subset of the square, defined by  $x^2 + y^2 < 1$  or ( $x > 0$  and  $|y| < 1$ ):

```

sage: B = M.open_subset('B',
.....:                    coord_def={X: (x^2+y^2<1,
.....:                                   [x>0, abs(y)<1])})
sage: XB = X.restrict(B)
sage: XB.valid_coordinates(-1/2, 0)
True
sage: XB.valid_coordinates(-1/2, 3/2)
False
sage: XB.valid_coordinates(3/2, 1/2)
True

```

## 1.5.2 Coordinate Functions

In the context of a topological manifold  $M$  over a topological field  $K$ , a *coordinate function* is a function from a chart codomain to  $K$ . In other words, a coordinate function is a  $K$ -valued function of the coordinates associated to some chart.

More precisely, let  $(U, \varphi)$  be a chart on  $M$ , i.e.  $U$  is an open subset of  $M$  and  $\varphi : U \rightarrow V \subset K^n$  is a homeomorphism from  $U$  to an open subset  $V$  of  $K^n$ . A *coordinate function associated to the chart*  $(U, \varphi)$  is a function

$$f : \begin{array}{ccc} V \subset K^n & \longrightarrow & K \\ (x^1, \dots, x^n) & \longmapsto & f(x^1, \dots, x^n) \end{array}$$

Coordinate functions are implemented by derived classes of the abstract base class `CoordFunction`.

The class `MultiCoordFunction` implements  $K^m$ -valued functions of the coordinates of a chart, with  $m$  a positive integer.

AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2013-2015) : initial version
- Travis Scrimshaw (2016) : make `CoordFunction` inheritate from `AlgebraElement`

**class** `sage.manifolds.coord_func.CoordFunction` (*parent*)

Bases: `sage.structure.element.AlgebraElement`

Abstract base class for coordinate functions.

If  $(U, \varphi)$  is a chart on a topological manifold  $M$  of dimension  $n$  over a topological field  $K$ , a *coordinate function* associated to  $(U, \varphi)$  is a map  $f : V \subset K^n \rightarrow K$ , where  $V$  is the codomain of  $\varphi$ . In other words,  $f$  is a  $K$ -valued function of the coordinates associated to the chart  $(U, \varphi)$ .

The class `CoordFunction` is an abstract one. Specific coordinate functions must be implemented by derived classes, like `CoordFunctionSymb` for symbolic coordinate functions.

INPUT:

- `parent` – the algebra of coordinate functions on a given chart

**arccos** ()

Arc cosine of `self`.

OUTPUT:

- coordinate function `arccos(f)`, where  $f$  is the current coordinate function

**arccosh** ()

Inverse hyperbolic cosine of `self`.

OUTPUT:

- coordinate function `arccosh(f)`, where  $f$  is the current coordinate function

**arcsin** ()

Arc sine of `self`.

OUTPUT:

- coordinate function `arcsin(f)`, where  $f$  is the current coordinate function

**arcsinh** ()

Inverse hyperbolic sine of `self`.

OUTPUT:

- coordinate function `arcsinh(f)`, where  $f$  is the current coordinate function

**arctan** ()

Arc tangent of `self`.

OUTPUT:

- coordinate function `arctan(f)`, where  $f$  is the current coordinate function

**arctanh** ()

Inverse hyperbolic tangent of `self`.

OUTPUT:

- coordinate function `arctanh(f)`, where  $f$  is the current coordinate function

**chart** ()

Return the chart with respect to which `self` is defined.

OUTPUT:

- a `Chart`

EXAMPLES:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(1+x+y^2)
sage: f.chart()
Chart (M, (x, y))
sage: f.chart() is X
True

```

**copy()**

Return an exact copy of the object.

OUTPUT:

- an instance of *CoordFunction*

**cos()**

Cosine of *self*.

OUTPUT:

- coordinate function  $\cos(f)$ , where  $f$  is the current coordinate function

**cosh()**

Hyperbolic cosine of *self*.

OUTPUT:

- coordinate function  $\cosh(f)$ , where  $f$  is the current coordinate function

**diff(coord)**

Return the partial derivative of *self* with respect to a coordinate.

INPUT:

- coord* – either the coordinate  $x^i$  with respect to which the derivative of the coordinate function  $f$  is to be taken, or the index  $i$  labelling this coordinate (with the index convention defined on the chart domain via the parameter *start\_index*)

OUTPUT:

- instance of *CoordFunction* representing the partial derivative  $\frac{\partial f}{\partial x^i}$

**disp()**

Display the function in arrow notation.

**display()**

Display the function in arrow notation.

**exp()**

Exponential of *self*.

OUTPUT:

- coordinate function  $\exp(f)$ , where  $f$  is the current coordinate function.

**expr()**

Return some data that, along with the chart, is sufficient to reconstruct the object.

For a symbolic coordinate function, this returns the symbol expression representing the function (see *sage.manifolds.coord\_func\_symb.CoordFunctionSymb.expr()*)

**log(base=None)**

Logarithm of *self*.

INPUT:

- `base` – (default: `None`) base of the logarithm; if `None`, the natural logarithm (i.e. logarithm to base `e`) is returned

OUTPUT:

- coordinate function  $\log_a(f)$ , where  $f$  is the current coordinate function and  $a$  is the base

**scalar\_field** (`name=None`, `latex_name=None`)

Construct the scalar field that has `self` as coordinate expression.

The domain of the scalar field is the open subset covered by the chart on which `self` is defined.

INPUT:

- `name` – (default: `None`) name given to the scalar field
- `latex_name` – (default: `None`) LaTeX symbol to denote the scalar field; if `None`, the LaTeX symbol is set to `name`

OUTPUT:

- a *ScalarField*

EXAMPLES:

Construction of a scalar field on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: fc = c_xy.function(x+2*y^3)
sage: f = fc.scalar_field() ; f
Scalar field on the 2-dimensional topological manifold M
sage: f.display()
M --> R
(x, y) |--> 2*y^3 + x
sage: f.coord_function(c_xy) is fc
True
```

**sin** ()

Sine of `self`.

OUTPUT:

- coordinate function  $\sin(f)$ , where  $f$  is the current coordinate function

**sinh** ()

Hyperbolic sine of `self`.

OUTPUT:

- coordinate function  $\sinh(f)$ , where  $f$  is the current coordinate function

**sqrt** ()

Square root of `self`.

OUTPUT:

- coordinate function  $\sqrt{f}$ , where  $f$  is the current coordinate function

**tan** ()

Tangent of `self`.

OUTPUT:

- coordinate function  $\tan(f)$ , where  $f$  is the current coordinate function

**tanh()**

Hyperbolic tangent of self.

OUTPUT:

- coordinate function  $\tanh(f)$ , where  $f$  is the current coordinate function

**class** `sage.manifolds.coord_func.MultiCoordFunction` (*chart, expressions*)

Bases: `sage.structure.sage_object.SageObject`

Coordinate function to some Cartesian power of the base field.

If  $n$  and  $m$  are two positive integers and  $(U, \varphi)$  is a chart on a topological manifold  $M$  of dimension  $n$  over a topological field  $K$ , a *multi-coordinate function* associated to  $(U, \varphi)$  is a map

$$\begin{aligned} f: V \subset K^n &\longrightarrow K^m \\ (x^1, \dots, x^n) &\longmapsto (f_1(x^1, \dots, x^n), \dots, f_m(x^1, \dots, x^n)), \end{aligned}$$

where  $V$  is the codomain of  $\varphi$ . In other words,  $f$  is a  $K^m$ -valued function of the coordinates associated to the chart  $(U, \varphi)$ . Each component  $f_i$  ( $1 \leq i \leq m$ ) is a coordinate function and is therefore stored as a `CoordFunction`.

INPUT:

- chart* – the chart  $(U, \varphi)$
- expressions* – list (or tuple) of length  $m$  of elements to construct the coordinate functions  $f_i$  ( $1 \leq i \leq m$ ); for symbolic coordinate functions, this must be symbolic expressions involving the chart coordinates, while for numerical coordinate functions, this must be data file names

EXAMPLES:

A function  $f: V \subset \mathbf{R}^2 \rightarrow \mathbf{R}^3$ :

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.multifunction(x-y, x*y, cos(x)*exp(y)); f
Coordinate functions (x - y, x*y, cos(x)*e^y) on the Chart (M, (x, y))
sage: type(f)
<class 'sage.manifolds.coord_func.MultiCoordFunction'>
sage: f(x,y)
(x - y, x*y, cos(x)*e^y)
sage: latex(f)
\left(x - y, x y, \cos\left(x\right) e^{\left(y\right)}\right)
```

Each real-valued function  $f_i$  ( $1 \leq i \leq m$ ) composing  $f$  can be accessed via the square-bracket operator, by providing  $i - 1$  as an argument:

```
sage: f[0]
x - y
sage: f[1]
x*y
sage: f[2]
cos(x)*e^y
```

We can give a more verbose explanation of each function:

```
sage: f[0].display()
(x, y) |--> x - y
```

Each `f[i-1]` is an instance of `CoordFunction`:

```
sage: isinstance(f[0], sage.manifolds.coord_func.CoordFunction)
True
```

In the present case,  $f[i-1]$  is an instance of the subclass *CoordFunctionSymb*:

```
sage: type(f[0])
<class 'sage.manifolds.coord_func_symb.CoordFunctionSymbRing_with_category.
↳element_class'>
```

A class *MultiCoordFunction* can represent a real-valued function (case  $m = 1$ ), although one should rather employ the class *CoordFunction* for this purpose:

```
sage: g = X.multifunction(x*y^2)
sage: g(x,y)
(x*y^2,)
```

Evaluating the functions at specified coordinates:

```
sage: f(1,2)
(-1, 2, cos(1)*e^2)
sage: var('a b')
(a, b)
sage: f(a,b)
(a - b, a*b, cos(a)*e^b)
sage: g(1,2)
(4,)
```

**chart()**

Return the chart with respect to which `self` is defined.

OUTPUT:

•a *Chart*

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.multifunction(x-y, x*y, cos(x)*exp(y))
sage: f.chart()
Chart (M, (x, y))
sage: f.chart() is X
True
```

**expr()**

Return a tuple of data, the item no. 'i' begin sufficient to reconstruct the coordinate function no.  $i$ .

In other words, if  $f$  is a multi-coordinate function, then  $f.chart().multifunction(*(f.expr()))$  results in a multi-coordinate function identical to  $f$ .

For a symbolic multi-coordinate function, *expr()* returns the tuple of the symbolic expressions of the coordinate functions composing the object.

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.multifunction(x-y, x*y, cos(x)*exp(y))
sage: f.expr()
```

```
(x - y, x*y, cos(x)*e^y)
sage: type(f.expr()[0])
<type 'sage.symbolic.expression.Expression'>
```

One shall always have:

```
sage: f.chart().multifunction(*(f.expr())) == f
True
```

### `jacobian()`

Return the Jacobian matrix of the system of coordinate functions.

`jacobian()` is a 2-dimensional array of size  $m \times n$ , where  $m$  is the number of functions and  $n$  the number of coordinates, the generic element being  $J_{ij} = \frac{\partial f_i}{\partial x^j}$  with  $1 \leq i \leq m$  (row index) and  $1 \leq j \leq n$  (column index).

OUTPUT:

- Jacobian matrix as a 2-dimensional array  $J$  of coordinate functions with  $J[i-1][j-1]$  being  $J_{ij} = \frac{\partial f_i}{\partial x^j}$  for  $1 \leq i \leq m$  and  $1 \leq j \leq n$

EXAMPLES:

Jacobian of a set of 3 functions of 2 coordinates:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.multifunction(x-y, x*y, y^3*cos(x))
sage: f.jacobian()
[      1      -1]
[      y      x]
[-y^3*sin(x) 3*y^2*cos(x)]
```

Each element of the result is a *coordinate function*:

```
sage: type(f.jacobian()[2,0])
<class 'sage.manifolds.coord_func_symb.CoordFunctionSymbRing_with_category.
  ↳element_class'>
sage: f.jacobian()[2,0].display()
(x, y) |--> -y^3*sin(x)
```

Test of the computation:

```
sage: [[f.jacobian()[i,j] == f[i].diff(j) for j in range(2)] for i in_
  ↳range(3)]
[[True, True], [True, True], [True, True]]
```

Test with `start_index = 1`:

```
sage: M = Manifold(2, 'M', structure='topological', start_index=1)
sage: X.<x,y> = M.chart()
sage: f = X.multifunction(x-y, x*y, y^3*cos(x))
sage: f.jacobian()
[      1      -1]
[      y      x]
[-y^3*sin(x) 3*y^2*cos(x)]
sage: [[f.jacobian()[i,j] == f[i].diff(j+1) for j in range(2)] # note the j+1
  .....:          for i in range(3)]
[[True, True], [True, True], [True, True]]
```

**jacobian\_det()**

Return the Jacobian determinant of the system of functions.

The number  $m$  of coordinate functions must equal the number  $n$  of coordinates.

OUTPUT:

- a *CoordFunction* representing the determinant

EXAMPLES:

Jacobian determinant of a set of 2 functions of 2 coordinates:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.multifunction(x-y, x*y)
sage: f.jacobian_det()
x + y
```

The output of *jacobian\_det()* is an instance of *CoordFunction* and can therefore be called on specific values of the coordinates, e.g.  $(x, y) = (1, 2)$ :

```
sage: type(f.jacobian_det())
<class 'sage.manifolds.coord_func_symb.CoordFunctionSymbRing_with_category.
  ↳element_class'>
sage: f.jacobian_det().display()
(x, y) |--> x + y
sage: f.jacobian_det()(1,2)
3
```

The result is cached:

```
sage: f.jacobian_det() is f.jacobian_det()
True
```

We verify the determinant of the Jacobian:

```
sage: f.jacobian_det() == det(matrix([[f[i].diff(j).expr() for j in range(2)]
...:                                for i in range(2)]))
True
```

Jacobian determinant of a set of 3 functions of 3 coordinates:

```
sage: M = Manifold(3, 'M', structure='topological')
sage: X.<x,y,z> = M.chart()
sage: f = X.multifunction(x*y+z^2, z^2*x+y^2*z, (x*y*z)^3)
sage: f.jacobian_det().display()
(x, y, z) |--> 6*x^3*y^5*z^3 - 3*x^4*y^3*z^4 - 12*x^2*y^4*z^5 + 6*x^3*y^2*z^6
```

We verify the determinant of the Jacobian:

```
sage: f.jacobian_det() == det(matrix([[f[i].diff(j).expr() for j in range(3)]
...:                                for i in range(3)]))
True
```

### 1.5.3 Symbolic Coordinate Functions

In the context of a topological manifold  $M$  over a topological field  $K$ , a *coordinate function* is a function from a chart codomain to  $K$ . In other words, a coordinate function is a  $K$ -valued function of the coordinates associated to some

chart.

More precisely, let  $(U, \varphi)$  be a chart on  $M$ , i.e.  $U$  is an open subset of  $M$  and  $\varphi : U \rightarrow V \subset K^n$  is a homeomorphism from  $U$  to an open subset  $V$  of  $K^n$ . A *coordinate function associated to the chart*  $(U, \varphi)$  is a function

$$f : \begin{array}{ccc} V \subset K^n & \longrightarrow & K \\ (x^1, \dots, x^n) & \longmapsto & f(x^1, \dots, x^n) \end{array}$$

This module implements symbolic coordinate functions via the class `CoordFunctionSymb`.

AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2013-2015) : initial version
- Travis Scrimshaw (2016) : make coordinate functions elements of `CoordFunctionSymbRing`.

**class** `sage.manifolds.coord_func_symb.CoordFunctionSymb` (*parent, expression*)

Bases: `sage.manifolds.coord_func.CoordFunction`

Coordinate function with symbolic representation.

If  $(U, \varphi)$  is a chart on a topological manifold  $M$  of dimension  $n$  over a topological field  $K$ , a *coordinate function* associated to  $(U, \varphi)$  is a map

$$f : \begin{array}{ccc} V \subset K^n & \longrightarrow & K \\ (x^1, \dots, x^n) & \longmapsto & f(x^1, \dots, x^n), \end{array}$$

where  $V$  is the codomain of  $\varphi$ . In other words,  $f$  is a  $K$ -valued function of the coordinates associated to the chart  $(U, \varphi)$ .

INPUT:

- `parent` – the algebra of coordinate functions on the chart  $(U, \varphi)$
- `expression` – a symbolic expression representing  $f(x^1, \dots, x^n)$ , where  $(x^1, \dots, x^n)$  are the coordinates of the chart  $(U, \varphi)$

EXAMPLES:

A symbolic coordinate function associated with a 2-dimensional chart:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x^2+3*y+1)
sage: type(f)
<class 'sage.manifolds.coord_func_symb.CoordFunctionSymbRing_with_category.
↳element_class'>
sage: f.display()
(x, y) |--> x^2 + 3*y + 1
sage: f(x,y)
x^2 + 3*y + 1
```

The symbolic expression is returned when asking the direct display of the function:

```
sage: f
x^2 + 3*y + 1
sage: latex(f)
x^{2} + 3 \, y + 1
```

A similar output is obtained by means of the method `expr()`:

```
sage: f.expr()
x^2 + 3*y + 1
```

The value of the function at specified coordinates is obtained by means of the standard parentheses notation:

```
sage: f(2,-1)
2
sage: var('a b')
(a, b)
sage: f(a,b)
a^2 + 3*b + 1
```

An unspecified coordinate function:

```
sage: g = X.function(function('G')(x, y))
sage: g
G(x, y)
sage: g.display()
(x, y) |--> G(x, y)
sage: g.expr()
G(x, y)
sage: g(2,3)
G(2, 3)
```

Coordinate functions can be compared to other values:

```
sage: f = X.function(x^2+3*y+1)
sage: f == 2
False
sage: f == x^2 + 3*y + 1
True
sage: g = X.function(x*y)
sage: f == g
False
sage: h = X.function(x^2+3*y+1)
sage: f == h
True
```

### Differences between `CoordFunctionSymb` and callable symbolic expressions

Callable symbolic expressions are defined directly from symbolic expressions of the coordinates:

```
sage: f0(x,y) = x^2 + 3*y + 1
sage: type(f0)
<type 'sage.symbolic.expression.Expression'>
sage: f0
(x, y) |--> x^2 + 3*y + 1
sage: f0(x,y)
x^2 + 3*y + 1
```

To get an output similar to that of `f0` for the coordinate function `f`, we must use the method `display()`:

```
sage: f
x^2 + 3*y + 1
sage: f.display()
(x, y) |--> x^2 + 3*y + 1
```

```
sage: f(x,y)
x^2 + 3*y + 1
```

More importantly, instances of *CoordFunctionSymb* differ from callable symbolic expression by the automatic simplifications in all operations. For instance, adding the two callable symbolic expressions:

```
sage: f0(x,y,z) = cos(x)^2 ; g0(x,y,z) = sin(x)^2
```

results in:

```
sage: f0 + g0
(x, y, z) |--> cos(x)^2 + sin(x)^2
```

To get 1, one has to call `simplify_trig()`:

```
sage: (f0 + g0).simplify_trig()
(x, y, z) |--> 1
```

On the contrary, the sum of the corresponding *CoordFunctionSymb* instances is automatically simplified (see `simplify_chain_real()` and `simplify_chain_generic()` for details):

```
sage: f = X.function(cos(x)^2) ; g = X.function(sin(x)^2)
sage: f + g
1
```

Another difference regards the display of partial derivatives: for callable symbolic functions, it relies on Pynac notation `D[0]`, `D[1]`, etc.:

```
sage: g = function('g')(x, y)
sage: f0(x,y) = diff(g, x) + diff(g, y)
sage: f0
(x, y) |--> diff(g(x, y), x) + diff(g(x, y), y)
```

while for coordinate functions, the display is more “textbook” like:

```
sage: f = X.function(diff(g, x) + diff(g, y))
sage: f
d(g)/dx + d(g)/dy
```

The difference is even more dramatic on LaTeX outputs:

```
sage: latex(f0)
\left( x, y \right) \ {\mapsto} \ \frac{\partial}{\partial x}g\left(x, y\right) +
\frac{\partial}{\partial y}g\left(x, y\right)
sage: latex(f)
\frac{\partial g}{\partial x} + \frac{\partial g}{\partial y}
```

Note that this regards only the display of coordinate functions: internally, the Pynac notation is still used, as we can check by asking for the symbolic expression stored in `f`:

```
sage: f.expr()
diff(g(x, y), x) + diff(g(x, y), y)
```

One can switch to Pynac notation by changing the options:

```
sage: Manifold.options.textbook_output=False
sage: latex(f)
```

```

\frac{\partial}{\partial x}g\left(x, y\right) + \frac{\partial}{\partial y}
↪g\left(x, y\right)
sage: Manifold.options._reset()
sage: latex(f)
\frac{\partial}{\partial x}g + \frac{\partial}{\partial y}

```

Another difference between *CoordFunctionSymb* and callable symbolic expression is the possibility to switch off the display of the arguments of unspecified functions. Consider for instance:

```

sage: f = X.function(function('u')(x, y) * function('v')(x, y))
sage: f
u(x, y)*v(x, y)
sage: f0(x,y) = function('u')(x, y) * function('v')(x, y)
sage: f0
(x, y) |--> u(x, y)*v(x, y)

```

If there is a clear understanding that  $u$  and  $v$  are functions of  $(x, y)$ , the explicit mention of the latter can be cumbersome in lengthy tensor expressions. We can switch it off by:

```

sage: Manifold.options.omit_function_arguments=True
sage: f
u*v

```

Note that neither the callable symbolic expression `f0` nor the internal expression of `f` is affected by the above command:

```

sage: f0
(x, y) |--> u(x, y)*v(x, y)
sage: f.expr()
u(x, y)*v(x, y)

```

We revert to the default behavior by:

```

sage: Manifold.options._reset()
sage: f
u(x, y)*v(x, y)

```

**arccos()**

Arc cosine of self.

OUTPUT:

- coordinate function  $\arccos(f)$ , where  $f$  is the current coordinate function

EXAMPLES:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x*y)
sage: f.arccos()
arccos(x*y)
sage: arccos(f) # equivalent to f.arccos()
arccos(x*y)
sage: acos(f) # equivalent to f.arccos()
arccos(x*y)
sage: arccos(f).display()
(x, y) |--> arccos(x*y)
sage: arccos(X.zero_function()).display()
(x, y) |--> 1/2*pi

```

**arccosh()**Inverse hyperbolic cosine of `self`.

OUTPUT:

- coordinate function  $\operatorname{arccosh}(f)$ , where  $f$  is the current coordinate function

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x*y)
sage: f.arccosh()
arccosh(x*y)
sage: arccosh(f) # equivalent to f.arccosh()
arccosh(x*y)
sage: acosh(f) # equivalent to f.arccosh()
arccosh(x*y)
sage: arccosh(f).display()
(x, y) |--> arccosh(x*y)
sage: arccosh(X.function(1)) == X.zero_function()
True
```

**arcsin()**Arc sine of `self`.

OUTPUT:

- coordinate function  $\operatorname{arcsin}(f)$ , where  $f$  is the current coordinate function

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x*y)
sage: f.arcsin()
arcsin(x*y)
sage: arcsin(f) # equivalent to f.arcsin()
arcsin(x*y)
sage: asin(f) # equivalent to f.arcsin()
arcsin(x*y)
sage: arcsin(f).display()
(x, y) |--> arcsin(x*y)
sage: arcsin(X.zero_function()) == X.zero_function()
True
```

**arcsinh()**Inverse hyperbolic sine of `self`.

OUTPUT:

- coordinate function  $\operatorname{arsinh}(f)$ , where  $f$  is the current coordinate function

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x*y)
sage: f.arcsinh()
arcsinh(x*y)
```

```

sage: arcsinh(f) # equivalent to f.arcsinh()
arcsinh(x*y)
sage: asinh(f) # equivalent to f.arcsinh()
arcsinh(x*y)
sage: arcsinh(f).display()
(x, y) |--> arcsinh(x*y)
sage: arcsinh(X.zero_function()) == X.zero_function()
True

```

**arctan()**

Arc tangent of `self`.

OUTPUT:

- coordinate function  $\arctan(f)$ , where  $f$  is the current coordinate function

EXAMPLES:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x*y)
sage: f.arctan()
arctan(x*y)
sage: arctan(f) # equivalent to f.arctan()
arctan(x*y)
sage: atan(f) # equivalent to f.arctan()
arctan(x*y)
sage: arctan(f).display()
(x, y) |--> arctan(x*y)
sage: arctan(X.zero_function()) == X.zero_function()
True

```

**arctanh()**

Inverse hyperbolic tangent of `self`.

OUTPUT:

- coordinate function  $\operatorname{arctanh}(f)$ , where  $f$  is the current coordinate function

EXAMPLES:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x*y)
sage: f.arctanh()
arctanh(x*y)
sage: arctanh(f) # equivalent to f.arctanh()
arctanh(x*y)
sage: atanh(f) # equivalent to f.arctanh()
arctanh(x*y)
sage: arctanh(f).display()
(x, y) |--> arctanh(x*y)
sage: arctanh(X.zero_function()) == X.zero_function()
True

```

**collect(s)**

Collect the coefficients of  $s$  in the expression of `self` into a group.

INPUT:

- $s$  – the symbol whose coefficients will be collected

OUTPUT:

- self with the coefficients of  $s$  grouped in its expression

EXAMPLES:

Action on a 2-dimensional coordinate function:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x^2*y + x*y + (x*y)^2)
sage: f.display()
(x, y) |--> x^2*y^2 + x^2*y + x*y
sage: f.collect(y)
x^2*y^2 + (x^2 + x)*y
```

The method `collect()` has changed the expression of  $f$ :

```
sage: f.display()
(x, y) |--> x^2*y^2 + (x^2 + x)*y
```

### `collect_common_factors()`

Collect common factors in the expression of `self`.

This method does not perform a full factorization but only looks for factors which are already explicitly present.

OUTPUT:

- self with the common factors collected in its expression

EXAMPLES:

Action on a 2-dimensional coordinate function:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x/(x^2*y + x*y))
sage: f.display()
(x, y) |--> x/(x^2*y + x*y)
sage: f.collect_common_factors()
1/((x + 1)*y)
```

The method `collect_common_factors()` has changed the expression of  $f$ :

```
sage: f.display()
(x, y) |--> 1/((x + 1)*y)
```

### `copy()`

Return an exact copy of the object.

OUTPUT:

- a *CoordFunctionSymb*

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x+y^2)
sage: g = f.copy(); g
y^2 + x
```

By construction,  $g$  is identical to  $f$ :

```
sage: type(g) == type(f)
True
sage: g == f
True
```

but it is not the same object:

```
sage: g is f
False
```

**cos()**

Cosine of `self`.

OUTPUT:

- coordinate function  $\cos(f)$ , where  $f$  is the current coordinate function

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x*y)
sage: f.cos()
cos(x*y)
sage: cos(f) # equivalent to f.cos()
cos(x*y)
sage: cos(f).display()
(x, y) |--> cos(x*y)
sage: cos(X.zero_function()).display()
(x, y) |--> 1
```

**cosh()**

Hyperbolic cosine of `self`.

OUTPUT:

- coordinate function  $\cosh(f)$ , where  $f$  is the current coordinate function

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x*y)
sage: f.cosh()
cosh(x*y)
sage: cosh(f) # equivalent to f.cosh()
cosh(x*y)
sage: cosh(f).display()
(x, y) |--> cosh(x*y)
sage: cosh(X.zero_function()).display()
(x, y) |--> 1
```

**diff(coord)**

Partial derivative with respect to a coordinate.

INPUT:

- `coord` – either the coordinate  $x^i$  with respect to which the derivative of the coordinate function  $f$  is to be taken, or the index  $i$  labelling this coordinate (with the index convention defined on the chart)

domain via the parameter `start_index`)

OUTPUT:

- a `CoordFunctionSymb` representing the partial derivative  $\frac{\partial f}{\partial x^i}$

EXAMPLES:

Partial derivatives of a 2-dimensional coordinate function:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x^2+3*y+1); f
x^2 + 3*y + 1
sage: f.diff(x)
2*x
sage: f.diff(y)
3
```

Each partial derivatives is itself a coordinate function:

```
sage: type(f.diff(x))
<class 'sage.manifolds.coord_func_symb.CoordFunctionSymbRing_with_category.
↳element_class'>
```

An index can be used instead of the coordinate symbol:

```
sage: f.diff(0)
2*x
sage: f.diff(1)
3
```

The index range depends on the convention used on the chart's domain:

```
sage: M = Manifold(2, 'M', structure='topological', start_index=1)
sage: X.<x,y> = M.chart()
sage: f = X.function(x^2+3*y+1)
sage: f.diff(0)
Traceback (most recent call last):
...
ValueError: coordinate index out of range
sage: f.diff(1)
2*x
sage: f.diff(2)
3
```

**disp()**

Display `self` in arrow notation.

The output is either text-formatted (console mode) or LaTeX-formatted (notebook mode).

EXAMPLES:

Coordinate function on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(cos(x*y/2))
sage: f.display()
(x, y) |--> cos(1/2*x*y)
```

```
sage: latex(f.display())
\left(x, y\right) \mapsto \cos\left(\frac{1}{2} \, x \, y\right)
```

A shortcut is `disp()`:

```
sage: f.disp()
(x, y) |--> cos(1/2*x*y)
```

Display of the zero function:

```
sage: X.zero_function().display()
(x, y) |--> 0
```

### `display()`

Display `self` in arrow notation.

The output is either text-formatted (console mode) or LaTeX-formatted (notebook mode).

EXAMPLES:

Coordinate function on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(cos(x*y/2))
sage: f.display()
(x, y) |--> cos(1/2*x*y)
sage: latex(f.display())
\left(x, y\right) \mapsto \cos\left(\frac{1}{2} \, x \, y\right)
```

A shortcut is `disp()`:

```
sage: f.disp()
(x, y) |--> cos(1/2*x*y)
```

Display of the zero function:

```
sage: X.zero_function().display()
(x, y) |--> 0
```

### `exp()`

Exponential of `self`.

OUTPUT:

- coordinate function  $\exp(f)$ , where  $f$  is the current coordinate function

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x+y)
sage: f.exp()
e^(x + y)
sage: exp(f) # equivalent to f.exp()
e^(x + y)
sage: exp(f).display()
(x, y) |--> e^(x + y)
sage: exp(X.zero_function())
1
```

**expand()**

Expand the coordinate expression of `self`.

OUTPUT:

- `self` with its expression expanded

EXAMPLES:

Expanding a 2-dimensional coordinate function:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function((x - y)^2)
sage: f.display()
(x, y) |--> (x - y)^2
sage: f.expand()
x^2 - 2*x*y + y^2
```

The method `expand()` has changed the expression of `f`:

```
sage: f.display()
(x, y) |--> x^2 - 2*x*y + y^2
```

**expr()**

Return the symbolic expression representing the image of `self`.

OUTPUT:

- symbolic expression involving the chart coordinates

EXAMPLES:

Coordinate function of a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x^2+3*y+1)
sage: f.expr()
x^2 + 3*y + 1
sage: type(f.expr())
<type 'sage.symbolic.expression.Expression'>
```

For a symbolic coordinate function, one shall always have:

```
sage: bool( f.expr() == f(*(f.chart()[ : ])) )
True
```

The method `expr()` is useful for accessing to all the symbolic expression functionalities in Sage; for instance:

```
sage: var('a')
a
sage: f = X.function(a*x*y); f.display()
(x, y) |--> a*x*y
sage: f.expr()
a*x*y
sage: f.expr().subs(a=2)
2*x*y
```

Note that for substituting the value of a coordinate, the function call can be used as well:

```
sage: f(x, 3)
3*a*x
sage: bool( f(x, 3) == f.expr().subs(y=3) )
True
```

**factor()**

Factorize the coordinate expression of `self`.

OUTPUT:

- `self` with its expression factorized

EXAMPLES:

Factorization of a 2-dimensional coordinate function:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x^2 + 2*x*y + y^2)
sage: f.display()
(x, y) |--> x^2 + 2*x*y + y^2
sage: f.factor()
(x + y)^2
```

The method `factor()` has changed the expression of `f`:

```
sage: f.display()
(x, y) |--> (x + y)^2
```

**log** (*base=None*)

Logarithm of `self`.

INPUT:

- `base` – (default: `None`) base of the logarithm; if `None`, the natural logarithm (i.e. logarithm to base  $e$ ) is returned

OUTPUT:

- coordinate function  $\log_a(f)$ , where  $f$  is the current coordinate function and  $a$  is the base

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x+y)
sage: f.log()
log(x + y)
sage: log(f) # equivalent to f.log()
log(x + y)
sage: log(f).display()
(x, y) |--> log(x + y)
sage: f.log(2)
log(x + y)/log(2)
sage: log(f, 2)
log(x + y)/log(2)
```

**simplify()**

Simplify the coordinate expression of `self`.

For details about the employed chain of simplifications, see `simplify_chain_real()` for coordinate functions on real manifolds and `simplify_chain_generic()` for the generic case.

OUTPUT:

- self with its expression simplified

EXAMPLES:

Simplification of a 2-dimension coordinate function:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(cos(x)^2+sin(x)^2 + sqrt(x^2))
sage: f.display()
(x, y) |--> cos(x)^2 + sin(x)^2 + sqrt(x^2)
sage: f.simplify()
abs(x) + 1
```

The method `simplify()` has changed the expression of `f`:

```
sage: f.display()
(x, y) |--> abs(x) + 1
```

Another example:

```
sage: f = X.function((x^2-1)/(x+1))
sage: f
(x^2 - 1)/(x + 1)
sage: f.simplify()
x - 1
```

Examples taking into account the declared range of a coordinate:

```
sage: M = Manifold(2, 'M_1', structure='topological')
sage: X.<x,y> = M.chart('x:(0,+oo) y')
sage: f = X.function(sqrt(x^2))
sage: f
x
sage: f.simplify()
x
```

```
sage: forget() # to clear the previous assumption on x
sage: M = Manifold(2, 'M_2', structure='topological')
sage: X.<x,y> = M.chart('x:(-oo,0) y')
sage: f = X.function(sqrt(x^2))
sage: f
sqrt(x^2)
sage: f.simplify()
-x
```

**sin()**

Sine of self.

OUTPUT:

- coordinate function  $\sin(f)$ , where  $f$  is the current coordinate function

EXAMPLES:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x*y)
sage: f.sin()
sin(x*y)
sage: sin(f) # equivalent to f.sin()
sin(x*y)
sage: sin(f).display()
(x, y) |--> sin(x*y)
sage: sin(X.zero_function()) == X.zero_function()
True

```

**sinh()**

Hyperbolic sine of self.

OUTPUT:

- coordinate function  $\sinh(f)$ , where  $f$  is the current coordinate function

EXAMPLES:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x*y)
sage: f.sinh()
sinh(x*y)
sage: sinh(f) # equivalent to f.sinh()
sinh(x*y)
sage: sinh(f).display()
(x, y) |--> sinh(x*y)
sage: sinh(X.zero_function()) == X.zero_function()
True

```

**sqrt()**

Square root of self.

OUTPUT:

- coordinate function  $\sqrt{f}$ , where  $f$  is the current coordinate function

EXAMPLES:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x+y)
sage: f.sqrt()
sqrt(x + y)
sage: sqrt(f) # equivalent to f.sqrt()
sqrt(x + y)
sage: sqrt(f).display()
(x, y) |--> sqrt(x + y)
sage: sqrt(X.zero_function()).display()
(x, y) |--> 0

```

**tan()**

Tangent of self.

OUTPUT:

- coordinate function  $\tan(f)$ , where  $f$  is the current coordinate function

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x*y)
sage: f.tan()
sin(x*y)/cos(x*y)
sage: tan(f) # equivalent to f.tan()
sin(x*y)/cos(x*y)
sage: tan(f).display()
(x, y) |--> sin(x*y)/cos(x*y)
sage: tan(X.zero_function()) == X.zero_function()
True
```

**tanh()**

Hyperbolic tangent of self.

OUTPUT:

- coordinate function  $\tanh(f)$ , where  $f$  is the current coordinate function

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x*y)
sage: f.tanh()
sinh(x*y)/cosh(x*y)
sage: tanh(f) # equivalent to f.tanh()
sinh(x*y)/cosh(x*y)
sage: tanh(f).display()
(x, y) |--> sinh(x*y)/cosh(x*y)
sage: tanh(X.zero_function()) == X.zero_function()
True
```

**class** sage.manifolds.coord\_func\_symb.**CoordFunctionSymbRing**(*chart*)

Bases: sage.structure.parent.Parent, sage.structure.unique\_representation.UniqueRepresentation

Ring of all symbolic coordinate functions on a chart.

INPUT:

- chart* – a coordinate chart, as an instance of class *Chart*

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: FR = X.function_ring(); FR
Ring of coordinate functions on Chart (M, (x, y))
sage: type(FR)
<class 'sage.manifolds.coord_func_symb.CoordFunctionSymbRing_with_category'>
sage: FR.category()
Category of commutative algebras over Symbolic Ring
```

**Element**

alias of *CoordFunctionSymb*

**from\_base\_ring**(*r*)

Return the canonical embedding of *r* into self.

INPUT:

- `r` – an element of `self.base_ring()`

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: FR = X.function_ring()
sage: f = FR.from_base_ring(x*y)
sage: f.display()
(x, y) |--> x*y
```

**is\_field()**

Return False as self is not an integral domain.

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: FR = X.function_ring()
sage: FR.is_integral_domain()
False
sage: FR.is_field()
False
```

**is\_integral\_domain()**

Return False as self is not an integral domain.

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: FR = X.function_ring()
sage: FR.is_integral_domain()
False
sage: FR.is_field()
False
```

**one()**

Return the constant function 1 in self.

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: FR = X.function_ring()
sage: FR.one()
1

sage: M = Manifold(2, 'M', structure='topological', field=Qp(5))
sage: X.<x,y> = M.chart()
sage: X.function_ring().one()
1 + O(5^20)
```

**zero()**

Return the constant function 0 in self.

EXAMPLES:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: FR = X.function_ring()
sage: FR.zero()
0

sage: M = Manifold(2, 'M', structure='topological', field=Qp(5))
sage: X.<x,y> = M.chart()
sage: X.function_ring().zero()
0

```

## 1.6 Scalar Fields

### 1.6.1 Algebra of Scalar Fields

The class *ScalarFieldAlgebra* implements the commutative algebra  $C^0(M)$  of scalar fields on a topological manifold  $M$  over a topological field  $K$ . By *scalar field*, it is meant a continuous function  $M \rightarrow K$ . The set  $C^0(M)$  is an algebra over  $K$ , whose ring product is the pointwise multiplication of  $K$ -valued functions, which is clearly commutative.

AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2014-2015): initial version
- Travis Scrimshaw (2016): review tweaks

REFERENCES:

- [Lee2011]
- [KN1963]

**class** sage.manifolds.scalarfield\_algebra.**ScalarFieldAlgebra** (*domain*)  
**Bases:** sage.structure.unique\_representation.UniqueRepresentation, sage.structure.parent.Parent

Commutative algebra of scalar fields on a topological manifold.

If  $M$  is a topological manifold over a topological field  $K$ , the commutative algebra of scalar fields on  $M$  is the set  $C^0(M)$  of all continuous maps  $M \rightarrow K$ . The set  $C^0(M)$  is an algebra over  $K$ , whose ring product is the pointwise multiplication of  $K$ -valued functions, which is clearly commutative.

If  $K = \mathbf{R}$  or  $K = \mathbf{C}$ , the field  $K$  over which the algebra  $C^0(M)$  is constructed is represented by the Symbolic Ring SR, since there is no exact representation of  $\mathbf{R}$  nor  $\mathbf{C}$ .

INPUT:

- *domain* – the topological manifold  $M$  on which the scalar fields are defined

EXAMPLES:

Algebras of scalar fields on the sphere  $S^2$  and on some open subsets of it:

```

sage: M = Manifold(2, 'M', structure='topological') # the 2-dimensional sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V) # S^2 is the union of U and V

```

```

sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                               intersection_name='W',
....:                               restrictions1= x^2+y^2!=0,
....:                               restrictions2= u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: CM = M.scalar_field_algebra(); CM
Algebra of scalar fields on the 2-dimensional topological manifold M
sage: W = U.intersection(V) # S^2 minus the two poles
sage: CW = W.scalar_field_algebra(); CW
Algebra of scalar fields on the Open subset W of the
2-dimensional topological manifold M

```

$C^0(M)$  and  $C^0(W)$  belong to the category of commutative algebras over  $\mathbf{R}$  (represented here by SymbolicRing):

```

sage: CM.category()
Category of commutative algebras over Symbolic Ring
sage: CM.base_ring()
Symbolic Ring
sage: CW.category()
Category of commutative algebras over Symbolic Ring
sage: CW.base_ring()
Symbolic Ring

```

The elements of  $C^0(M)$  are scalar fields on  $M$ :

```

sage: CM.an_element()
Scalar field on the 2-dimensional topological manifold M
sage: CM.an_element().display() # this sample element is a constant field
M --> R
on U: (x, y) |--> 2
on V: (u, v) |--> 2

```

Those of  $C^0(W)$  are scalar fields on  $W$ :

```

sage: CW.an_element()
Scalar field on the Open subset W of the 2-dimensional topological
manifold M
sage: CW.an_element().display() # this sample element is a constant field
W --> R
(x, y) |--> 2
(u, v) |--> 2

```

The zero element:

```

sage: CM.zero()
Scalar field zero on the 2-dimensional topological manifold M
sage: CM.zero().display()
zero: M --> R
on U: (x, y) |--> 0
on V: (u, v) |--> 0

```

```

sage: CW.zero()
Scalar field zero on the Open subset W of the 2-dimensional
topological manifold M
sage: CW.zero().display()
zero: W --> R

```

```
(x, y) |--> 0
(u, v) |--> 0
```

The unit element:

```
sage: CM.one()
Scalar field 1 on the 2-dimensional topological manifold M
sage: CM.one().display()
1: M --> R
on U: (x, y) |--> 1
on V: (u, v) |--> 1
```

```
sage: CW.one()
Scalar field 1 on the Open subset W of the 2-dimensional topological
manifold M
sage: CW.one().display()
1: W --> R
(x, y) |--> 1
(u, v) |--> 1
```

A generic element can be constructed by using a dictionary of the coordinate expressions defining the scalar field:

```
sage: f = CM({c_xy: atan(x^2+y^2), c_uv: pi/2 - atan(u^2+v^2)}); f
Scalar field on the 2-dimensional topological manifold M
sage: f.display()
M --> R
on U: (x, y) |--> arctan(x^2 + y^2)
on V: (u, v) |--> 1/2*pi - arctan(u^2 + v^2)
sage: f.parent()
Algebra of scalar fields on the 2-dimensional topological manifold M
```

Specific elements can also be constructed in this way:

```
sage: CM(0) == CM.zero()
True
sage: CM(1) == CM.one()
True
```

Note that the zero scalar field is cached:

```
sage: CM(0) is CM.zero()
True
```

Elements can also be constructed by means of the method `scalar_field()` acting on the domain (this allows one to set the name of the scalar field at the construction):

```
sage: f1 = M.scalar_field({c_xy: atan(x^2+y^2), c_uv: pi/2 - atan(u^2+v^2)},
....:                      name='f')
sage: f1.parent()
Algebra of scalar fields on the 2-dimensional topological manifold M
sage: f1 == f
True
sage: M.scalar_field(0, chart='all') == CM.zero()
True
```

The algebra  $C^0(M)$  coerces to  $C^0(W)$  since  $W$  is an open subset of  $M$ :

```
sage: CW.has_coerce_map_from(CM)
True
```

The reverse is of course false:

```
sage: CM.has_coerce_map_from(CW)
False
```

The coercion map is nothing but the restriction to  $W$  of scalar fields on  $M$ :

```
sage: fW = CW(f) ; fW
Scalar field on the Open subset W of the
2-dimensional topological manifold M
sage: fW.display()
W --> R
(x, y) |--> arctan(x^2 + y^2)
(u, v) |--> 1/2*pi - arctan(u^2 + v^2)
```

```
sage: CW(CM.one()) == CW.one()
True
```

The coercion map allows for the addition of elements of  $C^0(W)$  with elements of  $C^0(M)$ , the result being an element of  $C^0(W)$ :

```
sage: s = fW + f
sage: s.parent()
Algebra of scalar fields on the Open subset W of the
2-dimensional topological manifold M
sage: s.display()
W --> R
(x, y) |--> 2*arctan(x^2 + y^2)
(u, v) |--> pi - 2*arctan(u^2 + v^2)
```

Another coercion is that from the Symbolic Ring. Since the Symbolic Ring is the base ring for the algebra  $CM$ , the coercion of a symbolic expression  $s$  is performed by the operation  $s*CM.one()$ , which invokes the (reflected) multiplication operator. If the symbolic expression does not involve any chart coordinate, the outcome is a constant scalar field:

```
sage: h = CM(pi*sqrt(2)) ; h
Scalar field on the 2-dimensional topological manifold M
sage: h.display()
M --> R
on U: (x, y) |--> sqrt(2)*pi
on V: (u, v) |--> sqrt(2)*pi
sage: a = var('a')
sage: h = CM(a); h.display()
M --> R
on U: (x, y) |--> a
on V: (u, v) |--> a
```

If the symbolic expression involves some coordinate of one of the manifold's charts, the outcome is initialized only on the chart domain:

```
sage: h = CM(a+x); h.display()
M --> R
on U: (x, y) |--> a + x
sage: h = CM(a+u); h.display()
```

```
M --> R
on V: (u, v) |--> a + u
```

If the symbolic expression involves coordinates of different charts, the scalar field is created as a Python object, but is not initialized, in order to avoid any ambiguity:

```
sage: h = CM(x+u); h.display()
M --> R
```

```
sage: s = f - h ; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M --> R
on U: (x, y) |--> -sqrt(2)*pi + arctan(x^2 + y^2)
on V: (u, v) |--> -1/2*pi*(2*sqrt(2) - 1) - arctan(u^2 + v^2)
```

```
sage: s = f*h ; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M --> R
on U: (x, y) |--> sqrt(2)*pi*arctan(x^2 + y^2)
on V: (u, v) |--> 1/2*sqrt(2)*(pi^2 - 2*pi*arctan(u^2 + v^2))
```

```
sage: s = f/h ; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M --> R
on U: (x, y) |--> 1/2*sqrt(2)*arctan(x^2 + y^2)/pi
on V: (u, v) |--> 1/4*(sqrt(2)*pi - 2*sqrt(2)*arctan(u^2 + v^2))/pi
```

```
sage: f*(h+f) == f*h + f*f
True
```

Ring laws with coercion:

```
sage: f - fW == CW.zero()
True
sage: f/fW == CW.one()
True
sage: s = f*fW ; s
Scalar field on the Open subset W of the 2-dimensional topological manifold M
sage: s.display()
W --> R
(x, y) |--> arctan(x^2 + y^2)^2
(u, v) |--> 1/4*pi^2 - pi*arctan(u^2 + v^2) + arctan(u^2 + v^2)^2
sage: s/f == fW
True
```

Multiplication by a real number:

```
sage: s = 2*f ; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M --> R
on U: (x, y) |--> 2*arctan(x^2 + y^2)
```

```
on V: (u, v) |--> pi - 2*arctan(u^2 + v^2)
```

```
sage: 0*f == CM.zero()
True
sage: 1*f == f
True
sage: 2*(f/2) == f
True
sage: (f+2*f)/3 == f
True
sage: 1/3*(f+2*f) == f
True
```

The Sage test suite for algebras is passed:

```
sage: TestSuite(CM).run()
```

It is passed also for  $C^0(W)$ :

```
sage: TestSuite(CW).run()
```

### Element

alias of `ScalarField`

### one()

Return the unit element of the algebra.

This is nothing but the constant scalar field 1 on the manifold, where 1 is the unit element of the base field.

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: CM = M.scalar_field_algebra()
sage: h = CM.one(); h
Scalar field 1 on the 2-dimensional topological manifold M
sage: h.display()
1: M --> R
   (x, y) |--> 1
```

The result is cached:

```
sage: CM.one() is h
True
```

### zero()

Return the zero element of the algebra.

This is nothing but the constant scalar field 0 on the manifold, where 0 is the zero element of the base field.

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: CM = M.scalar_field_algebra()
sage: z = CM.zero(); z
Scalar field zero on the 2-dimensional topological manifold M
sage: z.display()
```

```
zero: M --> R
      (x, y) |--> 0
```

The result is cached:

```
sage: CM.zero() is z
True
```

## 1.6.2 Scalar Fields

Given a topological manifold  $M$  over a topological field  $K$  (in most applications,  $K = \mathbf{R}$  or  $K = \mathbf{C}$ ), a *scalar field* on  $M$  is a continuous map

$$f : M \longrightarrow K$$

Scalar fields are implemented by the class *ScalarField*.

AUTHORS:

- Ericourgoulhon, Michal Bejger (2013-2015): initial version
- Travis Scrimshaw (2016): review tweaks

REFERENCES:

- [Lee2011]
- [KN1963]

**class** sage.manifolds.scalarfield.**ScalarField**(parent, coord\_expression=None, chart=None, name=None, latex\_name=None)

Bases: sage.structure.element.CommutativeAlgebraElement

Scalar field on a topological manifold.

Given a topological manifold  $M$  over a topological field  $K$  (in most applications,  $K = \mathbf{R}$  or  $K = \mathbf{C}$ ), a *scalar field* on  $M$  is a continuous map

$$f : M \longrightarrow K.$$

A scalar field on  $M$  is an element of the commutative algebra  $C^0(M)$  (see *ScalarFieldAlgebra*).

INPUT:

- `parent` – the algebra of scalar fields containing the scalar field (must be an instance of class *ScalarFieldAlgebra*)
- `coord_expression` – (default: None) coordinate expression(s) of the scalar field; this can be either
  - a dictionary of coordinate expressions in various charts on the domain, with the charts as keys;
  - a single coordinate expression; if the argument `chart` is 'all', this expression is set to all the charts defined on the open set; otherwise, the expression is set in the specific chart provided by the argument `chart`
- `chart` – (default: None) chart defining the coordinates used in `coord_expression` when the latter is a single coordinate expression; if none is provided (default), the default chart of the open set is assumed. If `chart=='all'`, `coord_expression` is assumed to be independent of the chart (constant scalar field).
- `name` – (default: None) string; name (symbol) given to the scalar field

- `latex_name` – (default: None) string; LaTeX symbol to denote the scalar field; if none is provided, the LaTeX symbol is set to `name`

If `coord_expression` is None or incomplete, coordinate expressions can be added after the creation of the object, by means of the methods `add_expr()`, `add_expr_by_continuation()` and `set_expr()`.

EXAMPLES:

A scalar field on the 2-sphere:

```
sage: M = Manifold(2, 'M', structure='topological') # the 2-dimensional sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                                     intersection_name='W',
....:                                     restrictions1= x^2+y^2!=0,
....:                                     restrictions2= u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: f = M.scalar_field({c_xy: 1/(1+x^2+y^2), c_uv: (u^2+v^2)/(1+u^2+v^2)},
....:                    name='f') ; f
Scalar field f on the 2-dimensional topological manifold M
sage: f.display()
f: M --> R
on U: (x, y) |--> 1/(x^2 + y^2 + 1)
on V: (u, v) |--> (u^2 + v^2)/(u^2 + v^2 + 1)
```

For scalar fields defined by a single coordinate expression, the latter can be passed instead of the dictionary over the charts:

```
sage: g = U.scalar_field(x*y, chart=c_xy, name='g') ; g
Scalar field g on the Open subset U of the 2-dimensional topological
manifold M
```

The above is indeed equivalent to:

```
sage: g = U.scalar_field({c_xy: x*y}, name='g') ; g
Scalar field g on the Open subset U of the 2-dimensional topological
manifold M
```

Since `c_xy` is the default chart of `U`, the argument `chart` can be skipped:

```
sage: g = U.scalar_field(x*y, name='g') ; g
Scalar field g on the Open subset U of the 2-dimensional topological
manifold M
```

The scalar field  $g$  is defined on  $U$  and has an expression in terms of the coordinates  $(u, v)$  on  $W = U \cap V$ :

```
sage: g.display()
g: U --> R
(x, y) |--> x*y
on W: (u, v) |--> u*v/(u^4 + 2*u^2*v^2 + v^4)
```

Scalar fields on  $M$  can also be declared with a single chart:

```
sage: f = M.scalar_field(1/(1+x^2+y^2), chart=c_xy, name='f') ; f
Scalar field f on the 2-dimensional topological manifold M
```

Their definition must then be completed by providing the expressions on other charts, via the method `add_expr()`, to get a global cover of the manifold:

```
sage: f.add_expr((u^2+v^2)/(1+u^2+v^2), chart=c_uv)
sage: f.display()
f: M --> R
on U: (x, y) |--> 1/(x^2 + y^2 + 1)
on V: (u, v) |--> (u^2 + v^2)/(u^2 + v^2 + 1)
```

We can even first declare the scalar field without any coordinate expression and provide them subsequently:

```
sage: f = M.scalar_field(name='f')
sage: f.add_expr(1/(1+x^2+y^2), chart=c_xy)
sage: f.add_expr((u^2+v^2)/(1+u^2+v^2), chart=c_uv)
sage: f.display()
f: M --> R
on U: (x, y) |--> 1/(x^2 + y^2 + 1)
on V: (u, v) |--> (u^2 + v^2)/(u^2 + v^2 + 1)
```

We may also use the method `add_expr_by_continuation()` to complete the coordinate definition using the analytic continuation from domains in which charts overlap:

```
sage: f = M.scalar_field(1/(1+x^2+y^2), chart=c_xy, name='f') ; f
Scalar field f on the 2-dimensional topological manifold M
sage: f.add_expr_by_continuation(c_uv, U.intersection(V))
sage: f.display()
f: M --> R
on U: (x, y) |--> 1/(x^2 + y^2 + 1)
on V: (u, v) |--> (u^2 + v^2)/(u^2 + v^2 + 1)
```

A scalar field can also be defined by some unspecified function of the coordinates:

```
sage: h = U.scalar_field(function('H')(x, y), name='h') ; h
Scalar field h on the Open subset U of the 2-dimensional topological
manifold M
sage: h.display()
h: U --> R
(x, y) |--> H(x, y)
on W: (u, v) |--> H(u/(u^2 + v^2), v/(u^2 + v^2))
```

We may use the argument `latex_name` to specify the LaTeX symbol denoting the scalar field if the latter is different from name:

```
sage: latex(f)
f
sage: f = M.scalar_field({c_xy: 1/(1+x^2+y^2), c_uv: (u^2+v^2)/(1+u^2+v^2)},
.....:                    name='f', latex_name=r'\mathcal{F}')
sage: latex(f)
\mathcal{F}
```

The coordinate expression in a given chart is obtained via the method `expr()`, which returns a symbolic expression:

```
sage: f.expr(c_uv)
(u^2 + v^2)/(u^2 + v^2 + 1)
sage: type(f.expr(c_uv))
<type 'sage.symbolic.expression.Expression'>
```

The method `coord_function()` returns instead a function of the chart coordinates, i.e. an instance of `CoordFunction`:

```
sage: f.coord_function(c_uv)
(u^2 + v^2)/(u^2 + v^2 + 1)
sage: type(f.coord_function(c_uv))
<class 'sage.manifolds.coord_func_symb.CoordFunctionSymbRing_with_category.
↳element_class'>
sage: f.coord_function(c_uv).display()
(u, v) |--> (u^2 + v^2)/(u^2 + v^2 + 1)
```

The value returned by the method `expr()` is actually the coordinate expression of the chart function:

```
sage: f.expr(c_uv) is f.coord_function(c_uv).expr()
True
```

A constant scalar field is declared by setting the argument `chart` to `'all'`:

```
sage: c = M.scalar_field(2, chart='all', name='c') ; c
Scalar field c on the 2-dimensional topological manifold M
sage: c.display()
c: M --> R
on U: (x, y) |--> 2
on V: (u, v) |--> 2
```

A shortcut is to use the method `constant_scalar_field()`:

```
sage: c == M.constant_scalar_field(2)
True
```

The constant value can be some unspecified parameter:

```
sage: var('a')
a
sage: c = M.constant_scalar_field(a, name='c') ; c
Scalar field c on the 2-dimensional topological manifold M
sage: c.display()
c: M --> R
on U: (x, y) |--> a
on V: (u, v) |--> a
```

A special case of constant field is the zero scalar field:

```
sage: zer = M.constant_scalar_field(0) ; zer
Scalar field zero on the 2-dimensional topological manifold M
sage: zer.display()
zero: M --> R
on U: (x, y) |--> 0
on V: (u, v) |--> 0
```

It can be obtained directly by means of the function `zero_scalar_field()`:

```
sage: zer is M.zero_scalar_field()
True
```

A third way is to get it as the zero element of the algebra  $C^0(M)$  of scalar fields on  $M$  (see below):

```
sage: zer is M.scalar_field_algebra().zero()
True
```

By definition, a scalar field acts on the manifold's points, sending them to elements of the manifold's base field (real numbers in the present case):

```
sage: N = M.point((0,0), chart=c_uv) # the North pole
sage: S = M.point((0,0), chart=c_xy) # the South pole
sage: E = M.point((1,0), chart=c_xy) # a point at the equator
sage: f(N)
0
sage: f(S)
1
sage: f(E)
1/2
sage: h(E)
H(1, 0)
sage: c(E)
a
sage: zer(E)
0
```

A scalar field can be compared to another scalar field:

```
sage: f == g
False
```

...to a symbolic expression:

```
sage: f == x*y
False
sage: g == x*y
True
sage: c == a
True
```

...to a number:

```
sage: f == 2
False
sage: zer == 0
True
```

...to anything else:

```
sage: f == M
False
```

Standard mathematical functions are implemented:

```
sage: sqrt(f)
Scalar field sqrt(f) on the 2-dimensional topological manifold M
sage: sqrt(f).display()
sqrt(f): M --> R
on U: (x, y) |--> 1/sqrt(x^2 + y^2 + 1)
on V: (u, v) |--> sqrt(u^2 + v^2)/sqrt(u^2 + v^2 + 1)
```

```

sage: tan(f)
Scalar field tan(f) on the 2-dimensional topological manifold M
sage: tan(f).display()
tan(f): M --> R
on U: (x, y) |--> sin(1/(x^2 + y^2 + 1))/cos(1/(x^2 + y^2 + 1))
on V: (u, v) |--> sin((u^2 + v^2)/(u^2 + v^2 + 1))/cos((u^2 + v^2)/(u^2 + v^2 + 1))

```

## Arithmetics of scalar fields

Scalar fields on  $M$  (resp.  $U$ ) belong to the algebra  $C^0(M)$  (resp.  $C^0(U)$ ):

```

sage: f.parent()
Algebra of scalar fields on the 2-dimensional topological manifold M
sage: f.parent() is M.scalar_field_algebra()
True
sage: g.parent()
Algebra of scalar fields on the Open subset U of the 2-dimensional
topological manifold M
sage: g.parent() is U.scalar_field_algebra()
True

```

Consequently, scalar fields can be added:

```

sage: s = f + c ; s
Scalar field f+c on the 2-dimensional topological manifold M
sage: s.display()
f+c: M --> R
on U: (x, y) |--> (a*x^2 + a*y^2 + a + 1)/(x^2 + y^2 + 1)
on V: (u, v) |--> ((a + 1)*u^2 + (a + 1)*v^2 + a)/(u^2 + v^2 + 1)

```

and subtracted:

```

sage: s = f - c ; s
Scalar field f-c on the 2-dimensional topological manifold M
sage: s.display()
f-c: M --> R
on U: (x, y) |--> -(a*x^2 + a*y^2 + a - 1)/(x^2 + y^2 + 1)
on V: (u, v) |--> -((a - 1)*u^2 + (a - 1)*v^2 + a)/(u^2 + v^2 + 1)

```

Some tests:

```

sage: f + zer == f
True
sage: f - f == zer
True
sage: f + (-f) == zer
True
sage: (f+c)-f == c
True
sage: (f-c)+c == f
True

```

We may add a number (interpreted as a constant scalar field) to a scalar field:

```

sage: s = f + 1 ; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M --> R
on U: (x, y) |--> (x^2 + y^2 + 2)/(x^2 + y^2 + 1)
on V: (u, v) |--> (2*u^2 + 2*v^2 + 1)/(u^2 + v^2 + 1)
sage: (f+1)-1 == f
True

```

The number can be represented by a symbolic variable:

```

sage: s = a + f ; s
Scalar field on the 2-dimensional topological manifold M
sage: s == c + f
True

```

However if the symbolic variable is a chart coordinate, the addition is performed only on the chart domain:

```

sage: s = f + x; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M --> R
on U: (x, y) |--> (x^3 + x*y^2 + x + 1)/(x^2 + y^2 + 1)
sage: s = f + u; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M --> R
on V: (u, v) |--> (u^3 + (u + 1)*v^2 + u^2 + u)/(u^2 + v^2 + 1)

```

The addition of two scalar fields with different domains is possible if the domain of one of them is a subset of the domain of the other; the domain of the result is then this subset:

```

sage: f.domain()
2-dimensional topological manifold M
sage: g.domain()
Open subset U of the 2-dimensional topological manifold M
sage: s = f + g ; s
Scalar field on the Open subset U of the 2-dimensional topological
manifold M
sage: s.domain()
Open subset U of the 2-dimensional topological manifold M
sage: s.display()
U --> R
(x, y) |--> (x*y^3 + (x^3 + x)*y + 1)/(x^2 + y^2 + 1)
on W: (u, v) |--> (u^6 + 3*u^4*v^2 + 3*u^2*v^4 + v^6 + u*v^3
+ (u^3 + u)*v)/(u^6 + v^6 + (3*u^2 + 1)*v^4 + u^4 + (3*u^4 + 2*u^2)*v^2)

```

The operation actually performed is  $f|_U + g$ :

```

sage: s == f.restrict(U) + g
True

```

In Sage framework, the addition of  $f$  and  $g$  is permitted because there is a *coercion* of the parent of  $f$ , namely  $C^0(M)$ , to the parent of  $g$ , namely  $C^0(U)$  (see [ScalarFieldAlgebra](#)):

```

sage: CM = M.scalar_field_algebra()
sage: CU = U.scalar_field_algebra()

```

```
sage: CU.has_coerce_map_from(CM)
True
```

The coercion map is nothing but the restriction to domain  $U$ :

```
sage: CU.coerce(f) == f.restrict(U)
True
```

Since the algebra  $C^0(M)$  is a vector space over  $\mathbf{R}$ , scalar fields can be multiplied by a number, either an explicit one:

```
sage: s = 2*f ; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M --> R
on U: (x, y) |--> 2/(x^2 + y^2 + 1)
on V: (u, v) |--> 2*(u^2 + v^2)/(u^2 + v^2 + 1)
```

or a symbolic one:

```
sage: s = a*f ; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M --> R
on U: (x, y) |--> a/(x^2 + y^2 + 1)
on V: (u, v) |--> (u^2 + v^2)*a/(u^2 + v^2 + 1)
```

However, if the symbolic variable is a chart coordinate, the multiplication is performed only in the corresponding chart:

```
sage: s = x*f; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M --> R
on U: (x, y) |--> x/(x^2 + y^2 + 1)
sage: s = u*f; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M --> R
on V: (u, v) |--> (u^2 + v^2)*u/(u^2 + v^2 + 1)
```

Some tests:

```
sage: 0*f == 0
True
sage: 0*f == zer
True
sage: 1*f == f
True
sage: (-2)*f == - f - f
True
```

The ring multiplication of the algebras  $C^0(M)$  and  $C^0(U)$  is the pointwise multiplication of functions:

```
sage: s = f*f ; s
Scalar field f*f on the 2-dimensional topological manifold M
sage: s.display()
f*f: M --> R
```

```

on U: (x, y) |--> 1/(x^4 + y^4 + 2*(x^2 + 1)*y^2 + 2*x^2 + 1)
on V: (u, v) |--> (u^4 + 2*u^2*v^2 + v^4)/(u^4 + v^4 + 2*(u^2 + 1)*v^2
+ 2*u^2 + 1)
sage: s = g*h ; s
Scalar field g*h on the Open subset U of the 2-dimensional topological
manifold M
sage: s.display()
g*h: U --> R
(x, y) |--> x*y*H(x, y)
on W: (u, v) |--> u*v*H(u/(u^2 + v^2), v/(u^2 + v^2))/(u^4 + 2*u^2*v^2 + v^4)

```

Thanks to the coercion  $C^0(M) \rightarrow C^0(U)$  mentioned above, it is possible to multiply a scalar field defined on  $M$  by a scalar field defined on  $U$ , the result being a scalar field defined on  $U$ :

```

sage: f.domain(), g.domain()
(2-dimensional topological manifold M,
Open subset U of the 2-dimensional topological manifold M)
sage: s = f*g ; s
Scalar field on the Open subset U of the 2-dimensional topological
manifold M
sage: s.display()
U --> R
(x, y) |--> x*y/(x^2 + y^2 + 1)
on W: (u, v) |--> u*v/(u^4 + v^4 + (2*u^2 + 1)*v^2 + u^2)
sage: s == f.restrict(U)*g
True

```

Scalar fields can be divided (pointwise division):

```

sage: s = f/c ; s
Scalar field f/c on the 2-dimensional topological manifold M
sage: s.display()
f/c: M --> R
on U: (x, y) |--> 1/(a*x^2 + a*y^2 + a)
on V: (u, v) |--> (u^2 + v^2)/(a*u^2 + a*v^2 + a)
sage: s = g/h ; s
Scalar field g/h on the Open subset U of the 2-dimensional topological
manifold M
sage: s.display()
g/h: U --> R
(x, y) |--> x*y/H(x, y)
on W: (u, v) |--> u*v/((u^4 + 2*u^2*v^2 + v^4)*H(u/(u^2 + v^2), v/(u^2 + v^2)))
sage: s = f/g ; s
Scalar field on the Open subset U of the 2-dimensional topological
manifold M
sage: s.display()
U --> R
(x, y) |--> 1/(x*y^3 + (x^3 + x)*y)
on W: (u, v) |--> (u^6 + 3*u^4*v^2 + 3*u^2*v^4 + v^6)/(u*v^3 + (u^3 + u)*v)
sage: s == f.restrict(U)/g
True

```

For scalar fields defined on a single chart domain, we may perform some arithmetics with symbolic expressions involving the chart coordinates:

```

sage: s = g + x^2 - y ; s
Scalar field on the Open subset U of the 2-dimensional topological
manifold M

```

```
sage: s.display()
U --> R
(x, y) |--> x^2 + (x - 1)*y
on W: (u, v) |--> -(v^3 - u^2 + (u^2 - u)*v)/(u^4 + 2*u^2*v^2 + v^4)
```

```
sage: s = g*x ; s
Scalar field on the Open subset U of the 2-dimensional topological
manifold M
sage: s.display()
U --> R
(x, y) |--> x^2*y
on W: (u, v) |--> u^2*v/(u^6 + 3*u^4*v^2 + 3*u^2*v^4 + v^6)
```

```
sage: s = g/x ; s
Scalar field on the Open subset U of the 2-dimensional topological
manifold M
sage: s.display()
U --> R
(x, y) |--> y
on W: (u, v) |--> v/(u^2 + v^2)
sage: s = x/g ; s
Scalar field on the Open subset U of the 2-dimensional topological
manifold M
sage: s.display()
U --> R
(x, y) |--> 1/y
on W: (u, v) |--> (u^2 + v^2)/v
```

The test suite is passed:

```
sage: TestSuite(f).run()
sage: TestSuite(zer).run()
```

**add\_expr** (*coord\_expression*, *chart=None*)

Add some coordinate expression to the scalar field.

The previous expressions with respect to other charts are kept. To clear them, use `set_expr()` instead.

INPUT:

- `coord_expression` – coordinate expression of the scalar field
- `chart` – (default: `None`) chart in which `coord_expression` is defined; if `None`, the default chart of the scalar field's domain is assumed

**Warning:** If the scalar field has already expressions in other charts, it is the user's responsibility to make sure that the expression to be added is consistent with them.

EXAMPLES:

Adding scalar field expressions on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: f = M.scalar_field(x^2 + 2*x*y + 1)
sage: f._express
{Chart (M, (x, y)): x^2 + 2*x*y + 1}
```

```

sage: f.add_expr(3*y)
sage: f._express # the (x,y) expression has been changed:
{Chart (M, (x, y)): 3*y}
sage: c_uv.<u,v> = M.chart()
sage: f.add_expr(cos(u)-sin(v), c_uv)
sage: f._express # random (dict. output); f has now 2 expressions:
{Chart (M, (x, y)): 3*y, Chart (M, (u, v)): cos(u) - sin(v)}

```

**add\_expr\_by\_continuation** (*chart, subdomain*)

Set coordinate expression in a chart by continuation of the coordinate expression in a subchart.

The continuation is performed by demanding that the coordinate expression is identical to that in the restriction of the chart to a given subdomain.

INPUT:

- *chart* – coordinate chart  $(U, (x^i))$  in which the expression of the scalar field is to set
- *subdomain* – open subset  $V \subset U$  in which the expression in terms of the restriction of the coordinate chart  $(U, (x^i))$  to  $V$  is already known or can be evaluated by a change of coordinates.

EXAMPLES:

Scalar field on the sphere  $S^2$ :

```

sage: M = Manifold(2, 'S^2', structure='topological')
sage: U = M.open_subset('U') ; V = M.open_subset('V') # the complement of U
↳ resp. N pole and S pole
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart() # stereographic
↳ coordinates
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....: intersection_name='W', restrictions1= x^2+y^2!=0,
....: restrictions2= u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: W = U.intersection(V) # S^2 minus the two poles
sage: f = M.scalar_field(atan(x^2+y^2), chart=c_xy, name='f')

```

The scalar field has been defined only on the domain covered by the chart  $c_{xy}$ , i.e.  $U$ :

```

sage: f.display()
f: S^2 --> R
on U: (x, y) |--> arctan(x^2 + y^2)

```

We note that on  $W = U \cap V$ , the expression of  $f$  in terms of coordinates  $(u, v)$  can be deduced from that in the coordinates  $(x, y)$  thanks to the transition map between the two charts:

```

sage: f.display(c_uv.restrict(W))
f: S^2 --> R
on W: (u, v) |--> arctan(1/(u^2 + v^2))

```

We use this fact to extend the definition of  $f$  to the open subset  $V$ , covered by the chart  $c_{uv}$ :

```

sage: f.add_expr_by_continuation(c_uv, W)

```

Then,  $f$  is known on the whole sphere:

```

sage: f.display()
f: S^2 --> R

```

```

on U: (x, y) |--> arctan(x^2 + y^2)
on V: (u, v) |--> arctan(1/(u^2 + v^2))

```

**arccos()**

Arc cosine of the scalar field.

OUTPUT:

- the scalar field  $\arccos f$ , where  $f$  is the current scalar field

EXAMPLES:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x*y}, name='f', latex_name=r"\Phi")
sage: g = arccos(f) ; g
Scalar field arccos(f) on the 2-dimensional topological manifold M
sage: latex(g)
\arccos\left(\Phi\right)
sage: g.display()
arccos(f): M --> R
      (x, y) |--> arccos(x*y)

```

The notation `acos` can be used as well:

```

sage: acos(f)
Scalar field arccos(f) on the 2-dimensional topological manifold M
sage: acos(f) == g
True

```

Some tests:

```

sage: cos(g) == f
True
sage: arccos(M.constant_scalar_field(1)) == M.zero_scalar_field()
True
sage: arccos(M.zero_scalar_field()) == M.constant_scalar_field(pi/2)
True

```

**arccosh()**

Inverse hyperbolic cosine of the scalar field.

OUTPUT:

- the scalar field  $\operatorname{arccosh} f$ , where  $f$  is the current scalar field

EXAMPLES:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x*y}, name='f', latex_name=r"\Phi")
sage: g = arccosh(f) ; g
Scalar field arccosh(f) on the 2-dimensional topological manifold M
sage: latex(g)
\mathrm{arccosh}\left(\Phi\right)
sage: g.display()
arccosh(f): M --> R
      (x, y) |--> arccosh(x*y)

```

The notation `acosh` can be used as well:

```

sage: acosh(f)
Scalar field arccosh(f) on the 2-dimensional topological manifold M
sage: acosh(f) == g
True

```

Some tests:

```

sage: cosh(g) == f
True
sage: arccosh(M.constant_scalar_field(1)) == M.zero_scalar_field()
True

```

### **arcsin()**

Arc sine of the scalar field.

OUTPUT:

- the scalar field  $\arcsin f$ , where  $f$  is the current scalar field

EXAMPLES:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x*y}, name='f', latex_name=r"\Phi")
sage: g = arcsin(f) ; g
Scalar field arcsin(f) on the 2-dimensional topological manifold M
sage: latex(g)
\arcsin\left(\Phi\right)
sage: g.display()
arcsin(f): M --> R
      (x, y) |--> arcsin(x*y)

```

The notation `asin` can be used as well:

```

sage: asin(f)
Scalar field arcsin(f) on the 2-dimensional topological manifold M
sage: asin(f) == g
True

```

Some tests:

```

sage: sin(g) == f
True
sage: arcsin(M.zero_scalar_field()) == M.zero_scalar_field()
True
sage: arcsin(M.constant_scalar_field(1)) == M.constant_scalar_field(pi/2)
True

```

### **arcsinh()**

Inverse hyperbolic sine of the scalar field.

OUTPUT:

- the scalar field  $\operatorname{arsinh} f$ , where  $f$  is the current scalar field

EXAMPLES:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x*y}, name='f', latex_name=r"\Phi")

```

```

sage: g = arcsinh(f) ; g
Scalar field arcsinh(f) on the 2-dimensional topological manifold M
sage: latex(g)
\mathrm{arsinh}\left(\Phi\right)
sage: g.display()
arcsinh(f): M --> R
(x, y) |--> arcsinh(x*y)

```

The notation `asinh` can be used as well:

```

sage: asinh(f)
Scalar field arcsinh(f) on the 2-dimensional topological manifold M
sage: asinh(f) == g
True

```

Some tests:

```

sage: sinh(g) == f
True
sage: arcsinh(M.zero_scalar_field()) == M.zero_scalar_field()
True

```

### `arctan()`

Arc tangent of the scalar field.

OUTPUT:

- the scalar field `arctan f`, where `f` is the current scalar field

EXAMPLES:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x*y}, name='f', latex_name=r"\Phi")
sage: g = arctan(f) ; g
Scalar field arctan(f) on the 2-dimensional topological manifold M
sage: latex(g)
\arctan\left(\Phi\right)
sage: g.display()
arctan(f): M --> R
(x, y) |--> arctan(x*y)

```

The notation `atan` can be used as well:

```

sage: atan(f)
Scalar field arctan(f) on the 2-dimensional topological manifold M
sage: atan(f) == g
True

```

Some tests:

```

sage: tan(g) == f
True
sage: arctan(M.zero_scalar_field()) == M.zero_scalar_field()
True
sage: arctan(M.constant_scalar_field(1)) == M.constant_scalar_field(pi/4)
True

```

**arctanh()**

Inverse hyperbolic tangent of the scalar field.

OUTPUT:

- the scalar field  $\operatorname{arctanh} f$ , where  $f$  is the current scalar field

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x*y}, name='f', latex_name=r"\Phi")
sage: g = arctanh(f) ; g
Scalar field arctanh(f) on the 2-dimensional topological manifold M
sage: latex(g)
\, \mathrm{arctanh}\left(\Phi\right)
sage: g.display()
arctanh(f): M --> R
(x, y) |--> arctanh(x*y)
```

The notation `atanh` can be used as well:

```
sage: atanh(f)
Scalar field arctanh(f) on the 2-dimensional topological manifold M
sage: atanh(f) == g
True
```

Some tests:

```
sage: tanh(g) == f
True
sage: arctanh(M.zero_scalar_field()) == M.zero_scalar_field()
True
sage: arctanh(M.constant_scalar_field(1/2)) == M.constant_scalar_field(log(3)/
↪2)
True
```

**common\_charts** (*other*)

Find common charts for the expressions of the scalar field and `other`.

INPUT:

- `other` – a scalar field

OUTPUT:

- list of common charts; if no common chart is found, `None` is returned (instead of an empty list)

EXAMPLES:

Search for common charts on a 2-dimensional manifold with 2 overlapping domains:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: U = M.open_subset('U')
sage: c_xy.<x,y> = U.chart()
sage: V = M.open_subset('V')
sage: c_uv.<u,v> = V.chart()
sage: M.declare_union(U,V) # M is the union of U and V
sage: f = U.scalar_field(x^2)
sage: g = M.scalar_field(x+y)
sage: f.common_charts(g)
[Chart (U, (x, y))]
```

```

sage: g.add_expr(u, c_uv)
sage: f._express
{Chart (U, (x, y)): x^2}
sage: g._express # random (dictionary output)
{Chart (U, (x, y)): x + y, Chart (V, (u, v)): u}
sage: f.common_charts(g)
[Chart (U, (x, y))]

```

Common charts found as subcharts: the subcharts are introduced via a transition map between charts  $c_{xy}$  and  $c_{uv}$  on the intersecting subdomain  $W = U \cap V$ :

```

sage: trans = c_xy.transition_map(c_uv, (x+y, x-y), 'W', x<0, u+v<0)
sage: M.atlas()
[Chart (U, (x, y)), Chart (V, (u, v)), Chart (W, (x, y)),
 Chart (W, (u, v))]
sage: c_xy_W = M.atlas()[2]
sage: c_uv_W = M.atlas()[3]
sage: trans.inverse()
Change of coordinates from Chart (W, (u, v)) to Chart (W, (x, y))
sage: f.common_charts(g)
[Chart (U, (x, y))]
sage: f.expr(c_xy_W)
x^2
sage: f._express # random (dictionary output)
{Chart (U, (x, y)): x^2, Chart (W, (x, y)): x^2}
sage: g._express # random (dictionary output)
{Chart (U, (x, y)): x + y, Chart (V, (u, v)): u}
sage: g.common_charts(f) # c_xy_W is not returned because it is subchart of
↔ 'xy'
[Chart (U, (x, y))]
sage: f.expr(c_uv_W)
1/4*u^2 + 1/2*u*v + 1/4*v^2
sage: f._express # random (dictionary output)
{Chart (U, (x, y)): x^2, Chart (W, (x, y)): x^2,
 Chart (W, (u, v)): 1/4*u^2 + 1/2*u*v + 1/4*v^2}
sage: g._express # random (dictionary output)
{Chart (U, (x, y)): x + y, Chart (V, (u, v)): u}
sage: f.common_charts(g)
[Chart (U, (x, y)), Chart (W, (u, v))]
sage: # the expressions have been updated on the subcharts
sage: g._express # random (dictionary output)
{Chart (U, (x, y)): x + y, Chart (V, (u, v)): u,
 Chart (W, (u, v)): u}

```

Common charts found by computing some coordinate changes:

```

sage: W = U.intersection(V)
sage: f = W.scalar_field(x^2, c_xy_W)
sage: g = W.scalar_field(u+1, c_uv_W)
sage: f._express
{Chart (W, (x, y)): x^2}
sage: g._express
{Chart (W, (u, v)): u + 1}
sage: f.common_charts(g)
[Chart (W, (u, v)), Chart (W, (x, y))]
sage: f._express # random (dictionary output)
{Chart (W, (u, v)): 1/4*u^2 + 1/2*u*v + 1/4*v^2,
 Chart (W, (x, y)): x^2}

```

```
sage: g._express # random (dictionary output)
{Chart (W, (u, v)): u + 1, Chart (W, (x, y)): x + y + 1}
```

**coord\_function** (*chart=None, from\_chart=None*)

Return the function of the coordinates representing the scalar field in a given chart.

INPUT:

- *chart* – (default: None) chart with respect to which the coordinate expression is to be returned; if None, the default chart of the scalar field’s domain will be used
- *from\_chart* – (default: None) chart from which the required expression is computed if it is not known already in the chart *chart*; if None, a chart is picked in the known expressions

OUTPUT:

- instance of *CoordFunction* representing the coordinate function of the scalar field in the given chart

EXAMPLES:

Coordinate function on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: f = M.scalar_field(x*y^2)
sage: f.coord_function()
x*y^2
sage: f.coord_function(c_xy) # equivalent form (since c_xy is the default_
↳chart)
x*y^2
sage: type(f.coord_function())
<class 'sage.manifolds.coord_func_symb.CoordFunctionSymbRing_with_category.
↳element_class'>
```

Expression via a change of coordinates:

```
sage: c_uv.<u,v> = M.chart()
sage: c_uv.transition_map(c_xy, [u+v, u-v])
Change of coordinates from Chart (M, (u, v)) to Chart (M, (x, y))
sage: f._express # at this stage, f is expressed only in terms of (x,y)_
↳coordinates
{Chart (M, (x, y)): x*y^2}
sage: f.coord_function(c_uv) # forces the computation of the expression of f_
↳in terms of (u,v) coordinates
u^3 - u^2*v - u*v^2 + v^3
sage: f.coord_function(c_uv) == (u+v)*(u-v)^2 # check
True
sage: f._express # random (dict. output); f has now 2 coordinate expressions:
{Chart (M, (x, y)): x*y^2, Chart (M, (u, v)): u^3 - u^2*v - u*v^2 + v^3}
```

Usage in a physical context (simple Lorentz transformation - boost in  $x$  direction, with relative velocity  $v$  between  $o_1$  and  $o_2$  frames):

```
sage: M = Manifold(2, 'M', structure='topological')
sage: o1.<t,x> = M.chart()
sage: o2.<T,X> = M.chart()
sage: f = M.scalar_field(x^2 - t^2)
sage: f.coord_function(o1)
-t^2 + x^2
```

```

sage: v = var('v'); gam = 1/sqrt(1-v^2)
sage: o2.transition_map(o1, [gam*(T - v*X), gam*(X - v*T)])
Change of coordinates from Chart (M, (T, X)) to Chart (M, (t, x))
sage: f.coord_function(o2)
-T^2 + X^2

```

**copy()**

Return an exact copy of the scalar field.

## EXAMPLES:

Copy on a 2-dimensional manifold:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: f = M.scalar_field(x*y^2)
sage: g = f.copy()
sage: type(g)
<class 'sage.manifolds.scalarfield_algebra.ScalarFieldAlgebra_with_category.
↳element_class'>
sage: g.expr()
x*y^2
sage: g == f
True
sage: g is f
False

```

**cos()**

Cosine of the scalar field.

## OUTPUT:

- the scalar field  $\cos f$ , where  $f$  is the current scalar field

## EXAMPLES:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x*y}, name='f', latex_name=r"\Phi")
sage: g = cos(f) ; g
Scalar field cos(f) on the 2-dimensional topological manifold M
sage: latex(g)
\cos\left(\Phi\right)
sage: g.display()
cos(f): M --> R
(x, y) |--> cos(x*y)

```

Some tests:

```

sage: cos(M.zero_scalar_field()) == M.constant_scalar_field(1)
True
sage: cos(M.constant_scalar_field(pi/2)) == M.zero_scalar_field()
True

```

**cosh()**

Hyperbolic cosine of the scalar field.

## OUTPUT:

- the scalar field  $\cosh f$ , where  $f$  is the current scalar field

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x*y}, name='f', latex_name=r"\Phi")
sage: g = cosh(f) ; g
Scalar field cosh(f) on the 2-dimensional topological manifold M
sage: latex(g)
\cosh\left(\Phi\right)
sage: g.display()
cosh(f) : M --> R
      (x, y) |--> cosh(x*y)
```

Some test:

```
sage: cosh(M.zero_scalar_field()) == M.constant_scalar_field(1)
True
```

**disp** (*chart=None*)

Display the expression of the scalar field in a given chart.

Without any argument, this function displays the expressions of the scalar field in all the charts defined on the scalar field's domain that are not restrictions of another chart to some subdomain (the “top charts”).

INPUT:

- *chart* – (default: *None*) chart with respect to which the coordinate expression is to be displayed; if *None*, the display is performed in all the top charts in which the coordinate expression is known

The output is either text-formatted (console mode) or LaTeX-formatted (notebook mode).

EXAMPLES:

Various displays:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: f = M.scalar_field(sqrt(x+1), name='f')
sage: f.display()
f: M --> R
      (x, y) |--> sqrt(x + 1)
sage: latex(f.display())
\begin{array}{llcl} f:& M & \longrightarrow & \mathbb{R} \\ \leftarrow y \right) & & \longmapsto & \sqrt{x + 1} \end{array}
sage: g = M.scalar_field(function('G')(x, y), name='g')
sage: g.display()
g: M --> R
      (x, y) |--> G(x, y)
sage: latex(g.display())
\begin{array}{llcl} g:& M & \longrightarrow & \mathbb{R} \\ \leftarrow y \right) & & \longmapsto & G\left(x, y\right) \end{array}
```

A shortcut of `display()` is `disp()`:

```
sage: f.disp()
f: M --> R
      (x, y) |--> sqrt(x + 1)
```

**display** (*chart=None*)

Display the expression of the scalar field in a given chart.

Without any argument, this function displays the expressions of the scalar field in all the charts defined on the scalar field's domain that are not restrictions of another chart to some subdomain (the “top charts”).

INPUT:

- `chart` – (default: `None`) chart with respect to which the coordinate expression is to be displayed; if `None`, the display is performed in all the top charts in which the coordinate expression is known

The output is either text-formatted (console mode) or LaTeX-formatted (notebook mode).

EXAMPLES:

Various displays:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: f = M.scalar_field(sqrt(x+1), name='f')
sage: f.display()
f: M --> R
   (x, y) |--> sqrt(x + 1)
sage: latex(f.display())
\begin{array}{llcl} f:& M & \longrightarrow & \mathbb{R} \\ \leftarrow y \right) & & \longmapsto & \sqrt{x + 1} \end{array}
sage: g = M.scalar_field(function('G')(x, y), name='g')
sage: g.display()
g: M --> R
   (x, y) |--> G(x, y)
sage: latex(g.display())
\begin{array}{llcl} g:& M & \longrightarrow & \mathbb{R} \\ \leftarrow y \right) & & \longmapsto & G \left( x, y \right) \end{array}
```

A shortcut of `display()` is `disp()`:

```
sage: f.disp()
f: M --> R
   (x, y) |--> sqrt(x + 1)
```

**domain()**

Return the open subset on which the scalar field is defined.

OUTPUT:

- instance of class *TopologicalManifold* representing the manifold's open subset on which the scalar field is defined

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: f = M.scalar_field(x+2*y)
sage: f.domain()
2-dimensional topological manifold M
sage: U = M.open_subset('U', coord_def={c_xy: x<0})
sage: g = f.restrict(U)
sage: g.domain()
Open subset U of the 2-dimensional topological manifold M
```

**exp()**

Exponential of the scalar field.

OUTPUT:

- the scalar field  $\exp f$ , where  $f$  is the current scalar field

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x+y}, name='f', latex_name=r"\Phi")
sage: g = exp(f) ; g
Scalar field exp(f) on the 2-dimensional topological manifold M
sage: g.display()
exp(f): M --> R
      (x, y) |--> e^(x + y)
sage: latex(g)
\exp\left(\Phi\right)
```

Automatic simplifications occur:

```
sage: f = M.scalar_field({X: 2*ln(1+x^2)}, name='f')
sage: exp(f).display()
exp(f): M --> R
      (x, y) |--> x^4 + 2*x^2 + 1
```

The inverse function is  $\log()$ :

```
sage: log(exp(f)) == f
True
```

Some tests:

```
sage: exp(M.zero_scalar_field()) == M.constant_scalar_field(1)
True
sage: exp(M.constant_scalar_field(1)) == M.constant_scalar_field(e)
True
```

**expr** (*chart=None, from\_chart=None*)

Return the coordinate expression of the scalar field in a given chart.

INPUT:

- chart* – (default: None) chart with respect to which the coordinate expression is required; if None, the default chart of the scalar field’s domain will be used
- from\_chart* – (default: None) chart from which the required expression is computed if it is not known already in the chart *chart*; if None, a chart is picked in *self.\_express*

OUTPUT:

- symbolic expression representing the coordinate expression of the scalar field in the given chart.

EXAMPLES:

Expression of a scalar field on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: f = M.scalar_field(x*y^2)
sage: f.expr()
x*y^2
sage: f.expr(c_xy) # equivalent form (since c_xy is the default chart)
x*y^2
```

```
sage: type(f.expr())
<type 'sage.symbolic.expression.Expression'>
```

Expression via a change of coordinates:

```
sage: c_uv.<u,v> = M.chart()
sage: c_uv.transition_map(c_xy, [u+v, u-v])
Change of coordinates from Chart (M, (u, v)) to Chart (M, (x, y))
sage: f._express # at this stage, f is expressed only in terms of (x,y)
↳coordinates
{Chart (M, (x, y)): x*y^2}
sage: f.expr(c_uv) # forces the computation of the expression of f in terms
↳of (u,v) coordinates
u^3 - u^2*v - u*v^2 + v^3
sage: bool( f.expr(c_uv) == (u+v)*(u-v)^2 ) # check
True
sage: f._express # random (dict. output); f has now 2 coordinate expressions:
{Chart (M, (x, y)): x*y^2, Chart (M, (u, v)): u^3 - u^2*v - u*v^2 + v^3}
```

**function\_chart** (*chart=None, from\_chart=None*)

Deprecated.

Use `coord_function()` instead.

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: f = M.scalar_field(x*y^2)
sage: fc = f.function_chart()
doctest:...: DeprecationWarning: Use coord_function() instead.
See http://trac.sagemath.org/18640 for details.
sage: fc
x*y^2
```

**log()**

Natural logarithm of the scalar field.

OUTPUT:

- the scalar field  $\ln f$ , where  $f$  is the current scalar field

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x+y}, name='f', latex_name=r"\Phi")
sage: g = log(f) ; g
Scalar field ln(f) on the 2-dimensional topological manifold M
sage: g.display()
ln(f): M --> R
      (x, y) |--> log(x + y)
sage: latex(g)
\ln\left(\Phi\right)
```

The inverse function is `exp()`:

```
sage: exp(log(f)) == f
True
```

**restrict** (*subdomain*)

Restriction of the scalar field to an open subset of its domain of definition.

INPUT:

- *subdomain* – an open subset of the scalar field’s domain

OUTPUT:

- instance of *ScalarField* representing the restriction of the scalar field to *subdomain*

EXAMPLES:

Restriction of a scalar field defined on  $\mathbf{R}^2$  to the unit open disc:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart() # Cartesian coordinates
sage: U = M.open_subset('U', coord_def={X: x^2+y^2 < 1}) # U unit open disc
sage: f = M.scalar_field(cos(x*y), name='f')
sage: f_U = f.restrict(U) ; f_U
Scalar field f on the Open subset U of the 2-dimensional
topological manifold M
sage: f_U.display()
f: U --> R
    (x, y) |--> cos(x*y)
sage: f.parent()
Algebra of scalar fields on the 2-dimensional topological
manifold M
sage: f_U.parent()
Algebra of scalar fields on the Open subset U of the 2-dimensional
topological manifold M
```

The restriction to the whole domain is the identity:

```
sage: f.restrict(M) is f
True
sage: f_U.restrict(U) is f_U
True
```

Restriction of the zero scalar field:

```
sage: M.zero_scalar_field().restrict(U)
Scalar field zero on the Open subset U of the 2-dimensional
topological manifold M
sage: M.zero_scalar_field().restrict(U) is U.zero_scalar_field()
True
```

**set\_expr** (*coord\_expression*, *chart=None*)

Set the coordinate expression of the scalar field.

The expressions with respect to other charts are deleted, in order to avoid any inconsistency. To keep them, use *add\_expr()* instead.

INPUT:

- *coord\_expression* – coordinate expression of the scalar field
- *chart* – (default: None) chart in which *coord\_expression* is defined; if None, the default chart of the scalar field’s domain is assumed

EXAMPLES:

Setting scalar field expressions on a 2-dimensional manifold:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: f = M.scalar_field(x^2 + 2*x*y + 1)
sage: f._express
{Chart (M, (x, y)): x^2 + 2*x*y + 1}
sage: f.set_expr(3*y)
sage: f._express # the (x,y) expression has been changed:
{Chart (M, (x, y)): 3*y}
sage: c_uv.<u,v> = M.chart()
sage: f.set_expr(cos(u)-sin(v), c_uv)
sage: f._express # the (x,y) expression has been lost:
{Chart (M, (u, v)): cos(u) - sin(v)}
sage: f.set_expr(3*y)
sage: f._express # the (u,v) expression has been lost:
{Chart (M, (x, y)): 3*y}

```

**set\_name** (name=None, latex\_name=None)

Set (or change) the text name and LaTeX name of the scalar field.

INPUT:

- name – (string; default: None) name given to the scalar field
- latex\_name – (string; default: None) LaTeX symbol to denote the scalar field; if None while name is provided, the LaTeX symbol is set to name

EXAMPLES:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x+y})
sage: f = M.scalar_field({X: x+y}); f
Scalar field on the 2-dimensional topological manifold M
sage: f.set_name('f'); f
Scalar field f on the 2-dimensional topological manifold M
sage: latex(f)
f
sage: f.set_name('f', latex_name=r'\Phi'); f
Scalar field f on the 2-dimensional topological manifold M
sage: latex(f)
\Phi

```

**sin** ()

Sine of the scalar field.

OUTPUT:

- the scalar field  $\sin f$ , where  $f$  is the current scalar field

EXAMPLES:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x*y}, name='f', latex_name=r"\Phi")
sage: g = sin(f) ; g
Scalar field sin(f) on the 2-dimensional topological manifold M
sage: latex(g)
\sin\left(\Phi\right)
sage: g.display()

```

```
sin(f): M --> R
(x, y) |--> sin(x*y)
```

Some tests:

```
sage: sin(M.zero_scalar_field()) == M.zero_scalar_field()
True
sage: sin(M.constant_scalar_field(pi/2)) == M.constant_scalar_field(1)
True
```

### **sinh()**

Hyperbolic sine of the scalar field.

OUTPUT:

- the scalar field  $\sinh f$ , where  $f$  is the current scalar field

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x*y}, name='f', latex_name=r"\Phi")
sage: g = sinh(f) ; g
Scalar field sinh(f) on the 2-dimensional topological manifold M
sage: latex(g)
\sinh\left(\Phi\right)
sage: g.display()
sinh(f): M --> R
(x, y) |--> sinh(x*y)
```

Some test:

```
sage: sinh(M.zero_scalar_field()) == M.zero_scalar_field()
True
```

### **sqrt()**

Square root of the scalar field.

OUTPUT:

- the scalar field  $\sqrt{f}$ , where  $f$  is the current scalar field

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: 1+x^2+y^2}, name='f',
....:                      latex_name=r"\Phi")
sage: g = sqrt(f) ; g
Scalar field sqrt(f) on the 2-dimensional topological manifold M
sage: latex(g)
\sqrt{\Phi}
sage: g.display()
sqrt(f): M --> R
(x, y) |--> sqrt(x^2 + y^2 + 1)
```

Some tests:

```
sage: g^2 == f
True
```

```
sage: sqrt(M.zero_scalar_field()) == M.zero_scalar_field()
True
```

**tan()**

Tangent of the scalar field.

OUTPUT:

- the scalar field  $\tan f$ , where  $f$  is the current scalar field

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x*y}, name='f', latex_name=r"\Phi")
sage: g = tan(f) ; g
Scalar field tan(f) on the 2-dimensional topological manifold M
sage: latex(g)
\tan\left(\Phi\right)
sage: g.display()
tan(f): M --> R
      (x, y) |--> sin(x*y)/cos(x*y)
```

Some tests:

```
sage: tan(f) == sin(f) / cos(f)
True
sage: tan(M.zero_scalar_field()) == M.zero_scalar_field()
True
sage: tan(M.constant_scalar_field(pi/4)) == M.constant_scalar_field(1)
True
```

**tanh()**

Hyperbolic tangent of the scalar field.

OUTPUT:

- the scalar field  $\tanh f$ , where  $f$  is the current scalar field

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x*y}, name='f', latex_name=r"\Phi")
sage: g = tanh(f) ; g
Scalar field tanh(f) on the 2-dimensional topological manifold M
sage: latex(g)
\tanh\left(\Phi\right)
sage: g.display()
tanh(f): M --> R
      (x, y) |--> sinh(x*y)/cosh(x*y)
```

Some tests:

```
sage: tanh(f) == sinh(f) / cosh(f)
True
sage: tanh(M.zero_scalar_field()) == M.zero_scalar_field()
True
```

## 1.7 Continuous Maps

### 1.7.1 Sets of Morphisms between Topological Manifolds

The class `TopologicalManifoldHomset` implements sets of morphisms between two topological manifolds over the same topological field  $K$ , a morphism being a *continuous map* for the category of topological manifolds.

AUTHORS:

- Ericourgoulhon (2015): initial version
- Travis Scrimshaw (2016): review tweaks

REFERENCES:

- [Lee2011]
- [KN1963]

```
class sage.manifolds.manifold_homset.TopologicalManifoldHomset (domain, codomain,
                                                                name=None, latex_name=None)
```

Bases: `sage.structure.unique_representation.UniqueRepresentation`, `sage.categories.homset.Homset`

Set of continuous maps between two topological manifolds.

Given two topological manifolds  $M$  and  $N$  over a topological field  $K$ , the class `TopologicalManifoldHomset` implements the set  $\text{Hom}(M, N)$  of morphisms (i.e. continuous maps)  $M \rightarrow N$ .

This is a Sage *parent* class, whose *element* class is `ContinuousMap`.

INPUT:

- `domain` – `TopologicalManifold`; the domain topological manifold  $M$  of the morphisms
- `codomain` – `TopologicalManifold`; the codomain topological manifold  $N$  of the morphisms
- `name` – (default: `None`) string; the name of `self`; if `None`,  $\text{Hom}(M, N)$  will be used
- `latex_name` – (default: `None`) string; LaTeX symbol to denote `self`; if `None`,  $\text{Hom}(M, N)$  will be used

EXAMPLES:

Set of continuous maps between a 2-dimensional manifold and a 3-dimensional one:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: N = Manifold(3, 'N', structure='topological')
sage: Y.<u,v,w> = N.chart()
sage: H = Hom(M, N) ; H
Set of Morphisms from 2-dimensional topological manifold M to
3-dimensional topological manifold N in Category of manifolds over
Real Field with 53 bits of precision
sage: type(H)
<class 'sage.manifolds.manifold_homset.TopologicalManifoldHomset_with_category'>
sage: H.category()
Category of homsets of topological spaces
sage: latex(H)
\mathrm{Hom}\left(M, N\right)
sage: H.domain()
```

```
2-dimensional topological manifold M
sage: H.codomain()
3-dimensional topological manifold N
```

An element of H is a continuous map from M to N:

```
sage: H.Element
<class 'sage.manifolds.continuous_map.ContinuousMap'>
sage: f = H.an_element() ; f
Continuous map from the 2-dimensional topological manifold M to the
3-dimensional topological manifold N
sage: f.display()
M --> N
(x, y) |--> (u, v, w) = (0, 0, 0)
```

The test suite is passed:

```
sage: TestSuite(H).run()
```

When the codomain coincides with the domain, the homset is a set of *endomorphisms* in the category of topological manifolds:

```
sage: E = Hom(M, M) ; E
Set of Morphisms from 2-dimensional topological manifold M to
2-dimensional topological manifold M in Category of manifolds over
Real Field with 53 bits of precision
sage: E.category()
Category of endsets of topological spaces
sage: E.is_endomorphism_set()
True
sage: E is End(M)
True
```

In this case, the homset is a monoid for the law of morphism composition:

```
sage: E in Monoids()
True
```

This was of course not the case of  $H = \text{Hom}(M, N)$ :

```
sage: H in Monoids()
False
```

The identity element of the monoid is of course the identity map of M:

```
sage: E.one()
Identity map Id_M of the 2-dimensional topological manifold M
sage: E.one() is M.identity_map()
True
sage: E.one().display()
Id_M: M --> M
(x, y) |--> (x, y)
```

The test suite is passed by E:

```
sage: TestSuite(E).run()
```

This test suite includes more tests than in the case of H, since E has some extra structure (monoid).

**Element**

alias of `ContinuousMap`

**one()**

Return the identity element of `self` considered as a monoid (case of a set of endomorphisms).

This applies only when the codomain of the homset is equal to its domain, i.e. when the homset is of the type  $\text{Hom}(M, M)$ . Indeed,  $\text{Hom}(M, M)$  equipped with the law of morphisms composition is a monoid, whose identity element is nothing but the identity map of  $M$ .

OUTPUT:

- the identity map of  $M$ , as an instance of `ContinuousMap`

EXAMPLES:

The identity map of a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: H = Hom(M, M) ; H
Set of Morphisms from 2-dimensional topological manifold M to
2-dimensional topological manifold M in Category of manifolds over
Real Field with 53 bits of precision
sage: H in Monoids()
True
sage: H.one()
Identity map Id_M of the 2-dimensional topological manifold M
sage: H.one().parent() is H
True
sage: H.one().display()
Id_M: M --> M
      (x, y) |--> (x, y)
```

The identity map is cached:

```
sage: H.one() is H.one()
True
```

If the homset is not a set of endomorphisms, the identity element is meaningless:

```
sage: N = Manifold(3, 'N', structure='topological')
sage: Y.<u,v,w> = N.chart()
sage: Hom(M, N).one()
Traceback (most recent call last):
...
TypeError: Set of Morphisms
from 2-dimensional topological manifold M
to 3-dimensional topological manifold N
in Category of manifolds over Real Field with 53 bits of precision
is not a monoid
```

## 1.7.2 Continuous Maps Between Topological Manifolds

`ContinuousMap` implements continuous maps from a topological manifold  $M$  to some topological manifold  $N$  over the same topological field  $K$  as  $M$ .

AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2013-2015): initial version
- Travis Scrimshaw (2016): review tweaks

## REFERENCES:

- Chap. 1 of [KN1963]
- [Lee2011]

```
class sage.manifolds.continuous_map.ContinuousMap(parent, coord_functions=None,
                                                  name=None, latex_name=None,
                                                  is_isomorphism=False,
                                                  is_identity=False)
```

Bases: `sage.categories.morphism.Morphism`

Continuous map between two topological manifolds.

This class implements continuous maps of the type

$$\Phi : M \longrightarrow N,$$

where  $M$  and  $N$  are topological manifolds over the same topological field  $K$ .

Continuous maps are the morphisms of the category of topological manifolds. The set of all continuous maps from  $M$  to  $N$  is therefore the homset between  $M$  and  $N$ , which is denoted by  $\text{Hom}(M, N)$ .

The class `ContinuousMap` is a Sage *element* class, whose *parent* class is `TopologicalManifoldHomset`.

## INPUT:

- `parent` – homset  $\text{Hom}(M, N)$  to which the continuous map belongs
- `coord_functions` – a dictionary of the coordinate expressions (as lists or tuples of the coordinates of the image expressed in terms of the coordinates of the considered point) with the pairs of charts (`chart1`, `chart2`) as keys (`chart1` being a chart on  $M$  and `chart2` a chart on  $N$ )
- `name` – (default: `None`) name given to `self`
- `latex_name` – (default: `None`) LaTeX symbol to denote the continuous map; if `None`, the LaTeX symbol is set to `name`
- `is_isomorphism` – (default: `False`) determines whether the constructed object is a isomorphism (i.e. a homeomorphism); if set to `True`, then the manifolds  $M$  and  $N$  must have the same dimension
- `is_identity` – (default: `False`) determines whether the constructed object is the identity map; if set to `True`, then  $N$  must be  $M$  and the entry `coord_functions` is not used

---

**Note:** If the information passed by means of the argument `coord_functions` is not sufficient to fully specify the continuous map, further coordinate expressions, in other charts, can be subsequently added by means of the method `add_expr()`.

---

## EXAMPLES:

The standard embedding of the sphere  $S^2$  into  $\mathbf{R}^3$ :

```
sage: M = Manifold(2, 'S^2', structure='topological') # the 2-dimensional sphere
      ↪ S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
```

```

sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                                     intersection_name='W',
....:                                     restrictions1=x^2+y^2!=0,
....:                                     restrictions2=u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: N = Manifold(3, 'R^3', latex_name=r'\RR^3', structure='topological') # R^3
sage: c_cart.<X,Y,Z> = N.chart() # Cartesian coordinates on R^3
sage: Phi = M.continuous_map(N,
....:   {(c_xy, c_cart): [2*x/(1+x^2+y^2), 2*y/(1+x^2+y^2), (x^2+y^2-1)/(1+x^2+y^
↪2)],
....:   (c_uv, c_cart): [2*u/(1+u^2+v^2), 2*v/(1+u^2+v^2), (1-u^2-v^2)/(1+u^2+v^
↪2)]}),
....:   name='Phi', latex_name=r'\Phi')
sage: Phi
Continuous map Phi from the 2-dimensional topological manifold S^2
to the 3-dimensional topological manifold R^3
sage: Phi.parent()
Set of Morphisms from 2-dimensional topological manifold S^2
to 3-dimensional topological manifold R^3
in Category of manifolds over Real Field with 53 bits of precision
sage: Phi.parent() is Hom(M, N)
True
sage: type(Phi)
<class 'sage.manifolds.manifold_homset.TopologicalManifoldHomset_with_category.
↪element_class'>
sage: Phi.display()
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2 + y^
↪2 - 1)/(x^2 + y^2 + 1))
on V: (u, v) |--> (X, Y, Z) = (2*u/(u^2 + v^2 + 1), 2*v/(u^2 + v^2 + 1), -(u^2 +
↪v^2 - 1)/(u^2 + v^2 + 1))

```

It is possible to create the map using `continuous_map()` with only in a single pair of charts. The argument `coord_functions` is then a mere list of coordinate expressions (and not a dictionary) and the arguments `chart1` and `chart2` have to be provided if the charts differ from the default ones on the domain and/or codomain:

```

sage: Phi1 = M.continuous_map(N, [2*x/(1+x^2+y^2), 2*y/(1+x^2+y^2), (x^2+y^2-1)/
↪(1+x^2+y^2)],
....:                               chart1=c_xy, chart2=c_cart,
....:                               name='Phi', latex_name=r'\Phi')

```

Since `c_xy` and `c_cart` are the default charts on respectively `M` and `N`, they can be omitted, so that the above declaration is equivalent to:

```

sage: Phi1 = M.continuous_map(N, [2*x/(1+x^2+y^2), 2*y/(1+x^2+y^2), (x^2+y^2-1)/
↪(1+x^2+y^2)],
....:                               name='Phi', latex_name=r'\Phi')

```

With such a declaration, the continuous map `Phi1` is only partially defined on the manifold  $S^2$  as it is known in only one chart:

```

sage: Phi1.display()
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2 + y^
↪2 - 1)/(x^2 + y^2 + 1))

```

The definition can be completed by using `add_expr()`:

```
sage: Phi1.add_expr(c_uv, c_cart, [2*u/(1+u^2+v^2), 2*v/(1+u^2+v^2), (1-u^2-v^2)/
↪(1+u^2+v^2)])
sage: Phi1.display()
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2 + y^
↪2 - 1)/(x^2 + y^2 + 1))
on V: (u, v) |--> (X, Y, Z) = (2*u/(u^2 + v^2 + 1), 2*v/(u^2 + v^2 + 1), -(u^2 +
↪v^2 - 1)/(u^2 + v^2 + 1))
```

At this stage, `Phi1` and `Phi` are fully equivalent:

```
sage: Phi1 == Phi
True
```

The map acts on points:

```
sage: np = M.point((0,0), chart=c_uv) # the North pole
sage: Phi(np)
Point on the 3-dimensional topological manifold R^3
sage: Phi(np).coord() # Cartesian coordinates
(0, 0, 1)
sage: sp = M.point((0,0), chart=c_xy) # the South pole
sage: Phi(sp).coord() # Cartesian coordinates
(0, 0, -1)
```

The test suite is passed:

```
sage: TestSuite(Phi).run()
sage: TestSuite(Phi1).run()
```

Continuous maps can be composed by means of the operator `*`. Let us introduce the map  $\mathbf{R}^3 \rightarrow \mathbf{R}^2$  corresponding to the projection from the point  $(X, Y, Z) = (0, 0, 1)$  onto the equatorial plane  $Z = 0$ :

```
sage: P = Manifold(2, 'R^2', latex_name=r'\mathbb{R}^2', structure='topological') # R^2
↪(equatorial plane)
sage: cP.<xP, yP> = P.chart()
sage: Psi = N.continuous_map(P, (X/(1-Z), Y/(1-Z)), name='Psi',
....:                          latex_name=r'\Psi')
sage: Psi
Continuous map Psi from the 3-dimensional topological manifold R^3
to the 2-dimensional topological manifold R^2
sage: Psi.display()
Psi: R^3 --> R^2
(X, Y, Z) |--> (xP, yP) = (-X/(Z - 1), -Y/(Z - 1))
```

Then we compose `Psi` with `Phi`, thereby getting a map  $S^2 \rightarrow \mathbf{R}^2$ :

```
sage: ster = Psi * Phi ; ster
Continuous map from the 2-dimensional topological manifold S^2
to the 2-dimensional topological manifold R^2
```

Let us test on the South pole (`sp`) that `ster` is indeed the composite of `Psi` and `Phi`:

```
sage: ster(sp) == Psi(Phi(sp))
True
```

Actually `ster` is the stereographic projection from the North pole, as its coordinate expression reveals:

```
sage: ster.display()
S^2 --> R^2
on U: (x, y) |--> (xP, yP) = (x, y)
on V: (u, v) |--> (xP, yP) = (u/(u^2 + v^2), v/(u^2 + v^2))
```

If the codomain of a continuous map is 1-dimensional, the map can be defined by a single symbolic expression for each pair of charts and not by a list/tuple with a single element:

```
sage: N = Manifold(1, 'N', structure='topological')
sage: c_N = N.chart('X')
sage: Phi = M.continuous_map(N, {(c_xy, c_N): x^2+y^2,
.....:                          (c_uv, c_N): 1/(u^2+v^2)})

sage: Psi = M.continuous_map(N, {(c_xy, c_N): [x^2+y^2],
.....:                          (c_uv, c_N): [1/(u^2+v^2)]})
sage: Phi == Psi
True
```

Next we construct an example of continuous map  $\mathbf{R} \rightarrow \mathbf{R}^2$ :

```
sage: R = Manifold(1, 'R', structure='topological') # field R
sage: T.<t> = R.chart() # canonical chart on R
sage: R2 = Manifold(2, 'R^2', structure='topological') # R^2
sage: c_xy.<x,y> = R2.chart() # Cartesian coordinates on R^2
sage: Phi = R.continuous_map(R2, [cos(t), sin(t)], name='Phi'); Phi
Continuous map Phi from the 1-dimensional topological manifold R
to the 2-dimensional topological manifold R^2
sage: Phi.parent()
Set of Morphisms from 1-dimensional topological manifold R
to 2-dimensional topological manifold R^2
in Category of manifolds over Real Field with 53 bits of precision
sage: Phi.parent() is Hom(R, R2)
True
sage: Phi.display()
Phi: R --> R^2
t |--> (x, y) = (cos(t), sin(t))
```

An example of homeomorphism between the unit open disk and the Euclidean plane  $\mathbf{R}^2$ :

```
sage: D = R2.open_subset('D', coord_def={c_xy: x^2+y^2<1}) # the open unit disk
sage: Phi = D.homeomorphism(R2, [x/sqrt(1-x^2-y^2), y/sqrt(1-x^2-y^2)],
.....:                          name='Phi', latex_name=r'\Phi')
sage: Phi
Homeomorphism Phi from the Open subset D of the 2-dimensional
topological manifold R^2 to the 2-dimensional topological manifold R^2
sage: Phi.parent()
Set of Morphisms from Open subset D of the 2-dimensional topological
manifold R^2 to 2-dimensional topological manifold R^2 in Join of
Category of subobjects of sets and Category of manifolds over Real
Field with 53 bits of precision
sage: Phi.parent() is Hom(D, R2)
True
sage: Phi.display()
```

```
Phi: D --> R^2
(x, y) |--> (x, y) = (x/sqrt(-x^2 - y^2 + 1), y/sqrt(-x^2 - y^2 + 1))
```

The image of a point:

```
sage: p = D.point((1/2,0))
sage: q = Phi(p) ; q
Point on the 2-dimensional topological manifold R^2
sage: q.coord()
(1/3*sqrt(3), 0)
```

The inverse homeomorphism is computed by `inverse()`:

```
sage: Phi.inverse()
Homeomorphism Phi^(-1) from the 2-dimensional topological manifold R^2
to the Open subset D of the 2-dimensional topological manifold R^2
sage: Phi.inverse().display()
Phi^(-1): R^2 --> D
(x, y) |--> (x, y) = (x/sqrt(x^2 + y^2 + 1), y/sqrt(x^2 + y^2 + 1))
```

Equivalently, one may use the notations  $\wedge(-1)$  or  $\sim$  to get the inverse:

```
sage: Phi^(-1) is Phi.inverse()
True
sage: ~Phi is Phi.inverse()
True
```

Check that  $\sim$ Phi is indeed the inverse of Phi:

```
sage: (~Phi)(q) == p
True
sage: Phi * ~Phi == R2.identity_map()
True
sage: ~Phi * Phi == D.identity_map()
True
```

The coordinate expression of the inverse homeomorphism:

```
sage: (~Phi).display()
Phi^(-1): R^2 --> D
(x, y) |--> (x, y) = (x/sqrt(x^2 + y^2 + 1), y/sqrt(x^2 + y^2 + 1))
```

A special case of homeomorphism: the identity map of the open unit disk:

```
sage: id = D.identity_map() ; id
Identity map Id_D of the Open subset D of the 2-dimensional topological
manifold R^2
sage: latex(id)
\mathrm{Id}_{D}
sage: id.parent()
Set of Morphisms from Open subset D of the 2-dimensional topological
manifold R^2 to Open subset D of the 2-dimensional topological
manifold R^2 in Join of Category of subobjects of sets and Category of
manifolds over Real Field with 53 bits of precision
sage: id.parent() is Hom(D, D)
True
sage: id is Hom(D,D).one() # the identity element of the monoid Hom(D,D)
True
```

The identity map acting on a point:

```
sage: id(p)
Point on the 2-dimensional topological manifold R^2
sage: id(p) == p
True
sage: id(p) is p
True
```

The coordinate expression of the identity map:

```
sage: id.display()
Id_D: D --> D
      (x, y) |--> (x, y)
```

The identity map is its own inverse:

```
sage: id^(-1) is id
True
sage: ~id is id
True
```

**add\_expr** (*chart1*, *chart2*, *coord\_functions*)

Set a new coordinate representation of *self*.

The previous expressions with respect to other charts are kept. To clear them, use *set\_expr()* instead.

INPUT:

- *chart1* – chart for the coordinates on the map’s domain
- *chart2* – chart for the coordinates on the map’s codomain
- *coord\_functions* – the coordinate symbolic expression of the map in the above charts: list (or tuple) of the coordinates of the image expressed in terms of the coordinates of the considered point; if the dimension of the arrival manifold is 1, a single coordinate expression can be passed instead of a tuple with a single element

**Warning:** If the map has already expressions in other charts, it is the user’s responsibility to make sure that the expression to be added is consistent with them.

EXAMPLES:

Polar representation of a planar rotation initially defined in Cartesian coordinates:

```
sage: M = Manifold(2, 'R^2', latex_name=r'\RR^2', structure='topological') #
↳ the Euclidean plane R^2
sage: c_xy.<x,y> = M.chart() # Cartesian coordinate on R^2
sage: U = M.open_subset('U', coord_def={c_xy: (y!=0, x<0)}) # the complement
↳ of the segment y=0 and x>0
sage: c_cart = c_xy.restrict(U) # Cartesian coordinates on U
sage: c_spher.<r,ph> = U.chart(r'r:(0,+oo) ph:(0,2*pi):\phi') # spherical
↳ coordinates on U
```

We construct the links between spherical coordinates and Cartesian ones:

```

sage: ch_cart_spher = c_cart.transition_map(c_spher, [sqrt(x*x+y*y), atan2(y,
↪x)])
sage: ch_cart_spher.set_inverse(r*cos(ph), r*sin(ph), verbose=True)
Check of the inverse coordinate transformation:
    x == x
    y == y
    r == r
    ph == arctan2(r*sin(ph), r*cos(ph))
sage: rot = U.continuous_map(U, ((x - sqrt(3)*y)/2, (sqrt(3)*x + y)/2),
....:                          name='R')
sage: rot.display(c_cart, c_cart)
R: U --> U
    (x, y) |--> (-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)

```

If we calculate the expression in terms of spherical coordinates, via the method `display()`, we notice some difficulties in `arctan2` simplifications:

```

sage: rot.display(c_spher, c_spher)
R: U --> U
    (r, ph) |--> (r, arctan2(1/2*(sqrt(3)*cos(ph) + sin(ph))*r, -1/
↪2*(sqrt(3)*sin(ph) - cos(ph))*r))

```

Therefore, we use the method `add_expr()` to set the spherical-coordinate expression by hand:

```

sage: rot.add_expr(c_spher, c_spher, (r, ph+pi/3))
sage: rot.display(c_spher, c_spher)
R: U --> U
    (r, ph) |--> (r, 1/3*pi + ph)

```

The call to `add_expr()` has not deleted the expression in terms of Cartesian coordinates, as we can check by printing the internal dictionary `_coord_expression`, which stores the various internal representations of the continuous map:

```

sage: rot._coord_expression # random (dictionary output)
{(Chart (U, (x, y)), Chart (U, (x, y))):
  Coordinate functions (-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)
  on the Chart (U, (x, y)),
 (Chart (U, (r, ph)), Chart (U, (r, ph))):
  Coordinate functions (r, 1/3*pi + ph) on the Chart (U, (r, ph))}

```

If, on the contrary, we use `set_expr()`, the expression in Cartesian coordinates is lost:

```

sage: rot.set_expr(c_spher, c_spher, (r, ph+pi/3))
sage: rot._coord_expression
{(Chart (U, (r, ph)), Chart (U, (r, ph))):
  Coordinate functions (r, 1/3*pi + ph) on the Chart (U, (r, ph))}

```

It is recovered (thanks to the known change of coordinates) by a call to `display()`:

```

sage: rot.display(c_cart, c_cart)
R: U --> U
    (x, y) |--> (-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)

sage: rot._coord_expression # random (dictionary output)
{(Chart (U, (x, y)), Chart (U, (x, y))):
  Coordinate functions (-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)
  on the Chart (U, (x, y)),

```

```
(Chart (U, (r, ph)), Chart (U, (r, ph))):
Coordinate functions (r, 1/3*pi + ph) on the Chart (U, (r, ph))}
```

The rotation can be applied to a point by means of either coordinate system:

```
sage: p = M.point((1,2)) # p defined by its Cartesian coord.
sage: q = rot(p) # q is computed by means of Cartesian coord.
sage: p1 = M.point((sqrt(5), arctan(2)), chart=c_spher) # p1 is defined only
↳in terms of c_spher
sage: q1 = rot(p1) # computation by means of spherical coordinates
sage: q1 == q
True
```

**add\_expression** (*chart1, chart2, coord\_functions*)

Set a new coordinate representation of *self*.

The previous expressions with respect to other charts are kept. To clear them, use *set\_expr()* instead.

INPUT:

- *chart1* – chart for the coordinates on the map’s domain
- *chart2* – chart for the coordinates on the map’s codomain
- *coord\_functions* – the coordinate symbolic expression of the map in the above charts: list (or tuple) of the coordinates of the image expressed in terms of the coordinates of the considered point; if the dimension of the arrival manifold is 1, a single coordinate expression can be passed instead of a tuple with a single element

**Warning:** If the map has already expressions in other charts, it is the user’s responsibility to make sure that the expression to be added is consistent with them.

EXAMPLES:

Polar representation of a planar rotation initially defined in Cartesian coordinates:

```
sage: M = Manifold(2, 'R^2', latex_name=r'\RR^2', structure='topological') #
↳the Euclidean plane R^2
sage: c_xy.<x,y> = M.chart() # Cartesian coordinate on R^2
sage: U = M.open_subset('U', coord_def={c_xy: (y!=0, x<0)}) # the complement
↳of the segment y=0 and x>0
sage: c_cart = c_xy.restrict(U) # Cartesian coordinates on U
sage: c_spher.<r,ph> = U.chart(r'r:(0,+oo) ph:(0,2*pi):\phi') # spherical
↳coordinates on U
```

We construct the links between spherical coordinates and Cartesian ones:

```
sage: ch_cart_spher = c_cart.transition_map(c_spher, [sqrt(x*x+y*y), atan2(y,
↳x)])
sage: ch_cart_spher.set_inverse(r*cos(ph), r*sin(ph), verbose=True)
Check of the inverse coordinate transformation:
x == x
y == y
r == r
ph == arctan2(r*sin(ph), r*cos(ph))
sage: rot = U.continuous_map(U, ((x - sqrt(3)*y)/2, (sqrt(3)*x + y)/2),
....:                          name='R')
sage: rot.display(c_cart, c_cart)
```

```
R: U --> U
(x, y) |--> (-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)
```

If we calculate the expression in terms of spherical coordinates, via the method `display()`, we notice some difficulties in `arctan2` simplifications:

```
sage: rot.display(c_spher, c_spher)
R: U --> U
(r, ph) |--> (r, arctan2(1/2*(sqrt(3)*cos(ph) + sin(ph))*r, -1/
↪2*(sqrt(3)*sin(ph) - cos(ph))*r))
```

Therefore, we use the method `add_expr()` to set the spherical-coordinate expression by hand:

```
sage: rot.add_expr(c_spher, c_spher, (r, ph+pi/3))
sage: rot.display(c_spher, c_spher)
R: U --> U
(r, ph) |--> (r, 1/3*pi + ph)
```

The call to `add_expr()` has not deleted the expression in terms of Cartesian coordinates, as we can check by printing the internal dictionary `_coord_expression`, which stores the various internal representations of the continuous map:

```
sage: rot._coord_expression # random (dictionary output)
{(Chart (U, (x, y)), Chart (U, (x, y))):
Coordinate functions (-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)
on the Chart (U, (x, y)),
(Chart (U, (r, ph)), Chart (U, (r, ph))):
Coordinate functions (r, 1/3*pi + ph) on the Chart (U, (r, ph))}
```

If, on the contrary, we use `set_expr()`, the expression in Cartesian coordinates is lost:

```
sage: rot.set_expr(c_spher, c_spher, (r, ph+pi/3))
sage: rot._coord_expression
{(Chart (U, (r, ph)), Chart (U, (r, ph))):
Coordinate functions (r, 1/3*pi + ph) on the Chart (U, (r, ph))}
```

It is recovered (thanks to the known change of coordinates) by a call to `display()`:

```
sage: rot.display(c_cart, c_cart)
R: U --> U
(x, y) |--> (-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)

sage: rot._coord_expression # random (dictionary output)
{(Chart (U, (x, y)), Chart (U, (x, y))):
Coordinate functions (-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)
on the Chart (U, (x, y)),
(Chart (U, (r, ph)), Chart (U, (r, ph))):
Coordinate functions (r, 1/3*pi + ph) on the Chart (U, (r, ph))}
```

The rotation can be applied to a point by means of either coordinate system:

```
sage: p = M.point((1,2)) # p defined by its Cartesian coord.
sage: q = rot(p) # q is computed by means of Cartesian coord.
sage: p1 = M.point((sqrt(5), arctan(2)), chart=c_spher) # p1 is defined only_
↪in terms of c_spher
sage: q1 = rot(p1) # computation by means of spherical coordinates
sage: q1 == q
True
```

**coord\_functions** (*chart1=None, chart2=None*)

Return the functions of the coordinates representing *self* in a given pair of charts.

If these functions are not already known, they are computed from known ones by means of change-of-chart formulas.

INPUT:

- *chart1* – (default: None) chart on the domain of *self*; if None, the domain’s default chart is assumed
- *chart2* – (default: None) chart on the codomain of *self*; if None, the codomain’s default chart is assumed

OUTPUT:

- a *MultiCoordFunction* representing the continuous map in the above two charts

EXAMPLES:

Continuous map from a 2-dimensional manifold to a 3-dimensional one:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: N = Manifold(3, 'N', structure='topological')
sage: c_uv.<u,v> = M.chart()
sage: c_xyz.<x,y,z> = N.chart()
sage: Phi = M.continuous_map(N, (u*v, u/v, u+v), name='Phi',
....:                          latex_name=r'\Phi')
sage: Phi.display()
Phi: M --> N
      (u, v) |--> (x, y, z) = (u*v, u/v, u + v)
sage: Phi.coord_functions(c_uv, c_xyz)
Coordinate functions (u*v, u/v, u + v) on the Chart (M, (u, v))
sage: Phi.coord_functions() # equivalent to above since 'uv' and 'xyz' are
↪ default charts
Coordinate functions (u*v, u/v, u + v) on the Chart (M, (u, v))
sage: type(Phi.coord_functions())
<class 'sage.manifolds.coord_func.MultiCoordFunction'>
```

Coordinate representation in other charts:

```
sage: c_UV.<U,V> = M.chart() # new chart on M
sage: ch_uv_UV = c_uv.transition_map(c_UV, [u-v, u+v])
sage: ch_uv_UV.inverse() (U,V)
(1/2*U + 1/2*V, -1/2*U + 1/2*V)
sage: c_XYZ.<X,Y,Z> = N.chart() # new chart on N
sage: ch_xyz_XYZ = c_xyz.transition_map(c_XYZ,
....:                                  [2*x-3*y+z, y+z-x, -x+2*y-z])
sage: ch_xyz_XYZ.inverse() (X,Y,Z)
(3*X + Y + 4*Z, 2*X + Y + 3*Z, X + Y + Z)
sage: Phi.coord_functions(c_UV, c_xyz)
Coordinate functions (-1/4*U^2 + 1/4*V^2, -(U + V)/(U - V), V) on
the Chart (M, (U, V))
sage: Phi.coord_functions(c_uv, c_XYZ)
Coordinate functions (((2*u + 1)*v^2 + u*v - 3*u)/v,
-(u - 1)*v^2 - u*v - u)/v, -((u + 1)*v^2 + u*v - 2*u)/v) on the
Chart (M, (u, v))
sage: Phi.coord_functions(c_UV, c_XYZ)
Coordinate functions
```

```
(-1/2*(U^3 - (U - 2)*V^2 + V^3 - (U^2 + 2*U + 6)*V - 6*U)/(U - V),
 1/4*(U^3 - (U + 4)*V^2 + V^3 - (U^2 - 4*U + 4)*V - 4*U)/(U - V),
 1/4*(U^3 - (U - 4)*V^2 + V^3 - (U^2 + 4*U + 8)*V - 8*U)/(U - V))
on the Chart (M, (U, V))
```

Coordinate representation with respect to a subchart in the domain:

```
sage: A = M.open_subset('A', coord_def={c_uv: u>0})
sage: Phi.coord_functions(c_uv.restrict(A), c_xyz)
Coordinate functions (u*v, u/v, u + v) on the Chart (A, (u, v))
```

Coordinate representation with respect to a superchart in the codomain:

```
sage: B = N.open_subset('B', coord_def={c_xyz: x<0})
sage: c_xyz_B = c_xyz.restrict(B)
sage: Phi1 = M.continuous_map(B, {(c_uv, c_xyz_B): (u*v, u/v, u+v)})
sage: Phi1.coord_functions(c_uv, c_xyz_B) # definition charts
Coordinate functions (u*v, u/v, u + v) on the Chart (M, (u, v))
sage: Phi1.coord_functions(c_uv, c_xyz) # c_xyz = superchart of c_xyz_B
Coordinate functions (u*v, u/v, u + v) on the Chart (M, (u, v))
```

Coordinate representation with respect to a pair (subchart, superchart):

```
sage: Phi1.coord_functions(c_uv.restrict(A), c_xyz)
Coordinate functions (u*v, u/v, u + v) on the Chart (A, (u, v))
```

**disp** (*chart1=None, chart2=None*)

Display the expression of `self` in one or more pair of charts.

If the expression is not known already, it is computed from some expression in other charts by means of change-of-coordinate formulas.

INPUT:

- `chart1` – (default: `None`) chart on the domain of `self`; if `None`, the display is performed on all the charts on the domain in which the map is known or computable via some change of coordinates
- `chart2` – (default: `None`) chart on the codomain of `self`; if `None`, the display is performed on all the charts on the codomain in which the map is known or computable via some change of coordinates

The output is either text-formatted (console mode) or LaTeX-formatted (notebook mode).

EXAMPLES:

Standard embedding of the sphere  $S^2$  in  $\mathbf{R}^3$ :

```
sage: M = Manifold(2, 'S^2', structure='topological') # the 2-dimensional
↳ sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: N = Manifold(3, 'R^3', latex_name=r'\RR^3', structure='topological') #
↳ R^3
sage: c_cart.<X,Y,Z> = N.chart() # Cartesian coordinates on R^3
sage: Phi = M.continuous_map(N,
....:   {(c_xy, c_cart): [2*x/(1+x^2+y^2), 2*y/(1+x^2+y^2), (x^2+y^2-1)/(1+x^
↳ 2+y^2)],
....:   (c_uv, c_cart): [2*u/(1+u^2+v^2), 2*v/(1+u^2+v^2), (1-u^2-v^2)/(1+u^
↳ 2+v^2)]},
```

```

....: name='Phi', latex_name=r'\Phi')
sage: Phi.display(c_xy, c_cart)
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2_
↪+ y^2 - 1)/(x^2 + y^2 + 1))
sage: Phi.display(c_uv, c_cart)
Phi: S^2 --> R^3
on V: (u, v) |--> (X, Y, Z) = (2*u/(u^2 + v^2 + 1), 2*v/(u^2 + v^2 + 1), -(u^
↪2 + v^2 - 1)/(u^2 + v^2 + 1))

```

The LaTeX output:

```

sage: latex(Phi.display(c_xy, c_cart))
\begin{array}{llcl} \Phi:& S^2 & \longrightarrow & \mathbb{R}^3 \\ \\ \mbox{on } U & : & \left(x, y\right) & \longmapsto \\ & & \left(X, Y, Z\right) = & \left(\frac{2x}{x^2 + y^2 + 1}, \right. \\ & & \left. \frac{2y}{x^2 + y^2 + 1}, \right. \\ & & \left. \frac{x^2 + y^2 - 1}{x^2 + y^2 + 1}\right) \\ \end{array}

```

If the argument `chart2` is not specified, the display is performed on all the charts on the codomain in which the map is known or computable via some change of coordinates (here only one chart: `c_cart`):

```

sage: Phi.display(c_xy)
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2_
↪+ y^2 - 1)/(x^2 + y^2 + 1))

```

Similarly, if the argument `chart1` is omitted, the display is performed on all the charts on the domain of `Phi` in which the map is known or computable via some change of coordinates:

```

sage: Phi.display(chart2=c_cart)
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2_
↪+ y^2 - 1)/(x^2 + y^2 + 1))
on V: (u, v) |--> (X, Y, Z) = (2*u/(u^2 + v^2 + 1), 2*v/(u^2 + v^2 + 1), -(u^
↪2 + v^2 - 1)/(u^2 + v^2 + 1))

```

If neither `chart1` nor `chart2` is specified, the display is performed on all the pair of charts in which `Phi` is known or computable via some change of coordinates:

```

sage: Phi.display()
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2_
↪+ y^2 - 1)/(x^2 + y^2 + 1))
on V: (u, v) |--> (X, Y, Z) = (2*u/(u^2 + v^2 + 1), 2*v/(u^2 + v^2 + 1), -(u^
↪2 + v^2 - 1)/(u^2 + v^2 + 1))

```

If a chart covers entirely the map's domain, the mention "on ..." is omitted:

```

sage: Phi.restrict(U).display()
Phi: U --> R^3
(x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2 +
↪y^2 - 1)/(x^2 + y^2 + 1))

```

A shortcut of `display()` is `disp()`:

```

sage: Phi.display()
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2_
↪ + y^2 - 1)/(x^2 + y^2 + 1))
on V: (u, v) |--> (X, Y, Z) = (2*u/(u^2 + v^2 + 1), 2*v/(u^2 + v^2 + 1), -(u^
↪ 2 + v^2 - 1)/(u^2 + v^2 + 1))

```

**display** (*chart1=None, chart2=None*)

Display the expression of `self` in one or more pair of charts.

If the expression is not known already, it is computed from some expression in other charts by means of change-of-coordinate formulas.

INPUT:

- `chart1` – (default: `None`) chart on the domain of `self`; if `None`, the display is performed on all the charts on the domain in which the map is known or computable via some change of coordinates
- `chart2` – (default: `None`) chart on the codomain of `self`; if `None`, the display is performed on all the charts on the codomain in which the map is known or computable via some change of coordinates

The output is either text-formatted (console mode) or LaTeX-formatted (notebook mode).

EXAMPLES:

Standard embedding of the sphere  $S^2$  in  $\mathbb{R}^3$ :

```

sage: M = Manifold(2, 'S^2', structure='topological') # the 2-dimensional_
↪ sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: N = Manifold(3, 'R^3', latex_name=r'\RR^3', structure='topological') #_
↪ R^3
sage: c_cart.<X,Y,Z> = N.chart() # Cartesian coordinates on R^3
sage: Phi = M.continuous_map(N,
....: { (c_xy, c_cart): [2*x/(1+x^2+y^2), 2*y/(1+x^2+y^2), (x^2+y^2-1)/(1+x^
↪ 2+y^2)],
....: (c_uv, c_cart): [2*u/(1+u^2+v^2), 2*v/(1+u^2+v^2), (1-u^2-v^2)/(1+u^
↪ 2+v^2)] },
....: name='Phi', latex_name=r'\Phi')
sage: Phi.display(c_xy, c_cart)
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2_
↪ + y^2 - 1)/(x^2 + y^2 + 1))
sage: Phi.display(c_uv, c_cart)
Phi: S^2 --> R^3
on V: (u, v) |--> (X, Y, Z) = (2*u/(u^2 + v^2 + 1), 2*v/(u^2 + v^2 + 1), -(u^
↪ 2 + v^2 - 1)/(u^2 + v^2 + 1))

```

The LaTeX output:

```

sage: latex(Phi.display(c_xy, c_cart))
\begin{array}{llcl} \Phi:& S^2 & \longrightarrow & \mathbb{R}^3 \\ \\ \mbox{on} U : & \left(x, y\right) & \longmapsto & \\ & \left(X, Y, Z\right) = \left(\frac{2 \, x}{x^2 + y^2 + 1}, \right. & & \\ & \left. \frac{2 \, y}{x^2 + y^2 + 1}, \right. & &

```

```
\frac{x^{2} + y^{2} - 1}{x^{2} + y^{2} + 1}\right)
\end{array}
```

If the argument `chart2` is not specified, the display is performed on all the charts on the codomain in which the map is known or computable via some change of coordinates (here only one chart: `c_cart`):

```
sage: Phi.display(c_xy)
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2_
↪ + y^2 - 1)/(x^2 + y^2 + 1))
```

Similarly, if the argument `chart1` is omitted, the display is performed on all the charts on the domain of `Phi` in which the map is known or computable via some change of coordinates:

```
sage: Phi.display(chart2=c_cart)
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2_
↪ + y^2 - 1)/(x^2 + y^2 + 1))
on V: (u, v) |--> (X, Y, Z) = (2*u/(u^2 + v^2 + 1), 2*v/(u^2 + v^2 + 1), -(u^
↪ 2 + v^2 - 1)/(u^2 + v^2 + 1))
```

If neither `chart1` nor `chart2` is specified, the display is performed on all the pair of charts in which `Phi` is known or computable via some change of coordinates:

```
sage: Phi.display()
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2_
↪ + y^2 - 1)/(x^2 + y^2 + 1))
on V: (u, v) |--> (X, Y, Z) = (2*u/(u^2 + v^2 + 1), 2*v/(u^2 + v^2 + 1), -(u^
↪ 2 + v^2 - 1)/(u^2 + v^2 + 1))
```

If a chart covers entirely the map's domain, the mention "on ..." is omitted:

```
sage: Phi.restrict(U).display()
Phi: U --> R^3
(x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2 +
↪ y^2 - 1)/(x^2 + y^2 + 1))
```

A shortcut of `display()` is `disp()`:

```
sage: Phi.disp()
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2_
↪ + y^2 - 1)/(x^2 + y^2 + 1))
on V: (u, v) |--> (X, Y, Z) = (2*u/(u^2 + v^2 + 1), 2*v/(u^2 + v^2 + 1), -(u^
↪ 2 + v^2 - 1)/(u^2 + v^2 + 1))
```

**expr** (*chart1=None, chart2=None*)

Return the expression of `self` in terms of specified coordinates.

If the expression is not already known, it is computed from some known expression by means of change-of-chart formulas.

INPUT:

- `chart1` – (default: `None`) chart on the map's domain; if `None`, the domain's default chart is assumed
- `chart2` – (default: `None`) chart on the map's codomain; if `None`, the codomain's default chart is assumed

OUTPUT:

- symbolic expression representing the continuous map in the above two charts

EXAMPLES:

Continuous map from a 2-dimensional manifold to a 3-dimensional one:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: N = Manifold(3, 'N', structure='topological')
sage: c_uv.<u,v> = M.chart()
sage: c_xyz.<x,y,z> = N.chart()
sage: Phi = M.continuous_map(N, (u*v, u/v, u+v), name='Phi',
....:                          latex_name=r'\Phi')
sage: Phi.display()
Phi: M --> N
      (u, v) |--> (x, y, z) = (u*v, u/v, u + v)
sage: Phi.expr(c_uv, c_xyz)
(u*v, u/v, u + v)
sage: Phi.expr() # equivalent to above since 'uv' and 'xyz' are default
<charts
(u*v, u/v, u + v)
sage: type(Phi.expr()[0])
<type 'sage.symbolic.expression.Expression'>
```

Expressions in other charts:

```
sage: c_UV.<U,V> = M.chart() # new chart on M
sage: ch_uv_UV = c_uv.transition_map(c_UV, [u-v, u+v])
sage: ch_uv_UV.inverse()(U,V)
(1/2*U + 1/2*V, -1/2*U + 1/2*V)
sage: c_XYZ.<X,Y,Z> = N.chart() # new chart on N
sage: ch_xyz_XYZ = c_xyz.transition_map(c_XYZ,
....:                                  [2*x-3*y+z, y+z-x, -x+2*y-z])
sage: ch_xyz_XYZ.inverse()(X,Y,Z)
(3*X + Y + 4*Z, 2*X + Y + 3*Z, X + Y + Z)
sage: Phi.expr(c_UV, c_xyz)
(-1/4*U^2 + 1/4*V^2, -(U + V)/(U - V), V)
sage: Phi.expr(c_uv, c_XYZ)
(((2*u + 1)*v^2 + u*v - 3*u)/v,
 -(u - 1)*v^2 - u*v - u)/v,
 -(u + 1)*v^2 + u*v - 2*u)/v)
sage: Phi.expr(c_UV, c_XYZ)
(-1/2*(U^3 - (U - 2)*V^2 + V^3 - (U^2 + 2*U + 6)*V - 6*U)/(U - V),
 1/4*(U^3 - (U + 4)*V^2 + V^3 - (U^2 - 4*U + 4)*V - 4*U)/(U - V),
 1/4*(U^3 - (U - 4)*V^2 + V^3 - (U^2 + 4*U + 8)*V - 8*U)/(U - V))
```

A rotation in some Euclidean plane:

```
sage: M = Manifold(2, 'M', structure='topological') # the plane (minus a
<segment to have global regular spherical coordinates)
sage: c_spher.<r,ph> = M.chart(r'r:(0,+oo) ph:(0,2*pi):\phi') # spherical
<coordinates on the plane
sage: rot = M.continuous_map(M, (r, ph+pi/3), name='R') # pi/3 rotation
<around r=0
sage: rot.expr()
(r, 1/3*pi + ph)
```

Expression of the rotation in terms of Cartesian coordinates:

```

sage: c_cart.<x,y> = M.chart() # Declaration of Cartesian coordinates
sage: ch_spher_cart = c_spher.transition_map(c_cart,
....:          [r*cos(ph), r*sin(ph)]) # relation to spherical_
↳coordinates
sage: ch_spher_cart.set_inverse(sqrt(x^2+y^2), atan2(y,x), verbose=True)
Check of the inverse coordinate transformation:
r == r
ph == arctan2(r*sin(ph), r*cos(ph))
x == x
y == y
sage: rot.expr(c_cart, c_cart)
(-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)

```

**expression** (*chart1=None, chart2=None*)

Return the expression of `self` in terms of specified coordinates.

If the expression is not already known, it is computed from some known expression by means of change-of-chart formulas.

INPUT:

- `chart1` – (default: `None`) chart on the map’s domain; if `None`, the domain’s default chart is assumed
- `chart2` – (default: `None`) chart on the map’s codomain; if `None`, the codomain’s default chart is assumed

OUTPUT:

- symbolic expression representing the continuous map in the above two charts

EXAMPLES:

Continuous map from a 2-dimensional manifold to a 3-dimensional one:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: N = Manifold(3, 'N', structure='topological')
sage: c_uv.<u,v> = M.chart()
sage: c_xyz.<x,y,z> = N.chart()
sage: Phi = M.continuous_map(N, (u*v, u/v, u+v), name='Phi',
....:          latex_name=r'\Phi')
sage: Phi.display()
Phi: M --> N
      (u, v) |--> (x, y, z) = (u*v, u/v, u + v)
sage: Phi.expr(c_uv, c_xyz)
(u*v, u/v, u + v)
sage: Phi.expr() # equivalent to above since 'uv' and 'xyz' are default_
↳charts
(u*v, u/v, u + v)
sage: type(Phi.expr()[0])
<type 'sage.symbolic.expression.Expression'>

```

Expressions in other charts:

```

sage: c_UV.<U,V> = M.chart() # new chart on M
sage: ch_uv_UV = c_uv.transition_map(c_UV, [u-v, u+v])
sage: ch_uv_UV.inverse()(U,V)
(1/2*U + 1/2*V, -1/2*U + 1/2*V)
sage: c_XYZ.<X,Y,Z> = N.chart() # new chart on N
sage: ch_xyz_XYZ = c_xyz.transition_map(c_XYZ,
....:          [2*x-3*y+z, y+z-x, -x+2*y-z])
sage: ch_xyz_XYZ.inverse()(X,Y,Z)

```

```
(3*X + Y + 4*Z, 2*X + Y + 3*Z, X + Y + Z)
sage: Phi.expr(c_UV, c_xyz)
(-1/4*U^2 + 1/4*V^2, -(U + V)/(U - V), V)
sage: Phi.expr(c_uv, c_XYZ)
((2*u + 1)*v^2 + u*v - 3*u)/v,
-(u - 1)*v^2 - u*v - u)/v,
-(u + 1)*v^2 + u*v - 2*u)/v)
sage: Phi.expr(c_UV, c_XYZ)
(-1/2*(U^3 - (U - 2)*V^2 + V^3 - (U^2 + 2*U + 6)*V - 6*U)/(U - V),
1/4*(U^3 - (U + 4)*V^2 + V^3 - (U^2 - 4*U + 4)*V - 4*U)/(U - V),
1/4*(U^3 - (U - 4)*V^2 + V^3 - (U^2 + 4*U + 8)*V - 8*U)/(U - V))
```

A rotation in some Euclidean plane:

```
sage: M = Manifold(2, 'M', structure='topological') # the plane (minus a
↳segment to have global regular spherical coordinates)
sage: c_spher.<r,ph> = M.chart(r'r:(0,+oo) ph:(0,2*pi):\phi') # spherical
↳coordinates on the plane
sage: rot = M.continuous_map(M, (r, ph+pi/3), name='R') # pi/3 rotation
↳around r=0
sage: rot.expr()
(r, 1/3*pi + ph)
```

Expression of the rotation in terms of Cartesian coordinates:

```
sage: c_cart.<x,y> = M.chart() # Declaration of Cartesian coordinates
sage: ch_spher_cart = c_spher.transition_map(c_cart,
.....: [r*cos(ph), r*sin(ph)]) # relation to spherical
↳coordinates
sage: ch_spher_cart.set_inverse(sqrt(x^2+y^2), atan2(y,x), verbose=True)
Check of the inverse coordinate transformation:
r == r
ph == arctan2(r*sin(ph), r*cos(ph))
x == x
y == y
sage: rot.expr(c_cart, c_cart)
(-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)
```

**inverse()**

Return the inverse of `self` if it is an isomorphism.

OUTPUT:

- the inverse isomorphism

EXAMPLES:

The inverse of a rotation in the Euclidean plane:

```
sage: M = Manifold(2, 'R^2', latex_name=r'\RR^2', structure='topological')
sage: c_cart.<x,y> = M.chart()
sage: # A pi/3 rotation around the origin:
sage: rot = M.homeomorphism(M, ((x - sqrt(3)*y)/2, (sqrt(3)*x + y)/2),
.....: name='R')
sage: rot.inverse()
Homeomorphism R^(-1) of the 2-dimensional topological manifold R^2
sage: rot.inverse().display()
R^(-1): R^2 --> R^2
(x, y) |--> (1/2*sqrt(3)*y + 1/2*x, -1/2*sqrt(3)*x + 1/2*y)
```

Checking that applying successively the homeomorphism and its inverse results in the identity:

```
sage: (a, b) = var('a b')
sage: p = M.point((a,b)) # a generic point on M
sage: q = rot(p)
sage: p1 = rot.inverse()(q)
sage: p1 == p
True
```

The result is cached:

```
sage: rot.inverse() is rot.inverse()
True
```

The notations  $\wedge(-1)$  or  $\sim$  can also be used for the inverse:

```
sage: rot^(-1) is rot.inverse()
True
sage: ~rot is rot.inverse()
True
```

An example with multiple charts: the equatorial symmetry on the 2-sphere:

```
sage: M = Manifold(2, 'M', structure='topological') # the 2-dimensional_
↔ sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                                     intersection_name='W',
....:                                     restrictions1=x^2+y^2!=0,
....:                                     restrictions2=u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: s = M.homeomorphism(M, {(c_xy, c_uv): [x, y], (c_uv, c_xy): [u, v]},
....:                         name='s')
sage: s.display()
s: M --> M
on U: (x, y) |--> (u, v) = (x, y)
on V: (u, v) |--> (x, y) = (u, v)
sage: si = s.inverse(); si
Homeomorphism s^(-1) of the 2-dimensional topological manifold M
sage: si.display()
s^(-1): M --> M
on U: (x, y) |--> (u, v) = (x, y)
on V: (u, v) |--> (x, y) = (u, v)
```

The equatorial symmetry is of course an involution:

```
sage: si == s
True
```

### `is_identity()`

Check whether `self` is an identity map.

EXAMPLES:

Tests on continuous maps of a 2-dimensional manifold:

```

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: M.identity_map().is_identity() # obviously...
True
sage: Hom(M, M).one().is_identity() # a variant of the obvious
True
sage: a = M.continuous_map(M, coord_functions={(X,X): (x, y)})
sage: a.is_identity()
True
sage: a = M.continuous_map(M, coord_functions={(X,X): (x, y+1)})
sage: a.is_identity()
False

```

Of course, if the codomain of the map does not coincide with its domain, the outcome is False:

```

sage: N = Manifold(2, 'N', structure='topological')
sage: Y.<u,v> = N.chart()
sage: a = M.continuous_map(N, {(X,Y): (x, y)})
sage: a.display()
M --> N
(x, y) |--> (u, v) = (x, y)
sage: a.is_identity()
False

```

### **restrict** (*subdomain*, *subcodomain*=None)

Restriction of self to some open subset of its domain of definition.

INPUT:

- *subdomain* – *TopologicalManifold*; an open subset of the domain of self
- *subcodomain* – (default: None) an open subset of the codomain of self; if None, the codomain of self is assumed

OUTPUT:

- a *ContinuousMap* that is the restriction of self to subdomain

EXAMPLES:

Restriction to an annulus of a homeomorphism between the open unit disk and  $\mathbb{R}^2$ :

```

sage: M = Manifold(2, 'R^2', structure='topological') # R^2
sage: c_xy.<x,y> = M.chart() # Cartesian coord. on R^2
sage: D = M.open_subset('D', coord_def={c_xy: x^2+y^2<1}) # the open unit disk
sage: Phi = D.continuous_map(M, [x/sqrt(1-x^2-y^2), y/sqrt(1-x^2-y^2)],
....:                          name='Phi', latex_name=r'\Phi')
sage: Phi.display()
Phi: D --> R^2
(x, y) |--> (x, y) = (x/sqrt(-x^2 - y^2 + 1), y/sqrt(-x^2 - y^2 + 1))
sage: c_xy_D = c_xy.restrict(D)
sage: U = D.open_subset('U', coord_def={c_xy_D: x^2+y^2>1/2}) # the annulus 1/
↔ 2 < r < 1
sage: Phi.restrict(U)
Continuous map Phi
from the Open subset U of the 2-dimensional topological manifold R^2
to the 2-dimensional topological manifold R^2
sage: Phi.restrict(U).parent()
Set of Morphisms
from Open subset U of the 2-dimensional topological manifold R^2

```

```

to 2-dimensional topological manifold R^2
in Join of Category of subobjects of sets
and Category of manifolds over Real Field with 53 bits of precision
sage: Phi.domain()
Open subset D of the 2-dimensional topological manifold R^2
sage: Phi.restrict(U).domain()
Open subset U of the 2-dimensional topological manifold R^2
sage: Phi.restrict(U).display()
Phi: U --> R^2
(x, y) |--> (x, y) = (x/sqrt(-x^2 - y^2 + 1), y/sqrt(-x^2 - y^2 + 1))

```

The result is cached:

```

sage: Phi.restrict(U) is Phi.restrict(U)
True

```

The restriction of the identity map:

```

sage: id = D.identity_map() ; id
Identity map Id_D of the Open subset D of the 2-dimensional
topological manifold R^2
sage: id.restrict(U)
Identity map Id_U of the Open subset U of the 2-dimensional
topological manifold R^2
sage: id.restrict(U) is U.identity_map()
True

```

The codomain can be restricted (i.e. made tighter):

```

sage: Phi = D.continuous_map(M, [x/sqrt(1+x^2+y^2), y/sqrt(1+x^2+y^2)])
sage: Phi
Continuous map from
the Open subset D of the 2-dimensional topological manifold R^2
to the 2-dimensional topological manifold R^2
sage: Phi.restrict(D, subcodomain=D)
Continuous map from the Open subset D of the 2-dimensional
topological manifold R^2 to itself

```

**set\_expr** (*chart1*, *chart2*, *coord\_functions*)

Set a new coordinate representation of *self*.

The expressions with respect to other charts are deleted, in order to avoid any inconsistency. To keep them, use `add_expr()` instead.

INPUT:

- *chart1* – chart for the coordinates on the domain of *self*
- *chart2* – chart for the coordinates on the codomain of *self*
- *coord\_functions* – the coordinate symbolic expression of the map in the above charts: list (or tuple) of the coordinates of the image expressed in terms of the coordinates of the considered point; if the dimension of the arrival manifold is 1, a single coordinate expression can be passed instead of a tuple with a single element

EXAMPLES:

Polar representation of a planar rotation initially defined in Cartesian coordinates:

```

sage: M = Manifold(2, 'R^2', latex_name=r'\RR^2', structure='topological') #
↳the Euclidean plane R^2
sage: c_xy.<x,y> = M.chart() # Cartesian coordinate on R^2
sage: U = M.open_subset('U', coord_def={c_xy: (y!=0, x<0)}) # the complement
↳of the segment y=0 and x>0
sage: c_cart = c_xy.restrict(U) # Cartesian coordinates on U
sage: c_spher.<r,ph> = U.chart(r'r:(0,+oo) ph:(0,2*pi):\phi') # spherical
↳coordinates on U
sage: # Links between spherical coordinates and Cartesian ones:
sage: ch_cart_spher = c_cart.transition_map(c_spher,
.....:                                     [sqrt(x*x+y*y), atan2(y,x)])
sage: ch_cart_spher.set_inverse(r*cos(ph), r*sin(ph), verbose=True)
Check of the inverse coordinate transformation:
      x == x
      y == y
      r == r
      ph == arctan2(r*sin(ph), r*cos(ph))
sage: rot = U.continuous_map(U, ((x - sqrt(3)*y)/2, (sqrt(3)*x + y)/2),
.....:                          name='R')
sage: rot.display(c_cart, c_cart)
R: U --> U
      (x, y) |--> (-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)

```

Let us use the method `set_expr()` to set the spherical-coordinate expression by hand:

```

sage: rot.set_expr(c_spher, c_spher, (r, ph+pi/3))
sage: rot.display(c_spher, c_spher)
R: U --> U
      (r, ph) |--> (r, 1/3*pi + ph)

```

The expression in Cartesian coordinates has been erased:

```

sage: rot._coord_expression
{(Chart (U, (r, ph)),
  Chart (U, (r, ph))): Coordinate functions (r, 1/3*pi + ph)
on the Chart (U, (r, ph))}

```

It is recovered (thanks to the known change of coordinates) by a call to `display()`:

```

sage: rot.display(c_cart, c_cart)
R: U --> U
      (x, y) |--> (-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)

sage: rot._coord_expression # random (dictionary output)
{(Chart (U, (x, y)),
  Chart (U, (x, y))): Coordinate functions (-1/2*sqrt(3)*y + 1/2*x,
  1/2*sqrt(3)*x + 1/2*y) on the Chart (U, (x, y)),
  (Chart (U, (r, ph)),
  Chart (U, (r, ph))): Coordinate functions (r, 1/3*pi + ph)
on the Chart (U, (r, ph))}

```

### **set\_expression** (*chart1*, *chart2*, *coord\_functions*)

Set a new coordinate representation of `self`.

The expressions with respect to other charts are deleted, in order to avoid any inconsistency. To keep them, use `add_expr()` instead.

INPUT:

- `chart1` – chart for the coordinates on the domain of `self`
- `chart2` – chart for the coordinates on the codomain of `self`
- `coord_functions` – the coordinate symbolic expression of the map in the above charts: list (or tuple) of the coordinates of the image expressed in terms of the coordinates of the considered point; if the dimension of the arrival manifold is 1, a single coordinate expression can be passed instead of a tuple with a single element

## EXAMPLES:

Polar representation of a planar rotation initially defined in Cartesian coordinates:

```
sage: M = Manifold(2, 'R^2', latex_name=r'\RR^2', structure='topological') #
↳the Euclidean plane R^2
sage: c_xy.<x,y> = M.chart() # Cartesian coordinate on R^2
sage: U = M.open_subset('U', coord_def={c_xy: (y!=0, x<0)}) # the complement
↳of the segment y=0 and x>0
sage: c_cart = c_xy.restrict(U) # Cartesian coordinates on U
sage: c_spher.<r,ph> = U.chart(r'r:(0,+oo) ph:(0,2*pi):\phi') # spherical
↳coordinates on U
sage: # Links between spherical coordinates and Cartesian ones:
sage: ch_cart_spher = c_cart.transition_map(c_spher,
....:                                     [sqrt(x*x+y*y), atan2(y,x)])
sage: ch_cart_spher.set_inverse(r*cos(ph), r*sin(ph), verbose=True)
Check of the inverse coordinate transformation:
    x == x
    y == y
    r == r
    ph == arctan2(r*sin(ph), r*cos(ph))
sage: rot = U.continuous_map(U, ((x - sqrt(3)*y)/2, (sqrt(3)*x + y)/2),
....:                          name='R')
sage: rot.display(c_cart, c_cart)
R: U --> U
    (x, y) |--> (-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)
```

Let us use the method `set_expr()` to set the spherical-coordinate expression by hand:

```
sage: rot.set_expr(c_spher, c_spher, (r, ph+pi/3))
sage: rot.display(c_spher, c_spher)
R: U --> U
    (r, ph) |--> (r, 1/3*pi + ph)
```

The expression in Cartesian coordinates has been erased:

```
sage: rot._coord_expression
{(Chart (U, (r, ph)),
  Chart (U, (r, ph))): Coordinate functions (r, 1/3*pi + ph)
on the Chart (U, (r, ph))}
```

It is recovered (thanks to the known change of coordinates) by a call to `display()`:

```
sage: rot.display(c_cart, c_cart)
R: U --> U
    (x, y) |--> (-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)
sage: rot._coord_expression # random (dictionary output)
{(Chart (U, (x, y)),
  Chart (U, (x, y))): Coordinate functions (-1/2*sqrt(3)*y + 1/2*x,
1/2*sqrt(3)*x + 1/2*y) on the Chart (U, (x, y)),
```

```
(Chart (U, (r, ph)),  
Chart (U, (r, ph)): Coordinate functions (r, 1/3*pi + ph)  
on the Chart (U, (r, ph))}
```



## DIFFERENTIABLE MANIFOLDS

### 2.1 Differentiable Manifolds

Given a non-discrete topological field  $K$  (in most applications,  $K = \mathbf{R}$  or  $K = \mathbf{C}$ ; see however [Ser1992] for  $K = \mathbf{Q}_p$  and [Ber2008] for other fields), a *differentiable manifold over  $K$*  is a topological manifold  $M$  over  $K$  equipped with an atlas whose transition maps are of class  $C^k$  (i.e.  $k$ -times continuously differentiable) for a fixed positive integer  $k$  (possibly  $k = \infty$ ).  $M$  is then called a  *$C^k$ -manifold over  $K$* .

Note that

- if the mention of  $K$  is omitted, then  $K = \mathbf{R}$  is assumed;
- if  $K = \mathbf{C}$ , any  $C^k$ -manifold with  $k \geq 1$  is actually a  $C^\infty$ -manifold (even an analytic manifold);
- if  $K = \mathbf{R}$ , any  $C^k$ -manifold with  $k \geq 1$  admits a compatible  $C^\infty$ -structure (Whitney's smoothing theorem).

Differentiable manifolds are implemented via the class `DifferentiableManifold`. Open subsets of differentiable manifolds are also implemented via `DifferentiableManifold`, since they are differentiable manifolds by themselves.

The user interface is provided by the generic function `Manifold()`, with the argument `structure` set to 'differentiable' and the argument `diff_degree` set to  $k$ , or the argument `structure` set to 'smooth' (the default value).

#### Example 1: the 2-sphere as a differentiable manifold of dimension 2 over $\mathbf{R}$

One starts by declaring  $S^2$  as a 2-dimensional differentiable manifold:

```
sage: M = Manifold(2, 'S^2')
sage: M
2-dimensional differentiable manifold S^2
```

Since the base topological field has not been specified in the argument list of `Manifold`,  $\mathbf{R}$  is assumed:

```
sage: M.base_field()
Real Field with 53 bits of precision
sage: dim(M)
2
```

By default, the created object is a smooth manifold:

```
sage: M.diff_degree()
+Infinity
```

Let us consider the complement of a point, the “North pole” say; this is an open subset of  $S^2$ , which we call  $U$ :

```
sage: U = M.open_subset('U'); U
Open subset U of the 2-dimensional differentiable manifold S^2
```

A standard chart on  $U$  is provided by the stereographic projection from the North pole to the equatorial plane:

```
sage: stereoN.<x,y> = U.chart(); stereoN
Chart (U, (x, y))
```

Thanks to the operator  $\langle x, y \rangle$  on the left-hand side, the coordinates declared in a chart (here  $x$  and  $y$ ), are accessible by their names; they are Sage’s symbolic variables:

```
sage: y
y
sage: type(y)
<type 'sage.symbolic.expression.Expression'>
```

The South pole is the point of coordinates  $(x, y) = (0, 0)$  in the above chart:

```
sage: S = U.point((0,0), chart=stereoN, name='S'); S
Point S on the 2-dimensional differentiable manifold S^2
```

Let us call  $V$  the open subset that is the complement of the South pole and let us introduce on it the chart induced by the stereographic projection from the South pole to the equatorial plane:

```
sage: V = M.open_subset('V'); V
Open subset V of the 2-dimensional differentiable manifold S^2
sage: stereoS.<u,v> = V.chart(); stereoS
Chart (V, (u, v))
```

The North pole is the point of coordinates  $(u, v) = (0, 0)$  in this chart:

```
sage: N = V.point((0,0), chart=stereoS, name='N'); N
Point N on the 2-dimensional differentiable manifold S^2
```

To fully construct the manifold, we declare that it is the union of  $U$  and  $V$ :

```
sage: M.declare_union(U,V)
```

and we provide the transition map between the charts  $\text{stereoN} = (U, (x, y))$  and  $\text{stereoS} = (V, (u, v))$ , denoting by  $W$  the intersection of  $U$  and  $V$  ( $W$  is the subset of  $U$  defined by  $x^2 + y^2 \neq 0$ , as well as the subset of  $V$  defined by  $u^2 + v^2 \neq 0$ ):

```
sage: stereoN_to_S = stereoN.transition_map(stereoS,
....:                                     [x/(x^2+y^2), y/(x^2+y^2)], intersection_name='W',
....:                                     restrictions1= x^2+y^2!=0, restrictions2= u^2+v^2!=0)
sage: stereoN_to_S
Change of coordinates from Chart (W, (x, y)) to Chart (W, (u, v))
sage: stereoN_to_S.display()
u = x/(x^2 + y^2)
v = y/(x^2 + y^2)
```

We give the name  $W$  to the Python variable representing  $W = U \cap V$ :

```
sage: W = U.intersection(V)
```

The inverse of the transition map is computed by the method `inverse()`:

```
sage: stereoN_to_S.inverse()
Change of coordinates from Chart (W, (u, v)) to Chart (W, (x, y))
sage: stereoN_to_S.inverse().display()
x = u/(u^2 + v^2)
y = v/(u^2 + v^2)
```

At this stage, we have four open subsets on  $S^2$ :

```
sage: M.list_of_subsets()
[2-dimensional differentiable manifold S^2,
 Open subset U of the 2-dimensional differentiable manifold S^2,
 Open subset V of the 2-dimensional differentiable manifold S^2,
 Open subset W of the 2-dimensional differentiable manifold S^2]
```

$W$  is the open subset that is the complement of the two poles:

```
sage: N in W or S in W
False
```

The North pole lies in  $V$  and the South pole in  $U$ :

```
sage: N in V, N in U
(True, False)
sage: S in U, S in V
(True, False)
```

The manifold's (user) atlas contains four charts, two of them being restrictions of charts to a smaller domain:

```
sage: M.atlas()
[Chart (U, (x, y)), Chart (V, (u, v)), Chart (W, (x, y)), Chart (W, (u, v))]
```

Let us consider the point of coordinates (1,2) in the chart `stereoN`:

```
sage: p = M.point((1,2), chart=stereoN, name='p'); p
Point p on the 2-dimensional differentiable manifold S^2
sage: p.parent()
2-dimensional differentiable manifold S^2
sage: p in W
True
```

The coordinates of  $p$  in the chart `stereoS` are computed by letting the chart act on the point:

```
sage: stereoS(p)
(1/5, 2/5)
```

Given the definition of  $p$ , we have of course:

```
sage: stereoN(p)
(1, 2)
```

Similarly:

```
sage: stereoS(N)
(0, 0)
sage: stereoN(S)
(0, 0)
```

A differentiable scalar field on the sphere:

```

sage: f = M.scalar_field({stereoN: atan(x^2+y^2), stereoS: pi/2-atan(u^2+v^2)},
....:                      name='f')
sage: f
Scalar field f on the 2-dimensional differentiable manifold S^2
sage: f.display()
f: S^2 --> R
on U: (x, y) |--> arctan(x^2 + y^2)
on V: (u, v) |--> 1/2*pi - arctan(u^2 + v^2)
sage: f(p)
arctan(5)
sage: f(N)
1/2*pi
sage: f(S)
0
sage: f.parent()
Algebra of differentiable scalar fields on the 2-dimensional differentiable
manifold S^2
sage: f.parent().category()
Category of commutative algebras over Symbolic Ring

```

A differentiable manifold has a default vector frame, which, unless otherwise specified, is the coordinate frame associated with the first defined chart:

```

sage: M.default_frame()
Coordinate frame (U, (d/dx,d/dy))
sage: latex(M.default_frame())
\left(U, \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)\right)
sage: M.default_frame() is stereoN.frame()
True

```

A vector field on the sphere:

```

sage: w = M.vector_field('w')
sage: w[stereoN.frame(), :] = [x, y]
sage: w.add_comp_by_continuation(stereoS.frame(), W, stereoS)
sage: w.display() # display in the default frame (stereoN.frame())
w = x d/dx + y d/dy
sage: w.display(stereoS.frame())
w = -u d/du - v d/dv
sage: w.parent()
Module X(S^2) of vector fields on the 2-dimensional differentiable
manifold S^2
sage: w.parent().category()
Category of modules over Algebra of differentiable scalar fields on the
2-dimensional differentiable manifold S^2

```

Vector fields act on scalar fields:

```

sage: w(f)
Scalar field w(f) on the 2-dimensional differentiable manifold S^2
sage: w(f).display()
w(f): S^2 --> R
on U: (x, y) |--> 2*(x^2 + y^2)/(x^4 + 2*x^2*y^2 + y^4 + 1)
on V: (u, v) |--> 2*(u^2 + v^2)/(u^4 + 2*u^2*v^2 + v^4 + 1)
sage: w(f) == f.differential()(w)
True

```

The value of the vector field at point  $p$  is a vector tangent to the sphere:

```

sage: w.at(p)
Tangent vector w at Point p on the 2-dimensional differentiable manifold S^2
sage: w.at(p).display()
w = d/dx + 2 d/dy
sage: w.at(p).parent()
Tangent space at Point p on the 2-dimensional differentiable manifold S^2

```

A 1-form on the sphere:

```

sage: df = f.differential() ; df
1-form df on the 2-dimensional differentiable manifold S^2
sage: df.display()
df = 2*x/(x^4 + 2*x^2*y^2 + y^4 + 1) dx + 2*y/(x^4 + 2*x^2*y^2 + y^4 + 1) dy
sage: df.display(stereoS.frame())
df = -2*u/(u^4 + 2*u^2*v^2 + v^4 + 1) du - 2*v/(u^4 + 2*u^2*v^2 + v^4 + 1) dv
sage: df.parent()
Module /\^1(S^2) of 1-forms on the 2-dimensional differentiable manifold S^2
sage: df.parent().category()
Category of modules over Algebra of differentiable scalar fields on the
2-dimensional differentiable manifold S^2

```

The value of the 1-form at point  $p$  is a linear form on the tangent space at  $p$ :

```

sage: df.at(p)
Linear form df on the Tangent space at Point p on the 2-dimensional
differentiable manifold S^2
sage: df.at(p).display()
df = 1/13 dx + 2/13 dy
sage: df.at(p).parent()
Dual of the Tangent space at Point p on the 2-dimensional differentiable
manifold S^2

```

## Example 2: the Riemann sphere as a differentiable manifold of dimension 1 over $\mathbb{C}$

We declare the Riemann sphere  $\mathbb{C}^*$  as a 1-dimensional differentiable manifold over  $\mathbb{C}$ :

```

sage: M = Manifold(1, 'C*', field='complex'); M
1-dimensional complex manifold C*

```

We introduce a first open subset, which is actually  $\mathbb{C} = \mathbb{C}^* \setminus \{\infty\}$  if we interpret  $\mathbb{C}^*$  as the Alexandroff one-point compactification of  $\mathbb{C}$ :

```

sage: U = M.open_subset('U')

```

A natural chart on  $U$  is then nothing but the identity map of  $\mathbb{C}$ , hence we denote the associated coordinate by  $z$ :

```

sage: Z.<z> = U.chart()

```

The origin of the complex plane is the point of coordinate  $z = 0$ :

```

sage: O = U.point((0,), chart=Z, name='O'); O
Point O on the 1-dimensional complex manifold C*

```

Another open subset of  $\mathbb{C}^*$  is  $V = \mathbb{C}^* \setminus \{O\}$ :

```
sage: V = M.open_subset('V')
```

We define a chart on  $V$  such that the point at infinity is the point of coordinate 0 in this chart:

```
sage: W.<w> = V.chart(); W
Chart (V, (w,))
sage: inf = M.point((0,), chart=W, name='inf', latex_name=r'\infty')
sage: inf
Point inf on the 1-dimensional complex manifold C*
```

To fully construct the Riemann sphere, we declare that it is the union of  $U$  and  $V$ :

```
sage: M.declare_union(U,V)
```

and we provide the transition map between the two charts as  $w = 1/z$  on  $A = U \cap V$ :

```
sage: Z_to_W = Z.transition_map(W, 1/z, intersection_name='A',
....:                          restrictions1= z!=0, restrictions2= w!=0)
sage: Z_to_W
Change of coordinates from Chart (A, (z,)) to Chart (A, (w,))
sage: Z_to_W.display()
w = 1/z
sage: Z_to_W.inverse()
Change of coordinates from Chart (A, (w,)) to Chart (A, (z,))
sage: Z_to_W.inverse().display()
z = 1/w
```

Let consider the complex number  $i$  as a point of the Riemann sphere:

```
sage: i = M((I,), chart=Z, name='i'); i
Point i on the 1-dimensional complex manifold C*
```

Its coordinates w.r.t. the charts  $Z$  and  $W$  are:

```
sage: Z(i)
(I,)
sage: W(i)
(-I,)
```

and we have:

```
sage: i in U
True
sage: i in V
True
```

The following subsets and charts have been defined:

```
sage: M.list_of_subsets()
[Open subset A of the 1-dimensional complex manifold C*,
 1-dimensional complex manifold C*,
 Open subset U of the 1-dimensional complex manifold C*,
 Open subset V of the 1-dimensional complex manifold C*]
sage: M.atlas()
[Chart (U, (z,)), Chart (V, (w,)), Chart (A, (z,)), Chart (A, (w,))]
```

A constant map  $C^* \rightarrow C$ :

```

sage: f = M.constant_scalar_field(3+2*I, name='f'); f
Scalar field f on the 1-dimensional complex manifold C*
sage: f.display()
f: C* --> C
on U: z |--> 2*I + 3
on V: w |--> 2*I + 3
sage: f(0)
2*I + 3
sage: f(i)
2*I + 3
sage: f(inf)
2*I + 3
sage: f.parent()
Algebra of differentiable scalar fields on the 1-dimensional complex
manifold C*
sage: f.parent().category()
Category of commutative algebras over Symbolic Ring

```

A vector field on the Riemann sphere:

```

sage: v = M.vector_field(name='v')
sage: v[Z.frame(), 0] = z^2
sage: v.add_comp_by_continuation(W.frame(), U.intersection(V), W)
sage: v.display(Z.frame())
v = z^2 d/dz
sage: v.display(W.frame())
v = -d/dw
sage: v.parent()
Module X(C*) of vector fields on the 1-dimensional complex manifold C*

```

The vector field  $v$  acting on the scalar field  $f$ :

```

sage: v(f)
Scalar field v(f) on the 1-dimensional complex manifold C*

```

Since  $f$  is constant,  $v(f)$  is vanishing:

```

sage: v(f).display()
v(f): C* --> C
on U: z |--> 0
on V: w |--> 0

```

The value of the vector field  $v$  at the point  $\infty$  is a vector tangent to the Riemann sphere:

```

sage: v.at(inf)
Tangent vector v at Point inf on the 1-dimensional complex manifold C*
sage: v.at(inf).display()
v = -d/dw
sage: v.at(inf).parent()
Tangent space at Point inf on the 1-dimensional complex manifold C*

```

AUTHORS:

- Eric Gourgoulhon (2015): initial version
- Travis Scrimshaw (2016): review tweaks

REFERENCES:

- [Lee2013]
- [KN1963]
- [Huy2005]
- [Ser1992]
- [Ber2008]
- [BG1988]

```
class sage.manifolds.differentiable.manifold.DifferentiableManifold(n, name,
                                                                    field, struc-
                                                                    ture, ambi-
                                                                    ent=None,
                                                                    diff_degree=+Infinity,
                                                                    la-
                                                                    tex_name=None,
                                                                    start_index=0,
                                                                    cate-
                                                                    gory=None,
                                                                    unique_tag=None)
```

Bases: *sage.manifolds.manifold.TopologicalManifold*

Differentiable manifold over a topological field  $K$ .

Given a non-discrete topological field  $K$  (in most applications,  $K = \mathbf{R}$  or  $K = \mathbf{C}$ ; see however [Ser1992] for  $K = \mathbf{Q}_p$  and [Ber2008] for other fields), a *differentiable manifold over  $K$*  is a topological manifold  $M$  over  $K$  equipped with an atlas whose transition maps are of class  $C^k$  (i.e.  $k$ -times continuously differentiable) for a fixed positive integer  $k$  (possibly  $k = \infty$ ).  $M$  is then called a  $C^k$ -manifold over  $K$ .

Note that

- if the mention of  $K$  is omitted, then  $K = \mathbf{R}$  is assumed;
- if  $K = \mathbf{C}$ , any  $C^k$ -manifold with  $k \geq 1$  is actually a  $C^\infty$ -manifold (even an analytic manifold);
- if  $K = \mathbf{R}$ , any  $C^k$ -manifold with  $k \geq 1$  admits a compatible  $C^\infty$ -structure (Whitney's smoothing theorem).

INPUT:

- n – positive integer; dimension of the manifold
- name – string; name (symbol) given to the manifold
- field – field  $K$  on which the manifold is defined; allowed values are
  - 'real' or an object of type `RealField` (e.g., `RR`) for a manifold over  $\mathbf{R}$
  - 'complex' or an object of type `ComplexField` (e.g., `CC`) for a manifold over  $\mathbf{C}$
  - an object in the category of topological fields (see `Fields` and `TopologicalSpaces`) for other types of manifolds
- structure – manifold structure (see *DifferentialStructure* or *RealDifferentialStructure*)
- ambient – (default: `None`) if not `None`, must be a differentiable manifold; the created object is then an open subset of ambient
- diff\_degree – (default: `infinity`) degree  $k$  of differentiability
- latex\_name – (default: `None`) string; LaTeX symbol to denote the manifold; if none is provided, it is set to name

- `start_index` – (default: 0) integer; lower value of the range of indices used for “indexed objects” on the manifold, e.g. coordinates in a chart
- `category` – (default: None) to specify the category; if None, `Manifolds(field).Differentiable()` (or `Manifolds(field).Smooth()` if `diff_degree = infinity`) is assumed (see the category `Manifolds`)
- `unique_tag` – (default: None) tag used to force the construction of a new object when all the other arguments have been used previously (without `unique_tag`, the `UniqueRepresentation` behavior inherited from `ManifoldSubset`, via `TopologicalManifold`, would return the previously constructed object corresponding to these arguments).

## EXAMPLES:

A 4-dimensional differentiable manifold (over  $\mathbf{R}$ ):

```
sage: M = Manifold(4, 'M', latex_name=r'\mathcal{M}'); M
4-dimensional differentiable manifold M
sage: type(M)
<class 'sage.manifolds.differentiable.manifold.DifferentiableManifold_with_
→category'>
sage: latex(M)
\mathcal{M}
sage: dim(M)
4
```

Since the base field has not been specified,  $\mathbf{R}$  has been assumed:

```
sage: M.base_field()
Real Field with 53 bits of precision
```

Since the degree of differentiability has not been specified, the default value,  $C^\infty$ , has been assumed:

```
sage: M.diff_degree()
+Infinity
```

The input parameter `start_index` defines the range of indices on the manifold:

```
sage: M = Manifold(4, 'M')
sage: list(M.irange())
[0, 1, 2, 3]
sage: M = Manifold(4, 'M', start_index=1)
sage: list(M.irange())
[1, 2, 3, 4]
sage: list(Manifold(4, 'M', start_index=-2).irange())
[-2, -1, 0, 1]
```

A complex manifold:

```
sage: N = Manifold(3, 'N', field='complex'); N
3-dimensional complex manifold N
```

A differentiable manifold over  $\mathbf{Q}_5$ , the field of 5-adic numbers:

```
sage: N = Manifold(2, 'N', field=Qp(5)); N
2-dimensional differentiable manifold N over the 5-adic Field with
capped relative precision 20
```

A differentiable manifold is of course a topological manifold:

```
sage: isinstance(M, sage.manifolds.manifold.TopologicalManifold)
True
sage: isinstance(N, sage.manifolds.manifold.TopologicalManifold)
True
```

A differentiable manifold is a Sage *parent* object, in the category of differentiable (here smooth) manifolds over a given topological field (see Manifolds):

```
sage: isinstance(M, Parent)
True
sage: M.category()
Category of smooth manifolds over Real Field with 53 bits of precision
sage: from sage.categories.manifolds import Manifolds
sage: M.category() is Manifolds(RR).Smooth()
True
sage: M.category() is Manifolds(M.base_field()).Smooth()
True
sage: M in Manifolds(RR).Smooth()
True
sage: N in Manifolds(Qp(5)).Smooth()
True
```

The corresponding Sage *elements* are points:

```
sage: X.<t, x, y, z> = M.chart()
sage: p = M.an_element(); p
Point on the 4-dimensional differentiable manifold M
sage: p.parent()
4-dimensional differentiable manifold M
sage: M.is_parent_of(p)
True
sage: p in M
True
```

The manifold's points are instances of class *ManifoldPoint*:

```
sage: isinstance(p, sage.manifolds.point.ManifoldPoint)
True
```

Since an open subset of a differentiable manifold  $M$  is itself a differentiable manifold, open subsets of  $M$  have all attributes of manifolds:

```
sage: U = M.open_subset('U', coord_def={X: t>0}); U
Open subset U of the 4-dimensional differentiable manifold M
sage: U.category()
Join of Category of subobjects of sets and Category of smooth manifolds
over Real Field with 53 bits of precision
sage: U.base_field() == M.base_field()
True
sage: dim(U) == dim(M)
True
```

The manifold passes all the tests of the test suite relative to its category:

```
sage: TestSuite(M).run()
```

**affine\_connection** (*name*, *latex\_name=None*)  
Define an affine connection on the manifold.

See [AffineConnection](#) for a complete documentation.

INPUT:

- `name` – name given to the affine connection
- `latex_name` – (default: None) LaTeX symbol to denote the affine connection

OUTPUT:

- the affine connection, as an instance of [AffineConnection](#)

EXAMPLES:

Affine connection on an open subset of a 3-dimensional smooth manifold:

```
sage: M = Manifold(3, 'M', start_index=1)
sage: A = M.open_subset('A', latex_name=r'\mathcal{A}')
sage: nab = A.affine_connection('nabla', r'\nabla') ; nab
Affine connection nabla on the Open subset A of the 3-dimensional
differentiable manifold M
```

**See also:**

[AffineConnection](#) for more examples.

**automorphism\_field** (*name=None, latex\_name=None, dest\_map=None*)

Define a field of automorphisms (invertible endomorphisms in each tangent space) on `self`.

Via the argument `dest_map`, it is possible to let the field take its values on another manifold. More precisely, if  $M$  is the current manifold,  $N$  a differentiable manifold and  $\Phi : M \rightarrow N$  a differentiable map, a *field of automorphisms along  $M$  with values on  $N$*  is a differentiable map

$$t : M \longrightarrow T^{(1,1)}N$$

( $T^{(1,1)}N$  being the tensor bundle of type  $(1, 1)$  over  $N$ ) such that

$$\forall p \in M, t(p) \in \text{GL}(T_{\Phi(p)}N),$$

where  $\text{GL}(T_{\Phi(p)}N)$  is the general linear group of the tangent space  $T_{\Phi(p)}N$ .

The standard case of a field of automorphisms *on*  $M$  corresponds to  $N = M$  and  $\Phi = \text{Id}_M$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $N$  ( $M$  is then an open interval of  $\mathbf{R}$ ).

**See also:**

[AutomorphismField](#) for complete documentation.

INPUT:

- `name` – (default: None) name given to the field
- `latex_name` – (default: None) LaTeX symbol to denote the field; if none is provided, the LaTeX symbol is set to `name`
- `dest_map` – (default: None) the destination map  $\Phi : M \rightarrow N$ ; if None, it is assumed that  $N = M$  and that  $\Phi$  is the identity map (case of a field of automorphisms *on*  $M$ ), otherwise `dest_map` must be a [DiffMap](#)

OUTPUT:

- a [AutomorphismField](#) (or if  $N$  is parallelizable, a [AutomorphismFieldParal](#)) representing the defined field of automorphisms

EXAMPLES:

A field of automorphisms on a 3-dimensional manifold:

```
sage: M = Manifold(3, 'M')
sage: c_xyz.<x,y,z> = M.chart()
sage: a = M.automorphism_field('A') ; a
Field of tangent-space automorphisms A on the 3-dimensional
differentiable manifold M
sage: a.parent()
General linear group of the Free module X(M) of vector fields on
the 3-dimensional differentiable manifold M
```

**See also:**

For more examples, see *AutomorphismField*.

**automorphism\_field\_group** (*dest\_map=None*)

Return the group of tangent-space automorphism fields defined on *self*, possibly with values in another manifold, as a module over the algebra of scalar fields defined on *self*.

If  $M$  is the current manifold and  $\Phi$  a differentiable map  $\Phi : M \rightarrow N$ , where  $N$  is a differentiable manifold, this method called with *dest\_map* being  $\Phi$  returns the general linear group  $GL(\mathcal{X}(M, \Phi))$  of the module  $\mathcal{X}(M, \Phi)$  of vector fields along  $M$  with values in  $\Phi(M) \subset N$ .

INPUT:

- *dest\_map* – (default: `None`) destination map, i.e. a differentiable map  $\Phi : M \rightarrow N$ , where  $M$  is the current manifold and  $N$  a differentiable manifold; if `None`, it is assumed that  $N = M$  and that  $\Phi$  is the identity map, otherwise *dest\_map* must be a *DiffMap*

OUTPUT:

- a *AutomorphismFieldParalGroup* (if  $N$  is parallelizable) or a *AutomorphismFieldGroup* (if  $N$  is not parallelizable) representing  $GL(\mathcal{X}(U, \Phi))$

EXAMPLES:

Group of tangent-space automorphism fields of a 2-dimensional differentiable manifold:

```
sage: M = Manifold(2, 'M')
sage: M.automorphism_field_group()
General linear group of the Module X(M) of vector fields on the
2-dimensional differentiable manifold M
sage: M.automorphism_field_group().category()
Category of groups
```

**See also:**

For more examples, see *AutomorphismFieldParalGroup* and *AutomorphismFieldGroup*.

**change\_of\_frame** (*frame1, frame2*)

Return a change of vector frames defined on *self*.

INPUT:

- *frame1* – vector frame 1
- *frame2* – vector frame 2

OUTPUT:

- a *AutomorphismField* representing, at each point, the vector space automorphism  $P$  that relates frame 1,  $(e_i)$  say, to frame 2,  $(n_i)$  say, according to  $n_i = P(e_i)$

## EXAMPLES:

Change of vector frames induced by a change of coordinates:

```
sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart()
sage: c_uv.<u,v> = M.chart()
sage: c_xy.transition_map(c_uv, (x+y, x-y))
Change of coordinates from Chart (M, (x, y)) to Chart (M, (u, v))
sage: M.change_of_frame(c_xy.frame(), c_uv.frame())
Field of tangent-space automorphisms on the 2-dimensional
differentiable manifold M
sage: M.change_of_frame(c_xy.frame(), c_uv.frame())[: ]
[ 1/2  1/2]
[ 1/2 -1/2]
sage: M.change_of_frame(c_uv.frame(), c_xy.frame())
Field of tangent-space automorphisms on the 2-dimensional
differentiable manifold M
sage: M.change_of_frame(c_uv.frame(), c_xy.frame())[: ]
[ 1  1]
[ 1 -1]
sage: M.change_of_frame(c_uv.frame(), c_xy.frame()) == \
....:      M.change_of_frame(c_xy.frame(), c_uv.frame()).inverse()
True
```

In the present example, the manifold  $M$  is parallelizable, so that the module  $X(M)$  of vector fields on  $M$  is free. A change of frame on  $M$  is then identical to a change of basis in  $X(M)$ :

```
sage: XM = M.vector_field_module() ; XM
Free module X(M) of vector fields on the 2-dimensional
differentiable manifold M
sage: XM.print_bases()
Bases defined on the Free module X(M) of vector fields on the
2-dimensional differentiable manifold M:
- (M, (d/dx,d/dy)) (default basis)
- (M, (d/du,d/dv))
sage: XM.change_of_basis(c_xy.frame(), c_uv.frame())
Field of tangent-space automorphisms on the 2-dimensional
differentiable manifold M
sage: M.change_of_frame(c_xy.frame(), c_uv.frame()) is \
....:      XM.change_of_basis(c_xy.frame(), c_uv.frame())
True
```

**changes\_of\_frame()**

Return all the changes of vector frames defined on `self`.

## OUTPUT:

- dictionary of fields of tangent-space automorphisms representing the changes of frames, the keys being the pair of frames

## EXAMPLES:

Let us consider a first vector frame on a 2-dimensional differentiable manifold:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: e = X.frame(); e
Coordinate frame (M, (d/dx,d/dy))
```

At this stage, the dictionary of changes of frame is empty:

```
sage: M.changes_of_frame()
{}
```

We introduce a second frame on the manifold, relating it to frame  $e$  by a field of tangent space automorphisms:

```
sage: a = M.automorphism_field(name='a')
sage: a[:] = [[-y, x], [1, 2]]
sage: f = e.new_frame(a, 'f'); f
Vector frame (M, (f_0, f_1))
```

Then we have:

```
sage: M.changes_of_frame() # random (dictionary output)
{(Coordinate frame (M, (d/dx,d/dy)),
  Vector frame (M, (f_0,f_1))): Field of tangent-space
  automorphisms on the 2-dimensional differentiable manifold M,
  (Vector frame (M, (f_0,f_1)),
  Coordinate frame (M, (d/dx,d/dy))): Field of tangent-space
  automorphisms on the 2-dimensional differentiable manifold M}
```

Some checks:

```
sage: M.changes_of_frame()[(e,f)] == a
True
sage: M.changes_of_frame()[(f,e)] == a^(-1)
True
```

### **coframes()**

Return the list of coframes defined on open subsets of `self`.

OUTPUT:

- list of coframes defined on open subsets of `self`

EXAMPLES:

Coframes on subsets of  $\mathbb{R}^2$ :

```
sage: M = Manifold(2, 'R^2')
sage: c_cart.<x,y> = M.chart() # Cartesian coordinates on R^2
sage: M.coframes()
[Coordinate coframe (R^2, (dx,dy))]
sage: e = M.vector_frame('e')
sage: M.coframes()
[Coordinate coframe (R^2, (dx,dy)), Coframe (R^2, (e^0,e^1))]
sage: U = M.open_subset('U', coord_def={c_cart: x^2+y^2<1}) # unit disk
sage: U.coframes()
[Coordinate coframe (U, (dx,dy))]
sage: e.restrict(U)
Vector frame (U, (e_0,e_1))
sage: U.coframes()
[Coordinate coframe (U, (dx,dy)), Coframe (U, (e^0,e^1))]
sage: M.coframes()
[Coordinate coframe (R^2, (dx,dy)),
  Coframe (R^2, (e^0,e^1)),
  Coordinate coframe (U, (dx,dy)),
  Coframe (U, (e^0,e^1))]
```

**curve** (*coord\_expression*, *param*, *chart=None*, *name=None*, *latex\_name=None*)

Define a differentiable curve in the manifold.

**See also:**

*DifferentiableCurve* for details.

INPUT:

- *coord\_expression* – either

- (i) a dictionary whose keys are charts on the manifold and values the coordinate expressions (as lists or tuples) of the curve in the given chart

- (ii) a single coordinate expression in a given chart on the manifold, the latter being provided by the argument *chart*

in both cases, if the dimension of the manifold is 1, a single coordinate expression can be passed instead of a tuple with a single element

- *param* – a tuple of the type  $(t, t_{\min}, t_{\max})$ , where

- t* is the curve parameter used in *coord\_expression*;

- t<sub>min</sub>* is its minimal value;

- t<sub>max</sub>* its maximal value;

if  $t_{\min} = -\text{Infinity}$  and  $t_{\max} = +\text{Infinity}$ , they can be omitted and *t* can be passed for *param* instead of the tuple  $(t, t_{\min}, t_{\max})$

- *chart* – (default: *None*) chart on the manifold used for case (ii) above; if *None* the default chart of the manifold is assumed

- *name* – (default: *None*) string; symbol given to the curve

- *latex\_name* – (default: *None*) string; LaTeX symbol to denote the curve; if none is provided, name will be used

OUTPUT:

- *DifferentiableCurve*

EXAMPLES:

The lemniscate of Gerono in the 2-dimensional Euclidean plane:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: R.<t> = RealLine()
sage: c = M.curve([sin(t), sin(2*t)/2], (t, 0, 2*pi), name='c') ; c
Curve c in the 2-dimensional differentiable manifold M
```

The same definition with the coordinate expression passed as a dictionary:

```
sage: c = M.curve({X: [sin(t), sin(2*t)/2]}, (t, 0, 2*pi), name='c') ; c
Curve c in the 2-dimensional differentiable manifold M
```

An example of definition with *t<sub>min</sub>* and *t<sub>max</sub>* omitted: a helix in  $\mathbf{R}^3$ :

```
sage: R3 = Manifold(3, 'R^3')
sage: X.<x,y,z> = R3.chart()
sage: c = R3.curve([cos(t), sin(t), t], t, name='c') ; c
Curve c in the 3-dimensional differentiable manifold R^3
```

```
sage: c.domain() # check that t is unbounded
Real number line R
```

**See also:**

*DifferentiableCurve* for more examples, including plots.

**default\_frame()**

Return the default vector frame defined on `self`.

By *vector frame*, it is meant a field on the manifold that provides, at each point  $p$ , a vector basis of the tangent space at  $p$ .

Unless changed via *set\_default\_frame()*, the default frame is the first one defined on the manifold, usually implicitly as the coordinate basis associated with the first chart defined on the manifold.

OUTPUT:

- a *VectorFrame* representing the default vector frame

**EXAMPLES:**

The default vector frame is often the coordinate frame associated with the first chart defined on the manifold:

```
sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart()
sage: M.default_frame()
Coordinate frame (M, (d/dx,d/dy))
```

**diff\_degree()**

Return the manifold's degree of differentiability.

The degree of differentiability is the integer  $k$  (possibly  $k = \infty$ ) such that the manifold is a  $C^k$ -manifold over its base field.

**EXAMPLES:**

```
sage: M = Manifold(2, 'M')
sage: M.diff_degree()
+Infinity
sage: M = Manifold(2, 'M', structure='differentiable', diff_degree=3)
sage: M.diff_degree()
3
```

**diff\_form(*degree*, *name=None*, *latex\_name=None*, *dest\_map=None*)**

Define a differential form on `self`.

Via the argument `dest_map`, it is possible to let the differential form take its values on another manifold. More precisely, if  $M$  is the current manifold,  $N$  a differentiable manifold,  $\Phi : M \rightarrow N$  a differentiable map and  $p$  a non-negative integer, a *differential form of degree  $p$*  (or  $p$ -form) *along  $M$  with values on  $N$*  is a differentiable map

$$t : M \longrightarrow T^{(0,p)}N$$

( $T^{(0,p)}N$  being the tensor bundle of type  $(0, p)$  over  $N$ ) such that

$$\forall p \in M, t(p) \in \Lambda^p(T_{\Phi(p)}^*N),$$

where  $\Lambda^p(T_{\Phi(p)}^*N)$  is the  $p$ -th exterior power of the dual of the tangent space  $T_{\Phi(p)}N$ .

The standard case of a differential form *on*  $M$  corresponds to  $N = M$  and  $\Phi = \text{Id}_M$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $N$  ( $M$  is then an open interval of  $\mathbf{R}$ ).

For  $p = 1$ , one can use the method `one_form()` instead.

**See also:**

`DiffForm` for complete documentation.

INPUT:

- `degree` – the degree  $p$  of the differential form (i.e. its tensor rank)
- `name` – (default: `None`) name given to the differential form
- `latex_name` – (default: `None`) LaTeX symbol to denote the differential form; if none is provided, the LaTeX symbol is set to `name`
- `dest_map` – (default: `None`) the destination map  $\Phi : M \rightarrow N$ ; if `None`, it is assumed that  $N = M$  and that  $\Phi$  is the identity map (case of a differential form *on*  $M$ ), otherwise `dest_map` must be a `DiffMap`

OUTPUT:

- the  $p$ -form as a `DiffForm` (or if  $N$  is parallelizable, a `DiffFormParal`)

EXAMPLES:

A 2-form on a open subset of a 4-dimensional differentiable manifold:

```
sage: M = Manifold(4, 'M')
sage: A = M.open_subset('A', latex_name=r'\mathcal{A}'); A
Open subset A of the 4-dimensional differentiable manifold M
sage: c_xyzt.<x,y,z,t> = A.chart()
sage: f = A.diff_form(2, 'F'); f
2-form F on the Open subset A of the 4-dimensional differentiable
manifold M
```

See the documentation of class `DiffForm` for more examples.

**diff\_form\_module** (*degree, dest\_map=None*)

Return the set of differential forms of a given degree defined on `self`, possibly with values in another manifold, as a module over the algebra of scalar fields defined on `self`.

**See also:**

`DiffFormModule` for complete documentation.

INPUT:

- `degree` – positive integer; the degree  $p$  of the differential forms
- `dest_map` – (default: `None`) destination map, i.e. a differentiable map  $\Phi : M \rightarrow N$ , where  $M$  is the current manifold and  $N$  a differentiable manifold; if `None`, it is assumed that  $N = M$  and that  $\Phi$  is the identity map (case of differential forms *on*  $M$ ), otherwise `dest_map` must be a `DiffMap`

OUTPUT:

- a `DiffFormModule` (or if  $N$  is parallelizable, a `DiffFormFreeModule`) representing the module  $\Lambda^p(M, \Phi)$  of  $p$ -forms on  $M$  taking values on  $\Phi(M) \subset N$

EXAMPLES:

Module of 2-forms on a 3-dimensional parallelizable manifold:

```

sage: M = Manifold(3, 'M')
sage: X.<x, y, z> = M.chart()
sage: M.diff_form_module(2)
Free module /\^2(M) of 2-forms on the 3-dimensional differentiable
manifold M
sage: M.diff_form_module(2).category()
Category of finite dimensional modules over Algebra of
differentiable scalar fields on the 3-dimensional differentiable
manifold M
sage: M.diff_form_module(2).base_ring()
Algebra of differentiable scalar fields on the 3-dimensional
differentiable manifold M
sage: M.diff_form_module(2).rank()
3

```

The outcome is cached:

```

sage: M.diff_form_module(2) is M.diff_form_module(2)
True

```

**diff\_map**(*codomain*, *coord\_functions=None*, *chart1=None*, *chart2=None*, *name=None*, *latex\_name=None*)

Define a differentiable map between the current differentiable manifold and a differentiable manifold over the same topological field.

See [DiffMap](#) for a complete documentation.

INPUT:

- *codomain* – the map codomain (a differentiable manifold over the same topological field as the current differentiable manifold)
- *coord\_functions* – (default: None) if not None, must be either
  - (i) a dictionary of the coordinate expressions (as lists (or tuples) of the coordinates of the image expressed in terms of the coordinates of the considered point) with the pairs of charts (*chart1*, *chart2*) as keys (*chart1* being a chart on the current manifold and *chart2* a chart on *codomain*)
  - (ii) a single coordinate expression in a given pair of charts, the latter being provided by the arguments *chart1* and *chart2*

In both cases, if the dimension of the arrival manifold is 1, a single coordinate expression can be passed instead of a tuple with a single element

- *chart1* – (default: None; used only in case (ii) above) chart on the current manifold defining the start coordinates involved in *coord\_functions* for case (ii); if none is provided, the coordinates are assumed to refer to the manifold's default chart
- *chart2* – (default: None; used only in case (ii) above) chart on *codomain* defining the arrival coordinates involved in *coord\_functions* for case (ii); if none is provided, the coordinates are assumed to refer to the default chart of *codomain*
- *name* – (default: None) name given to the differentiable map
- *latex\_name* – (default: None) LaTeX symbol to denote the differentiable map; if none is provided, the LaTeX symbol is set to *name*

OUTPUT:

- the differentiable map, as an instance of [DiffMap](#)

## EXAMPLES:

A differentiable map between an open subset of  $S^2$  covered by regular spherical coordinates and  $\mathbb{R}^3$ :

```
sage: M = Manifold(2, 'S^2')
sage: U = M.open_subset('U')
sage: c_spher.<th,ph> = U.chart(r'th:(0,pi):\theta ph:(0,2*pi):\phi')
sage: N = Manifold(3, 'R^3', r'\RR^3')
sage: c_cart.<x,y,z> = N.chart() # Cartesian coord. on R^3
sage: Phi = U.diff_map(N, (sin(th)*cos(ph), sin(th)*sin(ph), cos(th)),
....:                      name='Phi', latex_name=r'\Phi')
sage: Phi
Differentiable map Phi from the Open subset U of the 2-dimensional
differentiable manifold S^2 to the 3-dimensional differentiable
manifold R^3
```

The same definition, but with a dictionary with pairs of charts as keys (case (i) above):

```
sage: Phi1 = U.diff_map(N,
....:      {(c_spher, c_cart): (sin(th)*cos(ph), sin(th)*sin(ph),
....:      cos(th))}, name='Phi', latex_name=r'\Phi')
sage: Phi1 == Phi
True
```

The differentiable map acting on a point:

```
sage: p = U.point((pi/2, pi)) ; p
Point on the 2-dimensional differentiable manifold S^2
sage: Phi(p)
Point on the 3-dimensional differentiable manifold R^3
sage: Phi(p).coord(c_cart)
(-1, 0, 0)
sage: Phi1(p) == Phi(p)
True
```

See the documentation of class *DiffMap* for more examples.

**diff\_mapping**(*codomain*, *coord\_functions=None*, *chart1=None*, *chart2=None*, *name=None*, *latex\_name=None*)

Deprecated.

Use *diff\_map()* instead.

## EXAMPLES:

```
sage: M = Manifold(2, 'M'); X.<x,y> = M.chart()
sage: N = Manifold(2, 'N'); Y.<u,v> = N.chart()
sage: Phi = M.diff_mapping(N, {(X,Y): [x+y, x-y]}, name='Phi')
doctest:...: DeprecationWarning: Use diff_map() instead.
See http://trac.sagemath.org/18783 for details.
sage: Phi
Differentiable map Phi from the 2-dimensional differentiable
manifold M to the 2-dimensional differentiable manifold N
```

**diffeomorphism**(*codomain*, *coord\_functions=None*, *chart1=None*, *chart2=None*, *name=None*, *latex\_name=None*)

Define a diffeomorphism between the current manifold and another one.

See *DiffMap* for a complete documentation.

INPUT:

- `codomain` – codomain of the diffeomorphism (the arrival manifold or some subset of it)
  - `coord_functions` – (default: `None`) if not `None`, must be either
    - (i) a dictionary of the coordinate expressions (as lists (or tuples) of the coordinates of the image expressed in terms of the coordinates of the considered point) with the pairs of charts (`chart1`, `chart2`) as keys (`chart1` being a chart on the current manifold and `chart2` a chart on `codomain`)
    - (ii) a single coordinate expression in a given pair of charts, the latter being provided by the arguments `chart1` and `chart2`
- In both cases, if the dimension of the arrival manifold is 1, a single coordinate expression can be passed instead of a tuple with a single element
- `chart1` – (default: `None`; used only in case (ii) above) chart on the current manifold defining the start coordinates involved in `coord_functions` for case (ii); if none is provided, the coordinates are assumed to refer to the manifold’s default chart
  - `chart2` – (default: `None`; used only in case (ii) above) chart on `codomain` defining the arrival coordinates involved in `coord_functions` for case (ii); if none is provided, the coordinates are assumed to refer to the default chart of `codomain`
  - `name` – (default: `None`) name given to the diffeomorphism
  - `latex_name` – (default: `None`) LaTeX symbol to denote the diffeomorphism; if none is provided, the LaTeX symbol is set to `name`

OUTPUT:

- the diffeomorphism, as an instance of `DiffMap`

EXAMPLES:

Diffeomorphism between the open unit disk in  $\mathbf{R}^2$  and  $\mathbf{R}^2$ :

```
sage: M = Manifold(2, 'M') # the open unit disk
sage: forget() # for doctests only
sage: c_xy.<x,y> = M.chart('x:(-1,1) y:(-1,1)') # Cartesian coord on M
sage: c_xy.add_restrictions(x^2+y^2<1)
sage: N = Manifold(2, 'N') # R^2
sage: c_XY.<X,Y> = N.chart() # canonical coordinates on R^2
sage: Phi = M.diffeomorphism(N, [x/sqrt(1-x^2-y^2), y/sqrt(1-x^2-y^2)],
....:                               name='Phi', latex_name=r'\Phi')
sage: Phi
Diffeomorphism Phi from the 2-dimensional differentiable manifold M
to the 2-dimensional differentiable manifold N
sage: Phi.display()
Phi: M --> N
(x, y) |--> (X, Y) = (x/sqrt(-x^2 - y^2 + 1), y/sqrt(-x^2 - y^2 + 1))
```

The inverse diffeomorphism:

```
sage: Phi^(-1)
Diffeomorphism Phi^(-1) from the 2-dimensional differentiable
manifold N to the 2-dimensional differentiable manifold M
sage: (Phi^(-1)).display()
Phi^(-1): N --> M
(X, Y) |--> (x, y) = (X/sqrt(X^2 + Y^2 + 1), Y/sqrt(X^2 + Y^2 + 1))
```

See the documentation of class `DiffMap` for more examples.

**frames()**

Return the list of vector frames defined on open subsets of `self`.

OUTPUT:

- list of vector frames defined on open subsets of `self`

EXAMPLES:

Vector frames on subsets of  $\mathbf{R}^2$ :

```
sage: M = Manifold(2, 'R^2')
sage: c_cart.<x,y> = M.chart() # Cartesian coordinates on R^2
sage: M.frames()
[Coordinate frame (R^2, (d/dx,d/dy))]
sage: e = M.vector_frame('e')
sage: M.frames()
[Coordinate frame (R^2, (d/dx,d/dy)),
 Vector frame (R^2, (e_0,e_1))]
sage: U = M.open_subset('U', coord_def={c_cart: x^2+y^2<1}) # unit disk
sage: U.frames()
[Coordinate frame (U, (d/dx,d/dy))]
sage: M.frames()
[Coordinate frame (R^2, (d/dx,d/dy)),
 Vector frame (R^2, (e_0,e_1)),
 Coordinate frame (U, (d/dx,d/dy))]
```

**is\_manifestly\_parallelizable()**

Return True if `self` is known to be a parallelizable and False otherwise.

If False is returned, either the manifold is not parallelizable or no vector frame has been defined on it yet.

EXAMPLES:

A just created manifold is a priori not manifestly parallelizable:

```
sage: M = Manifold(2, 'M')
sage: M.is_manifestly_parallelizable()
False
```

Defining a vector frame on it makes it parallelizable:

```
sage: e = M.vector_frame('e')
sage: M.is_manifestly_parallelizable()
True
```

Defining a coordinate chart on the whole manifold also makes it parallelizable:

```
sage: N = Manifold(4, 'N')
sage: X.<t,x,y,z> = N.chart()
sage: N.is_manifestly_parallelizable()
True
```

**lorentz\_metric** (*name, signature='positive', latex\_name=None, dest\_map=None*)

Deprecated.

Use `lorentzian_metric()` instead.

EXAMPLES:

```

sage: M = Manifold(4, 'M')
sage: g = M.lorentz_metric('g')
doctest:...: DeprecationWarning: Use lorentzian_metric() instead.
See http://trac.sagemath.org/19209 for details.
sage: g
Lorentzian metric g on the 4-dimensional differentiable manifold M

```

**lorentzian\_metric** (*name, signature='positive', latex\_name=None, dest\_map=None*)

Define a Lorentzian metric on the manifold.

A *Lorentzian metric* is a field of nondegenerate symmetric bilinear forms acting in the tangent spaces, with signature  $(-, +, \dots, +)$  or  $(+, -, \dots, -)$ .

See *PseudoRiemannianMetric* for a complete documentation.

INPUT:

- *name* – name given to the metric
- *signature* – (default: 'positive') sign of the metric signature:
  - if set to 'positive', the signature is  $n-2$ , where  $n$  is the manifold's dimension, i.e.  $(-, +, \dots, +)$
  - if set to 'negative', the signature is  $-n+2$ , i.e.  $(+, -, \dots, -)$
- *latex\_name* – (default: None) LaTeX symbol to denote the metric; if None, it is formed from *name*
- *dest\_map* – (default: None) instance of class *DiffMap* representing the destination map  $\Phi : U \rightarrow M$ , where  $U$  is the current manifold; if None, the identity map is assumed (case of a metric field on  $U$ )

OUTPUT:

- instance of *PseudoRiemannianMetric* representing the defined Lorentzian metric.

EXAMPLES:

Metric of Minkowski spacetime:

```

sage: M = Manifold(4, 'M')
sage: X.<t,x,y,z> = M.chart()
sage: g = M.lorentzian_metric('g'); g
Lorentzian metric g on the 4-dimensional differentiable manifold M
sage: g[0,0], g[1,1], g[2,2], g[3,3] = -1, 1, 1, 1
sage: g.display()
g = -dt*dt + dx*dx + dy*dy + dz*dz
sage: g.signature()
2

```

Choice of a negative signature:

```

sage: g = M.lorentzian_metric('g', signature='negative'); g
Lorentzian metric g on the 4-dimensional differentiable manifold M
sage: g[0,0], g[1,1], g[2,2], g[3,3] = 1, -1, -1, -1
sage: g.display()
g = dt*dt - dx*dx - dy*dy - dz*dz
sage: g.signature()
-2

```

**metric** (*name, signature=None, latex\_name=None, dest\_map=None*)

Define a pseudo-Riemannian metric on the manifold.

A *pseudo-Riemannian metric* is a field of nondegenerate symmetric bilinear forms acting in the tangent spaces. See [PseudoRiemannianMetric](#) for a complete documentation.

INPUT:

- `name` – name given to the metric
- `signature` – (default: `None`) signature  $S$  of the metric as a single integer:  $S = n_+ - n_-$ , where  $n_+$  (resp.  $n_-$ ) is the number of positive terms (resp. number of negative terms) in any diagonal writing of the metric components; if `signature` is not provided,  $S$  is set to the manifold's dimension (Riemannian signature)
- `latex_name` – (default: `None`) LaTeX symbol to denote the metric; if `None`, it is formed from `name`
- `dest_map` – (default: `None`) instance of class [DiffMap](#) representing the destination map  $\Phi : U \rightarrow M$ , where  $U$  is the current manifold; if `None`, the identity map is assumed (case of a metric field on  $U$ )

OUTPUT:

- instance of [PseudoRiemannianMetric](#) representing the defined pseudo-Riemannian metric.

EXAMPLES:

Metric on a 3-dimensional manifold:

```
sage: M = Manifold(3, 'M', start_index=1)
sage: c_xyz.<x,y,z> = M.chart()
sage: g = M.metric('g'); g
Riemannian metric g on the 3-dimensional differentiable manifold M
```

**See also:**

[PseudoRiemannianMetric](#) for more examples.

**one\_form** (`name=None`, `latex_name=None`, `dest_map=None`)

Define a 1-form on the manifold.

Via the argument `dest_map`, it is possible to let the 1-form take its values on another manifold. More precisely, if  $M$  is the current manifold,  $N$  a differentiable manifold and  $\Phi : M \rightarrow N$  a differentiable map, a *1-form along  $M$  with values on  $N$*  is a differentiable map

$$t : M \longrightarrow T^*N$$

( $T^*N$  being the cotangent bundle of  $N$ ) such that

$$\forall p \in M, t(p) \in T_{\Phi(p)}^*N,$$

where  $T_{\Phi(p)}^*$  is the dual of the tangent space  $T_{\Phi(p)}N$ .

The standard case of a 1-form *on*  $M$  corresponds to  $N = M$  and  $\Phi = \text{Id}_M$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $N$  ( $M$  is then an open interval of  $\mathbf{R}$ ).

**See also:**

[DiffForm](#) for complete documentation.

INPUT:

- `name` – (default: `None`) name given to the 1-form
- `latex_name` – (default: `None`) LaTeX symbol to denote the 1-form; if none is provided, the LaTeX symbol is set to `name`

- `dest_map` – (default: `None`) the destination map  $\Phi : M \rightarrow N$ ; if `None`, it is assumed that  $N = M$  and that  $\Phi$  is the identity map (case of a 1-form on  $M$ ), otherwise `dest_map` must be a *DiffMap*

OUTPUT:

- the 1-form as a *DiffForm* (or if  $N$  is parallelizable, a *DiffFormParal*)

EXAMPLES:

A 1-form on a 3-dimensional open subset:

```
sage: M = Manifold(3, 'M')
sage: A = M.open_subset('A', latex_name=r'\mathcal{A}')
sage: X.<x,y,z> = A.chart()
sage: om = A.one_form('omega', r'\omega') ; om
1-form omega on the Open subset A of the 3-dimensional
differentiable manifold M
sage: om.parent()
Free module /\^1(A) of 1-forms on the Open subset A of the
3-dimensional differentiable manifold M
```

**See also:**

For more examples, see *DiffForm*.

**open\_subset** (*name*, *latex\_name=None*, *coord\_def={}*)

Create an open subset of the manifold.

An open subset is a set that is (i) included in the manifold and (ii) open with respect to the manifold's topology. It is a differentiable manifold by itself. Hence the returned object is an instance of *DifferentiableManifold*.

INPUT:

- `name` – name given to the open subset
- `latex_name` – (default: `None`) LaTeX symbol to denote the subset; if none is provided, it is set to `name`
- `coord_def` – (default: `{}`) definition of the subset in terms of coordinates; `coord_def` must be a dictionary with keys `charts` in the manifold's atlas and values the symbolic expressions formed by the coordinates to define the subset.

OUTPUT:

- the open subset, as an instance of *DifferentiableManifold*

EXAMPLES:

Creating an open subset of a differentiable manifold:

```
sage: M = Manifold(2, 'M')
sage: A = M.open_subset('A'); A
Open subset A of the 2-dimensional differentiable manifold M
```

As an open subset of a differentiable manifold,  $A$  is itself a differentiable manifold, on the same topological field and of the same dimension as  $M$ :

```
sage: A.category()
Join of Category of subobjects of sets and Category of smooth
manifolds over Real Field with 53 bits of precision
sage: A.base_field() == M.base_field()
True
```

```
sage: dim(A) == dim(M)
True
```

Creating an open subset of A:

```
sage: B = A.open_subset('B'); B
Open subset B of the 2-dimensional differentiable manifold M
```

We have then:

```
sage: A.list_of_subsets()
[Open subset A of the 2-dimensional differentiable manifold M,
 Open subset B of the 2-dimensional differentiable manifold M]
sage: B.is_subset(A)
True
sage: B.is_subset(M)
True
```

Defining an open subset by some coordinate restrictions: the open unit disk in of the Euclidean plane:

```
sage: X.<x,y> = M.chart() # Cartesian coordinates on M
sage: U = M.open_subset('U', coord_def={X: x^2+y^2<1}); U
Open subset U of the 2-dimensional differentiable manifold M
```

Since the argument `coord_def` has been set, U is automatically endowed with a chart, which is the restriction of X to U:

```
sage: U.atlas()
[Chart (U, (x, y))]
sage: U.default_chart()
Chart (U, (x, y))
sage: U.default_chart() is X.restrict(U)
True
```

An point in U:

```
sage: p = U.an_element(); p
Point on the 2-dimensional differentiable manifold M
sage: X(p) # the coordinates (x,y) of p
(0, 0)
sage: p in U
True
```

Checking whether various points, defined by their coordinates w.r.t. chart X, are in U:

```
sage: M((0,1/2)) in U
True
sage: M((0,1)) in U
False
sage: M((1/2,1)) in U
False
sage: M((-1/2,1/3)) in U
True
```

**riemann\_metric** (*name*, *latex\_name=None*, *dest\_map=None*)

Deprecated.

Use `riemannian_metric()` instead.

EXAMPLES:

```
sage: M = Manifold(3, 'M')
sage: g = M.riemann_metric('g')
doctest:...: DeprecationWarning: Use riemannian_metric() instead.
See http://trac.sagemath.org/19209 for details.
sage: g
Riemannian metric g on the 3-dimensional differentiable manifold M
```

**riemannian\_metric** (*name*, *latex\_name=None*, *dest\_map=None*)

Define a Riemannian metric on the manifold.

A *Riemannian metric* is a field of positive definite symmetric bilinear forms acting in the tangent spaces.

See *PseudoRiemannianMetric* for a complete documentation.

INPUT:

- *name* – name given to the metric
- *latex\_name* – (default: None) LaTeX symbol to denote the metric; if None, it is formed from *name*
- *dest\_map* – (default: None) instance of class *DiffMap* representing the destination map  $\Phi : U \rightarrow M$ , where  $U$  is the current manifold; if None, the identity map is assumed (case of a metric field *on*  $U$ )

OUTPUT:

- instance of *PseudoRiemannianMetric* representing the defined Riemannian metric.

EXAMPLES:

Metric of the hyperbolic plane  $H^2$ :

```
sage: H2 = Manifold(2, 'H^2', start_index=1)
sage: X.<x,y> = H2.chart('x y: (0,+oo)') # Poincaré half-plane coord.
sage: g = H2.riemannian_metric('g')
sage: g[1,1], g[2,2] = 1/y^2, 1/y^2
sage: g
Riemannian metric g on the 2-dimensional differentiable manifold H^2
sage: g.display()
g = y^(-2) dx*dx + y^(-2) dy*dy
sage: g.signature()
2
```

See also:

*PseudoRiemannianMetric* for more examples.

**set\_change\_of\_frame** (*frame1*, *frame2*, *change\_of\_frame*, *compute\_inverse=True*)

Relate two vector frames by an automorphism.

This updates the internal dictionary `self._frame_changes`.

INPUT:

- *frame1* – frame 1, denoted  $(e_i)$  below
- *frame2* – frame 2, denoted  $(f_i)$  below
- *change\_of\_frame* – instance of class *AutomorphismFieldParal* describing the automorphism  $P$  that relates the basis  $(e_i)$  to the basis  $(f_i)$  according to  $f_i = P(e_i)$

- `compute_inverse` (default: True) – if set to True, the inverse automorphism is computed and the change from basis  $(f_i)$  to  $(e_i)$  is set to it in the internal dictionary `self._frame_changes`

EXAMPLES:

Connecting two vector frames on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart()
sage: e = M.vector_frame('e')
sage: f = M.vector_frame('f')
sage: a = M.automorphism_field()
sage: a[e,:] = [[1,2],[0,3]]
sage: M.set_change_of_frame(e, f, a)
sage: f[0].display(e)
f_0 = e_0
sage: f[1].display(e)
f_1 = 2 e_0 + 3 e_1
sage: e[0].display(f)
e_0 = f_0
sage: e[1].display(f)
e_1 = -2/3 f_0 + 1/3 f_1
sage: M.change_of_frame(e, f)[e,:]
[1 2]
[0 3]
```

**set\_default\_frame** (*frame*)

Changing the default vector frame on `self`.

INPUT:

- *frame* – *VectorFrame* a vector frame defined on the manifold

EXAMPLES:

Changing the default frame on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart()
sage: e = M.vector_frame('e')
sage: M.default_frame()
Coordinate frame (M, (d/dx,d/dy))
sage: M.set_default_frame(e)
sage: M.default_frame()
Vector frame (M, (e_0,e_1))
```

**sym\_bilin\_form\_field** (*name=None, latex\_name=None, dest\_map=None*)

Define a field of symmetric bilinear forms on `self`.

Via the argument `dest_map`, it is possible to let the field take its values on another manifold. More precisely, if  $M$  is the current manifold,  $N$  a differentiable manifold and  $\Phi : M \rightarrow N$  a differentiable map, a *field of symmetric bilinear forms along  $M$  with values on  $N$*  is a differentiable map

$$t : M \longrightarrow T^{(0,2)}N$$

$(T^{(0,2)}N)$  being the tensor bundle of type  $(0, 2)$  over  $N$  such that

$$\forall p \in M, t(p) \in S(T_{\Phi(p)}N),$$

where  $S(T_{\Phi(p)}N)$  is the space of symmetric bilinear forms on the tangent space  $T_{\Phi(p)}N$ .

The standard case of fields of symmetric bilinear forms *on*  $M$  corresponds to  $N = M$  and  $\Phi = \text{Id}_M$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $N$  ( $M$  is then an open interval of  $\mathbf{R}$ ).

INPUT:

- `name` – (default: `None`) name given to the field
- `latex_name` – (default: `None`) LaTeX symbol to denote the field; if none is provided, the LaTeX symbol is set to `name`
- `dest_map` – (default: `None`) the destination map  $\Phi : M \rightarrow N$ ; if `None`, it is assumed that  $N = M$  and that  $\Phi$  is the identity map (case of a field *on*  $M$ ), otherwise `dest_map` must be an instance of class `DiffMap`

OUTPUT:

- a `TensorField` (or if  $N$  is parallelizable, a `TensorFieldParal`) of tensor type  $(0, 2)$  and symmetric representing the defined field of symmetric bilinear forms

EXAMPLES:

A field of symmetric bilinear forms on a 3-dimensional manifold:

```
sage: M = Manifold(3, 'M')
sage: c_xyz.<x,y,z> = M.chart()
sage: t = M.sym_bilin_form_field('T'); t
Field of symmetric bilinear forms T on the 3-dimensional
differentiable manifold M
```

Such a object is a tensor field of rank 2 and type  $(0, 2)$ :

```
sage: t.parent()
Free module T^(0,2)(M) of type-(0,2) tensors fields on the
3-dimensional differentiable manifold M
sage: t.tensor_rank()
2
sage: t.tensor_type()
(0, 2)
```

The LaTeX symbol is deduced from the name or can be specified when creating the object:

```
sage: latex(t)
T
sage: om = M.sym_bilin_form_field('Omega', r'\Omega')
sage: latex(om)
\Omega
```

Components with respect to some vector frame:

```
sage: e = M.vector_frame('e') ; M.set_default_frame(e)
sage: t.set_comp()
Fully symmetric 2-indices components w.r.t. Vector frame
(M, (e_0, e_1, e_2))
sage: type(t.comp())
<class 'sage.tensor.modules.comp.CompFullySym'>
```

For the default frame, the components are accessed with the square brackets:

```
sage: t[0,0], t[0,1], t[0,2] = (1, 2, 3)
sage: t[1,1], t[1,2] = (4, 5)
sage: t[2,2] = 6
```

The other components are deduced by symmetry:

```
sage: t[1,0], t[2,0], t[2,1]
(2, 3, 5)
sage: t[:]
[1 2 3]
[2 4 5]
[3 5 6]
```

A symmetric bilinear form acts on vector pairs:

```
sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart()
sage: t = M.sym_bilin_form_field('T')
sage: t[0,0], t[0,1], t[1,1] = (-1, x, y*x)
sage: v1 = M.vector_field('V_1')
sage: v1[:] = (y,x)
sage: v2 = M.vector_field('V_2')
sage: v2[:] = (x+y,2)
sage: s = t(v1,v2) ; s
Scalar field T(V_1,V_2) on the 2-dimensional differentiable
manifold M
sage: s.expr()
x^3 + (3*x^2 + x)*y - y^2
sage: s.expr() - t[0,0]*v1[0]*v2[0] - \
....: t[0,1]*(v1[0]*v2[1]+v1[1]*v2[0]) - t[1,1]*v1[1]*v2[1]
0
sage: latex(s)
T\left(V_1,V_2\right)
```

Adding two symmetric bilinear forms results in another symmetric bilinear form:

```
sage: a = M.sym_bilin_form_field()
sage: a[0,0], a[0,1], a[1,1] = (1,2,3)
sage: b = M.sym_bilin_form_field()
sage: b[0,0], b[0,1], b[1,1] = (-1,4,5)
sage: s = a + b ; s
Field of symmetric bilinear forms on the 2-dimensional
differentiable manifold M
sage: s[:]
[0 6]
[6 8]
```

But adding a symmetric bilinear form with a non-symmetric bilinear form results in a generic type (0,2) tensor:

```
sage: c = M.tensor_field(0,2)
sage: c[:] = [[-2, -3], [1,7]]
sage: s1 = a + c ; s1
Tensor field of type (0,2) on the 2-dimensional differentiable
manifold M
sage: s1[:]
[-1 -1]
[ 3 10]
```

```

sage: s2 = c + a ; s2
Tensor field of type (0,2) on the 2-dimensional differentiable
manifold M
sage: s2[:]
[-1 -1]
[ 3 10]

```

**tangent\_identity\_field** (*name='Id', latex\_name=None, dest\_map=None*)

Return the field of identity maps in the tangent spaces on *self*.

Via the argument *dest\_map*, it is possible to let the field take its values on another manifold. More precisely, if  $M$  is the current manifold,  $N$  a differentiable manifold and  $\Phi : M \rightarrow N$  a differentiable map, a *field of identity maps along  $M$  with values on  $N$*  is a differentiable map

$$t : M \longrightarrow T^{(1,1)}N$$

( $T^{(1,1)}N$  being the tensor bundle of type  $(1, 1)$  over  $N$ ) such that

$$\forall p \in M, t(p) = \text{Id}_{T_{\Phi(p)}N},$$

where  $\text{Id}_{T_{\Phi(p)}N}$  is the identity map of the tangent space  $T_{\Phi(p)}N$ .

The standard case of a field of identity maps *on*  $M$  corresponds to  $N = M$  and  $\Phi = \text{Id}_M$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $N$  ( $M$  is then an open interval of  $\mathbf{R}$ ).

INPUT:

- *name* – (string; default: ‘Id’) name given to the field of identity maps
- *latex\_name* – (string; default: None) LaTeX symbol to denote the field of identity map; if none is provided, the LaTeX symbol is set to ‘ $\text{Id}$ ’ if *name* is ‘Id’ and to *name* otherwise
- *dest\_map* – (default: None) the destination map  $\Phi : M \rightarrow N$ ; if None, it is assumed that  $N = M$  and that  $\Phi$  is the identity map (case of a field of identity maps *on*  $M$ ), otherwise *dest\_map* must be a *DiffMap*

OUTPUT:

- a *AutomorphismField* (or if  $N$  is parallelizable, a *AutomorphismFieldParal*) representing the field of identity maps

EXAMPLES:

Field of tangent-space identity maps on a 3-dimensional manifold:

```

sage: M = Manifold(3, 'M', start_index=1)
sage: c_xyz.<x,y,z> = M.chart()
sage: a = M.tangent_identity_field(); a
Field of tangent-space identity maps on the 3-dimensional
differentiable manifold M
sage: a.comp()
Kronecker delta of size 3x3

```

**See also:**

For more examples, see *AutomorphismField*.

**tangent\_space** (*point*)

Tangent space to *self* at a given point.

INPUT:

- `point` – *ManifoldPoint*; point  $p$  on the manifold

OUTPUT:

- *TangentSpace* representing the tangent vector space  $T_p M$ , where  $M$  is the current manifold

EXAMPLES:

A tangent space to a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: p = M.point((2, -3), name='p')
sage: Tp = M.tangent_space(p); Tp
Tangent space at Point p on the 2-dimensional differentiable
manifold M
sage: Tp.category()
Category of finite dimensional vector spaces over Symbolic Ring
sage: dim(Tp)
2
```

**See also:**

*TangentSpace* for more examples.

**tensor\_field** ( $k, l, name=None, latex_name=None, sym=None, antisym=None, dest_map=None$ )

Define a tensor field on `self`.

Via the argument `dest_map`, it is possible to let the tensor field take its values on another manifold. More precisely, if  $M$  is the current manifold,  $N$  a differentiable manifold,  $\Phi : M \rightarrow N$  a differentiable map and  $(k, l)$  a pair of non-negative integers, a *tensor field of type  $(k, l)$  along  $M$  with values on  $N$*  is a differentiable map

$$t : M \longrightarrow T^{(k,l)}N$$

( $T^{(k,l)}N$  being the tensor bundle of type  $(k, l)$  over  $N$ ) such that

$$\forall p \in M, t(p) \in T^{(k,l)}(T_{\Phi(p)}N),$$

where  $T^{(k,l)}(T_{\Phi(p)}N)$  is the space of tensors of type  $(k, l)$  on the tangent space  $T_{\Phi(p)}N$ .

The standard case of tensor fields *on*  $M$  corresponds to  $N = M$  and  $\Phi = \text{Id}_M$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $N$  ( $M$  is then an open interval of  $\mathbf{R}$ ).

See *TensorField* for a complete documentation.

INPUT:

- `k` – the contravariant rank  $k$ , the tensor type being  $(k, l)$
- `l` – the covariant rank  $l$ , the tensor type being  $(k, l)$
- `name` – (default: `None`) name given to the tensor field
- `latex_name` – (default: `None`) LaTeX symbol to denote the tensor field; if `None`, the LaTeX symbol is set to `name`
- `sym` – (default: `None`) a symmetry or a list of symmetries among the tensor arguments: each symmetry is described by a tuple containing the positions of the involved arguments, with the convention `position=0` for the first argument; for instance:
  - `sym = (0, 1)` for a symmetry between the 1st and 2nd arguments

–`sym` =  $[(0, 2), (1, 3, 4)]$  for a symmetry between the 1st and 3rd arguments and a symmetry between the 2nd, 4th and 5th arguments

- `antisym` – (default: `None`) antisymmetry or list of antisymmetries among the arguments, with the same convention as for `sym`
- `dest_map` – (default: `None`) the destination map  $\Phi : M \rightarrow N$ ; if `None`, it is assumed that  $N = M$  and that  $\Phi$  is the identity map (case of a tensor field *on*  $M$ ), otherwise `dest_map` must be a *DiffMap*

OUTPUT:

- a *TensorField* (or if  $N$  is parallelizable, a *TensorFieldParal*) representing the defined tensor field

EXAMPLES:

A tensor field of type  $(2, 0)$  on an open subset of a 3-dimensional differentiable manifold:

```
sage: M = Manifold(3, 'M')
sage: U = M.open_subset('U')
sage: c_xyz.<x,y,z> = U.chart()
sage: t = U.tensor_field(2, 0, 'T'); t
Tensor field T of type (2,0) on the Open subset U of the
3-dimensional differentiable manifold M
```

The type  $(2, 0)$  tensor fields on  $U$  form the set  $\mathcal{T}^{(2,0)}(U)$ , which is a module over the algebra  $C^k(U)$  of differentiable scalar fields on  $U$ :

```
sage: t.parent()
Free module T^(2,0)(U) of type-(2,0) tensors fields on the Open
subset U of the 3-dimensional differentiable manifold M
sage: t in U.tensor_field_module((2,0))
True
```

**See also:**

For more examples, see *TensorField*.

**tensor\_field\_module** (*tensor\_type*, *dest\_map=None*)

Return the set of tensor fields of a given type defined on `self`, possibly with values in another manifold, as a module over the algebra of scalar fields defined on `self`.

**See also:**

*TensorFieldModule* for complete documentation.

INPUT:

- `tensor_type` – pair  $(k, l)$  with  $k$  being the contravariant rank and  $l$  the covariant rank
- `dest_map` – (default: `None`) destination map, i.e. a differentiable map  $\Phi : M \rightarrow N$ , where  $M$  is the current manifold and  $N$  a differentiable manifold; if `None`, it is assumed that  $N = M$  and that  $\Phi$  is the identity map (case of tensor fields *on*  $M$ ), otherwise `dest_map` must be a *DiffMap*

OUTPUT:

- a *TensorFieldModule* (or if  $N$  is parallelizable, a *TensorFieldFreeModule*) representing the module  $\mathcal{T}^{(k,l)}(M, \Phi)$  of type- $(k, l)$  tensor fields on  $M$  taking values on  $\Phi(M) \subset M$

EXAMPLES:

Module of type- $(2, 1)$  tensor fields on a 3-dimensional open subset of a differentiable manifold:

```

sage: M = Manifold(3, 'M')
sage: U = M.open_subset('U')
sage: c_xyz.<x,y,z> = U.chart()
sage: TU = U.tensor_field_module((2,1)) ; TU
Free module T^(2,1)(U) of type-(2,1) tensors fields on the Open
subset U of the 3-dimensional differentiable manifold M
sage: TU.category()
Category of finite dimensional modules over Algebra of
differentiable scalar fields on the Open subset U of the
3-dimensional differentiable manifold M
sage: TU.base_ring()
Algebra of differentiable scalar fields on the Open subset U of
the 3-dimensional differentiable manifold M
sage: TU.base_ring() is U.scalar_field_algebra()
True
sage: TU.an_element()
Tensor field of type (2,1) on the Open subset U of the
3-dimensional differentiable manifold M
sage: TU.an_element().display()
2 d/dx*d/dx*d/dx

```

**vector\_field** (*name=None, latex\_name=None, dest\_map=None*)

Define a vector field on self.

Via the argument `dest_map`, it is possible to let the vector field take its values on another manifold. More precisely, if  $M$  is the current manifold,  $N$  a differentiable manifold and  $\Phi : M \rightarrow N$  a differentiable map, a *vector field along  $M$  with values on  $N$*  is a differentiable map

$$v : M \longrightarrow TN$$

( $TN$  being the tangent bundle of  $N$ ) such that

$$\forall p \in M, v(p) \in T_{\Phi(p)}N,$$

where  $T_{\Phi(p)}N$  is the tangent space to  $N$  at the point  $\Phi(p)$ .

The standard case of vector fields *on*  $M$  corresponds to  $N = M$  and  $\Phi = \text{Id}_M$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $N$  ( $M$  is then an open interval of  $\mathbf{R}$ ).

See *VectorField* for a complete documentation.

INPUT:

- `name` – (default: None) name given to the vector field
- `latex_name` – (default: None) LaTeX symbol to denote the vector field; if none is provided, the LaTeX symbol is set to `name`
- `dest_map` – (default: None) the destination map  $\Phi : M \rightarrow N$ ; if None, it is assumed that  $N = M$  and that  $\Phi$  is the identity map (case of a vector field *on*  $M$ ), otherwise `dest_map` must be a *DiffMap*

OUTPUT:

- a *VectorField* (or if  $N$  is parallelizable, a *VectorFieldParal*) representing the defined vector field

EXAMPLES:

A vector field on a open subset of a 3-dimensional differentiable manifold:

```

sage: M = Manifold(3, 'M')
sage: U = M.open_subset('U')
sage: c_xyz.<x,y,z> = U.chart()
sage: v = U.vector_field('v'); v
Vector field v on the Open subset U of the 3-dimensional
differentiable manifold M

```

The vector fields on  $U$  form the set  $\mathcal{X}(U)$ , which is a module over the algebra  $C^k(U)$  of differentiable scalar fields on  $U$ :

```

sage: v.parent()
Free module X(U) of vector fields on the Open subset U of the
3-dimensional differentiable manifold M
sage: v in U.vector_field_module()
True

```

**See also:**

For more examples, see [VectorField](#).

**vector\_field\_module** (*dest\_map=None, force\_free=False*)

Return the set of vector fields defined on `self`, possibly with values in another differentiable manifold, as a module over the algebra of scalar fields defined on the manifold.

See [VectorFieldModule](#) for a complete documentation.

**INPUT:**

- `dest_map` – (default: `None`) destination map, i.e. a differentiable map  $\Phi : M \rightarrow N$ , where  $M$  is the current manifold and  $N$  a differentiable manifold; if `None`, it is assumed that  $N = M$  and that  $\Phi$  is the identity map (case of vector fields *on*  $M$ ), otherwise `dest_map` must be a [DiffMap](#)
- `force_free` – (default: `False`) if set to `True`, force the construction of a *free* module (this implies that  $N$  is parallelizable)

**OUTPUT:**

- a [VectorFieldModule](#) (or if  $N$  is parallelizable, a [VectorFieldFreeModule](#)) representing the module  $\mathcal{X}(M, \Phi)$  of vector fields on  $M$  taking values on  $\Phi(M) \subset N$

**EXAMPLES:**

Vector field module  $\mathcal{X}(U) := \mathcal{X}(U, \text{Id}_U)$  of the complement  $U$  of the two poles on the sphere  $\mathbb{S}^2$ :

```

sage: S2 = Manifold(2, 'S^2')
sage: U = S2.open_subset('U') # the complement of the two poles
sage: spher_coord.<th,ph> = U.chart(r'th:(0,pi):\theta ph:(0,2*pi):\phi') #_
↪ spherical coordinates
sage: XU = U.vector_field_module() ; XU
Free module X(U) of vector fields on the Open subset U of
the 2-dimensional differentiable manifold S^2
sage: XU.category()
Category of finite dimensional modules over Algebra of
differentiable scalar fields on the Open subset U of
the 2-dimensional differentiable manifold S^2
sage: XU.base_ring()
Algebra of differentiable scalar fields on the Open subset U of
the 2-dimensional differentiable manifold S^2
sage: XU.base_ring() is U.scalar_field_algebra()
True

```

$\mathcal{X}(U)$  is a free module because  $U$  is parallelizable (being a chart domain):

```
sage: U.is_manifestly_parallelizable()
True
```

Its rank is the manifold's dimension:

```
sage: XU.rank()
2
```

The elements of  $\mathcal{X}(U)$  are vector fields on  $U$ :

```
sage: XU.an_element()
Vector field on the Open subset U of the 2-dimensional
differentiable manifold S^2
sage: XU.an_element().display()
2 d/dth + 2 d/dph
```

Vector field module  $\mathcal{X}(U, \Phi)$  of the  $\mathbf{R}^3$ -valued vector fields along  $U$ , associated with the embedding  $\Phi$  of  $\mathbb{S}^2$  into  $\mathbf{R}^3$ :

```
sage: R3 = Manifold(3, 'R^3')
sage: cart_coord.<x, y, z> = R3.chart()
sage: Phi = U.diff_map(R3,
...:      [sin(th)*cos(ph), sin(th)*sin(ph), cos(th)], name='Phi')
sage: XU_R3 = U.vector_field_module(dest_map=Phi) ; XU_R3
Free module X(U,Phi) of vector fields along the Open subset U of
the 2-dimensional differentiable manifold S^2 mapped into the
3-dimensional differentiable manifold R^3
sage: XU_R3.base_ring()
Algebra of differentiable scalar fields on the Open subset U of the
2-dimensional differentiable manifold S^2
```

$\mathcal{X}(U, \Phi)$  is a free module because  $\mathbf{R}^3$  is parallelizable and its rank is 3:

```
sage: XU_R3.rank()
3
```

**vector\_frame** (*symbol=None, latex\_symbol=None, dest\_map=None, from\_frame=None*)

Define a vector frame on self.

A *vector frame* is a field on the manifold that provides, at each point  $p$  of the manifold, a vector basis of the tangent space at  $p$ .

**See also:**

[VectorFrame](#) for complete documentation.

INPUT:

- *symbol* – (default: None) a letter (of a few letters) to denote a generic vector of the frame; can be set to None if the parameter *from\_frame* is filled
- *latex\_symbol* – (default: None) symbol to denote a generic vector of the frame; if None, the value of *symbol* is used.
- *dest\_map* – (default: None) *DiffMap*; destination map  $\Phi : U \rightarrow V$ ; if None, the identity is assumed (case of a vector frame on  $U$ )
- *from\_frame* – (default: None) vector frame  $\tilde{e}$  on the codomain  $V$  of the destination map  $\Phi$ ; the returned frame  $e$  is then such that for all  $p \in U$ , we have  $e(p) = \tilde{e}(\Phi(p))$

OUTPUT:

- a *VectorFrame* representing the defined vector frame

EXAMPLES:

Setting a vector frame on a 3-dimensional open subset:

```
sage: M = Manifold(3, 'M')
sage: A = M.open_subset('A', latex_name=r'\mathcal{A}'); A
Open subset A of the 3-dimensional differentiable manifold M
sage: c_xyz.<x,y,z> = A.chart()
sage: e = A.vector_frame('e'); e
Vector frame (A, (e_0,e_1,e_2))
sage: e[0]
Vector field e_0 on the Open subset A of the 3-dimensional
differentiable manifold M
```

**See also:**

For more examples, see *VectorFrame*.

## 2.2 Coordinate Charts on Differentiable Manifolds

The class *DiffChart* implements coordinate charts on a differentiable manifold over a topological field  $K$  (in most applications,  $K = \mathbf{R}$  or  $K = \mathbf{C}$ ).

The subclass *RealDiffChart* is devoted to the case  $K = \mathbf{R}$ , for which the concept of coordinate range is meaningful. Moreover, *RealDiffChart* is endowed with some plotting capabilities (cf. method *plot()*).

Transition maps between charts are implemented via the class *DiffCoordChange*.

AUTHORS:

- Ericourgoulhon, Michal Bejger (2013-2015) : initial version

REFERENCES:

- Chap. 1 of [Lee2013]

```
class sage.manifolds.differentiable.chart.DiffChart (domain, coordinates='',
                                                    names=None)
```

Bases: *sage.manifolds.chart.Chart*

Chart on a differentiable manifold.

Given a differentiable manifold  $M$  of dimension  $n$  over a topological field  $K$ , a *chart* is a member  $(U, \varphi)$  of the manifold's differentiable atlas;  $U$  is then an open subset of  $M$  and  $\varphi : U \rightarrow V \subset K^n$  is a homeomorphism from  $U$  to an open subset  $V$  of  $K^n$ .

The components  $(x^1, \dots, x^n)$  of  $\varphi$ , defined by  $\varphi(p) = (x^1(p), \dots, x^n(p)) \in K^n$  for any point  $p \in U$ , are called the *coordinates* of the chart  $(U, \varphi)$ .

INPUT:

- *domain* – open subset  $U$  on which the chart is defined
- *coordinates* – (default: '' (empty string)) single string defining the coordinate symbols, with ' ' (whitespace) as a separator; each item has at most two fields, separated by ':':
  1. The coordinate symbol (a letter or a few letters)

2.(optional) The LaTeX spelling of the coordinate; if not provided the coordinate symbol given in the first field will be used.

If it contains any LaTeX expression, the string `coordinates` must be declared with the prefix ‘r’ (for “raw”) to allow for a proper treatment of LaTeX’s backslash character (see examples below). If no LaTeX spelling is to be set for any coordinate, the argument `coordinates` can be omitted when the shortcut operator `<, >` is used via Sage parser (see examples below)

•`names` – (default: None) unused argument, except if `coordinates` is not provided; it must then be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator `<, >` is used).

#### EXAMPLES:

A chart on a complex 2-dimensional differentiable manifold:

```
sage: M = Manifold(2, 'M', field='complex')
sage: X = M.chart('x y'); X
Chart (M, (x, y))
sage: latex(X)
\left(M, (x, y)\right)
sage: type(X)
<class 'sage.manifolds.differentiable.chart.DiffChart'>
```

To manipulate the coordinates  $(x, y)$  as global variables, one has to set:

```
sage: x, y = X[:]
```

However, a shortcut is to use the declarator `<x, y>` in the left-hand side of the chart declaration (there is then no need to pass the string `'x y'` to `chart()`):

```
sage: M = Manifold(2, 'M', field='complex')
sage: X.<x,y> = M.chart(); X
Chart (M, (x, y))
```

The coordinates are then immediately accessible:

```
sage: y
y
sage: x is X[0] and y is X[1]
True
```

The trick is performed by Sage parser:

```
sage: preparse("X.<x,y> = M.chart()")
"X = M.chart(names=('x', 'y')); (x, y) = X._first_ngens(2)"
```

Note that `x` and `y` declared in `<x, y>` are mere Python variable names and do not have to coincide with the coordinate symbols; for instance, one may write:

```
sage: M = Manifold(2, 'M', field='complex')
sage: X.<x1,y1> = M.chart('x y'); X
Chart (M, (x, y))
```

Then `y` is not known as a global Python variable and the coordinate `y` is accessible only through the global variable `y1`:

```
sage: y1
y
sage: latex(y1)
```

```
Y
sage: y1 is X[1]
True
```

However, having the name of the Python variable coincide with the coordinate symbol is quite convenient; so it is recommended to declare:

```
sage: M = Manifold(2, 'M', field='complex')
sage: X.<x,y> = M.chart()
```

In the above example, the chart X covers entirely the manifold M:

```
sage: X.domain()
2-dimensional complex manifold M
```

Of course, one may declare a chart only on an open subset of M:

```
sage: U = M.open_subset('U')
sage: Y.<z1, z2> = U.chart(r'z1:\zeta_1 z2:\zeta_2'); Y
Chart (U, (z1, z2))
sage: Y.domain()
Open subset U of the 2-dimensional complex manifold M
```

In the above declaration, we have also specified some LaTeX writing of the coordinates different from the text one:

```
sage: latex(z1)
{\zeta_1}
```

Note the prefix `r` in front of the string `r'z1:\zeta_1 z2:\zeta_2'`; it makes sure that the backslash character is treated as an ordinary character, to be passed to the LaTeX interpreter.

Coordinates are Sage symbolic variables (see `sage.symbolic.expression`):

```
sage: type(z1)
<type 'sage.symbolic.expression.Expression'>
```

In addition to the Python variable name provided in the operator `<., .>`, the coordinates are accessible by their indices:

```
sage: Y[0], Y[1]
(z1, z2)
```

The index range is that declared during the creation of the manifold. By default, it starts at 0, but this can be changed via the parameter `start_index`:

```
sage: M1 = Manifold(2, 'M_1', field='complex', start_index=1)
sage: Z.<u,v> = M1.chart()
sage: Z[1], Z[2]
(u, v)
```

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

```
sage: Y[: ]
(z1, z2)
```

Each constructed chart is automatically added to the manifold's user atlas:

```
sage: M.atlas()
[Chart (M, (x, y)), Chart (U, (z1, z2))]
```

and to the atlas of the chart's domain:

```
sage: U.atlas()
[Chart (U, (z1, z2))]
```

Manifold subsets have a *default chart*, which, unless changed via the method `set_default_chart()`, is the first defined chart on the subset (or on an open subset of it):

```
sage: M.default_chart()
Chart (M, (x, y))
sage: U.default_chart()
Chart (U, (z1, z2))
```

The default charts are not privileged charts on the manifold, but rather charts whose name can be skipped in the argument list of functions having an optional `chart=` argument.

The action of the chart map  $\varphi$  on a point is obtained by means of the call operator, i.e. the operator `()`:

```
sage: p = M.point((1+i, 2), chart=X); p
Point on the 2-dimensional complex manifold M
sage: X(p)
(I + 1, 2)
sage: X(p) == p.coord(X)
True
```

A vector frame is naturally associated to each chart:

```
sage: X.frame()
Coordinate frame (M, (d/dx,d/dy))
sage: Y.frame()
Coordinate frame (U, (d/dz1,d/dz2))
```

as well as a dual frame (basis of 1-forms):

```
sage: X.coframe()
Coordinate coframe (M, (dx,dy))
sage: Y.coframe()
Coordinate coframe (U, (dz1,dz2))
```

**See also:**

[RealDiffChart](#) for charts on differentiable manifolds over  $\mathbf{R}$ .

**coframe()**

Return the coframe (basis of coordinate differentials) associated with `self`.

OUTPUT:

- a `CoordCoFrame` representing the coframe

EXAMPLES:

Coordinate coframe associated with some chart on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart()
sage: c_xy.coframe()
```

```
Coordinate coframe (M, (dx,dy))
sage: type(c_xy.coframe())
<class 'sage.manifolds.differentiable.vectorframe.CoordCoFrame'>
```

Check that `c_xy.coframe()` is indeed the coordinate coframe associated with the coordinates  $(x, y)$ :

```
sage: dx = c_xy.coframe()[0] ; dx
1-form dx on the 2-dimensional differentiable manifold M
sage: dy = c_xy.coframe()[1] ; dy
1-form dy on the 2-dimensional differentiable manifold M
sage: ex = c_xy.frame()[0] ; ex
Vector field d/dx on the 2-dimensional differentiable manifold M
sage: ey = c_xy.frame()[1] ; ey
Vector field d/dy on the 2-dimensional differentiable manifold M
sage: dx(ex).display()
dx(d/dx): M --> R
(x, y) |--> 1
sage: dx(ey).display()
dx(d/dy): M --> R
(x, y) |--> 0
sage: dy(ex).display()
dy(d/dx): M --> R
(x, y) |--> 0
sage: dy(ey).display()
dy(d/dy): M --> R
(x, y) |--> 1
```

### `frame()`

Return the vector frame (coordinate frame) associated with `self`.

OUTPUT:

- a *CoordFrame* representing the coordinate frame

EXAMPLES:

Coordinate frame associated with some chart on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart()
sage: c_xy.frame()
Coordinate frame (M, (d/dx,d/dy))
sage: type(c_xy.frame())
<class 'sage.manifolds.differentiable.vectorframe.CoordFrame'>
```

Check that `c_xy.frame()` is indeed the coordinate frame associated with the coordinates  $(x, y)$ :

```
sage: ex = c_xy.frame()[0] ; ex
Vector field d/dx on the 2-dimensional differentiable manifold M
sage: ey = c_xy.frame()[1] ; ey
Vector field d/dy on the 2-dimensional differentiable manifold M
sage: ex(M.scalar_field(x)).display()
M --> R
(x, y) |--> 1
sage: ex(M.scalar_field(y)).display()
M --> R
(x, y) |--> 0
sage: ey(M.scalar_field(x)).display()
M --> R
```

```
(x, y) |--> 0
sage: ey(M.scalar_field(y)).display()
M --> R
(x, y) |--> 1
```

**restrict** (*subset*, *restrictions=None*)

Return the restriction of `self` to some subset.

If the current chart is  $(U, \varphi)$ , a *restriction* (or *subchart*) is a chart  $(V, \psi)$  such that  $V \subset U$  and  $\psi = \varphi|_V$ .

If such subchart has not been defined yet, it is constructed here.

The coordinates of the subchart bare the same names as the coordinates of the original chart.

INPUT:

- *subset* – open subset  $V$  of the chart domain  $U$
- *restrictions* – (default: `None`) list of coordinate restrictions defining the subset  $V$

A restriction can be any symbolic equality or inequality involving the coordinates, such as  $x > y$  or  $x^2 + y^2 \neq 0$ . The items of the list *restrictions* are combined with the `and` operator; if some restrictions are to be combined with the `or` operator instead, they have to be passed as a tuple in some single item of the list *restrictions*. For example:

```
restrictions = [x > y, (x != 0, y != 0), z^2 < x]
```

means  $(x > y)$  and  $((x \neq 0) \text{ or } (y \neq 0))$  and  $(z^2 < x)$ . If the list *restrictions* contains only one item, this item can be passed as such, i.e. writing  $x > y$  instead of the single element list  $[x > y]$ .

OUTPUT:

- a *DiffChart*  $(V, \psi)$

EXAMPLES:

Coordinates on the unit open ball of  $\mathbb{C}^2$  as a subchart of the global coordinates of  $\mathbb{C}^2$ :

```
sage: M = Manifold(2, 'C^2', field='complex')
sage: X.<z1, z2> = M.chart()
sage: B = M.open_subset('B')
sage: X_B = X.restrict(B, abs(z1)^2 + abs(z2)^2 < 1); X_B
Chart (B, (z1, z2))
```

**transition\_map** (*other*, *transformations*, *intersection\_name=None*, *restrictions1=None*, *restrictions2=None*)

Construct the transition map between the current chart,  $(U, \varphi)$  say, and another one,  $(V, \psi)$  say.

If  $n$  is the manifold's dimension, the *transition map* is the map

$$\psi \circ \varphi^{-1} : \varphi(U \cap V) \subset K^n \rightarrow \psi(U \cap V) \subset K^n,$$

where  $K$  is the manifold's base field. In other words, the transition map expresses the coordinates  $(y^1, \dots, y^n)$  of  $(V, \psi)$  in terms of the coordinates  $(x^1, \dots, x^n)$  of  $(U, \varphi)$  on the open subset where the two charts intersect, i.e. on  $U \cap V$ .

By definition, the transition map  $\psi \circ \varphi^{-1}$  must be of classe  $C^k$ , where  $k$  is the degree of differentiability of the manifold (cf. `diff_degree()`).

INPUT:

- *other* – the chart  $(V, \psi)$

- `transformations` – tuple (or list)  $(Y_1, \dots, Y_2)$ , where  $Y_i$  is the symbolic expression of the coordinate  $y^i$  in terms of the coordinates  $(x^1, \dots, x^n)$
- `intersection_name` – (default: None) name to be given to the subset  $U \cap V$  if the latter differs from  $U$  or  $V$
- `restrictions1` – (default: None) list of conditions on the coordinates of the current chart that define  $U \cap V$  if the latter differs from  $U$ . `restrictions1` must be a list of symbolic equalities or inequalities involving the coordinates, such as  $x > y$  or  $x^2 + y^2 \neq 0$ . The items of the list `restrictions1` are combined with the `and` operator; if some restrictions are to be combined with the `or` operator instead, they have to be passed as a tuple in some single item of the list `restrictions1`. For example, `restrictions1 = [x > y, (x != 0, y != 0), z^2 < x]` means  $(x > y)$  and  $((x \neq 0) \text{ or } (y \neq 0))$  and  $(z^2 < x)$ . If the list `restrictions1` contains only one item, this item can be passed as such, i.e. writing  $x > y$  instead of the single-element list `[x > y]`.
- `restrictions2` – (default: None) list of conditions on the coordinates of the chart  $(V, \psi)$  that define  $U \cap V$  if the latter differs from  $V$  (see `restrictions1` for the syntax)

OUTPUT:

- The transition map  $\psi \circ \varphi^{-1}$  defined on  $U \cap V$ , as an instance of `DiffCoordChange`.

EXAMPLES:

Transition map between two stereographic charts on the circle  $S^1$ :

```
sage: M = Manifold(1, 'S^1')
sage: U = M.open_subset('U') # Complement of the North pole
sage: cU.<x> = U.chart() # Stereographic chart from the North pole
sage: V = M.open_subset('V') # Complement of the South pole
sage: cV.<y> = V.chart() # Stereographic chart from the South pole
sage: M.declare_union(U,V) # S^1 is the union of U and V
sage: trans = cU.transition_map(cV, 1/x, intersection_name='W',
....:                          restrictions1= x!=0, restrictions2 = y!=0)
sage: trans
Change of coordinates from Chart (W, (x,)) to Chart (W, (y,))
sage: trans.display()
y = 1/x
```

The subset  $W$ , intersection of  $U$  and  $V$ , has been created by `transition_map()`:

```
sage: M.list_of_subsets()
[1-dimensional differentiable manifold S^1,
 Open subset U of the 1-dimensional differentiable manifold S^1,
 Open subset V of the 1-dimensional differentiable manifold S^1,
 Open subset W of the 1-dimensional differentiable manifold S^1]
sage: W = M.list_of_subsets()[3]
sage: W is U.intersection(V)
True
sage: M.atlas()
[Chart (U, (x,)), Chart (V, (y,)), Chart (W, (x,)), Chart (W, (y,))]
```

Transition map between the polar chart and the Cartesian one on  $\mathbf{R}^2$ :

```
sage: M = Manifold(2, 'R^2')
sage: c_cart.<x,y> = M.chart()
sage: U = M.open_subset('U') # the complement of the half line {y=0, x >= 0}
sage: c_spher.<r,phi> = U.chart(r'r:(0,+oo) phi:(0,2*pi):\phi')
sage: trans = c_spher.transition_map(c_cart, (r*cos(phi), r*sin(phi)),
....:                               restrictions2=(y!=0, x<0))
```

```
sage: trans
Change of coordinates from Chart (U, (r, phi)) to Chart (U, (x, y))
sage: trans.display()
x = r*cos(phi)
y = r*sin(phi)
```

In this case, no new subset has been created since  $U \cap M = U$ :

```
sage: M.list_of_subsets()
[2-dimensional differentiable manifold R^2,
 Open subset U of the 2-dimensional differentiable manifold R^2]
```

but a new chart has been created:  $(U, (x, y))$ :

```
sage: M.atlas()
[Chart (R^2, (x, y)), Chart (U, (r, phi)), Chart (U, (x, y))]
```

**class** sage.manifolds.differentiable.chart.**DiffCoordChange** (*chart1*, *chart2*, \**transformations*)

Bases: *sage.manifolds.chart.CoordChange*

Transition map between two charts of a differentiable manifold.

Giving two coordinate charts  $(U, \varphi)$  and  $(V, \psi)$  on a differentiable manifold  $M$  of dimension  $n$  over a topological field  $K$ , the *transition map from  $(U, \varphi)$  to  $(V, \psi)$*  is the map

$$\psi \circ \varphi^{-1} : \varphi(U \cap V) \subset K^n \rightarrow \psi(U \cap V) \subset K^n,$$

In other words, the transition map  $\psi \circ \varphi^{-1}$  expresses the coordinates  $(y^1, \dots, y^n)$  of  $(V, \psi)$  in terms of the coordinates  $(x^1, \dots, x^n)$  of  $(U, \varphi)$  on the open subset where the two charts intersect, i.e. on  $U \cap V$ .

By definition, the transition map  $\psi \circ \varphi^{-1}$  must be of class  $C^k$ , where  $k$  is the degree of differentiability of the manifold (cf. *diff\_degree()*).

INPUT:

- *chart1* – chart  $(U, \varphi)$
- *chart2* – chart  $(V, \psi)$
- *transformations* – tuple (or list)  $(Y_1, \dots, Y_2)$ , where  $Y_i$  is the symbolic expression of the coordinate  $y^i$  in terms of the coordinates  $(x^1, \dots, x^n)$

EXAMPLES:

Transition map on a 2-dimensional differentiable manifold:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: Y.<u,v> = M.chart()
sage: X_to_Y = X.transition_map(Y, [x+y, x-y])
sage: X_to_Y
Change of coordinates from Chart (M, (x, y)) to Chart (M, (u, v))
sage: type(X_to_Y)
<class 'sage.manifolds.differentiable.chart.DiffCoordChange'>
sage: X_to_Y.display()
u = x + y
v = x - y
```

**jacobian()**

Return the Jacobian matrix of self.

If `self` corresponds to the change of coordinates

$$y^i = Y^i(x^1, \dots, x^n) \quad 1 \leq i \leq n$$

the Jacobian matrix  $J$  is given by

$$J_{ij} = \frac{\partial Y^i}{\partial x^j}$$

where  $i$  is the row index and  $j$  the column one.

OUTPUT:

- Jacobian matrix  $J$ , the elements  $J_{ij}$  of which being coordinate functions (cf. *CoordFunction*)

EXAMPLES:

Jacobian matrix of a 2-dimensional transition map:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: Y.<u,v> = M.chart()
sage: X_to_Y = X.transition_map(Y, [x+y^2, 3*x-y])
sage: X_to_Y.jacobian()
[ 1 2*y]
[ 3 -1]
```

Each element of the Jacobian matrix is a coordinate function:

```
sage: parent(X_to_Y.jacobian()[0,0])
Ring of coordinate functions on Chart (M, (x, y))
```

**jacobian\_det()**

Return the Jacobian determinant of `self`.

The Jacobian determinant is the determinant of the Jacobian matrix (see *jacobian()*).

OUTPUT:

- determinant of the Jacobian matrix  $J$  as a coordinate function (cf. *CoordFunction*)

EXAMPLES:

Jacobian determinant of a 2-dimensional transition map:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: Y.<u,v> = M.chart()
sage: X_to_Y = X.transition_map(Y, [x+y^2, 3*x-y])
sage: X_to_Y.jacobian_det()
-6*y - 1
sage: X_to_Y.jacobian_det() == det(X_to_Y.jacobian())
True
```

The Jacobian determinant is a coordinate function:

```
sage: parent(X_to_Y.jacobian_det())
Ring of coordinate functions on Chart (M, (x, y))
```

```
class sage.manifolds.differentiable.chart.RealDiffChart (domain, coordinates='',
                                                         names=None)
Bases: sage.manifolds.differentiable.chart.DiffChart, sage.manifolds.chart.RealChart
```

Chart on a differentiable manifold over  $\mathbf{R}$ .

Given a differentiable manifold  $M$  of dimension  $n$  over  $\mathbf{R}$ , a *chart* is a member  $(U, \varphi)$  of the manifold's differentiable atlas;  $U$  is then an open subset of  $M$  and  $\varphi : U \rightarrow V \subset \mathbf{R}^n$  is a homeomorphism from  $U$  to an open subset  $V$  of  $\mathbf{R}^n$ .

The components  $(x^1, \dots, x^n)$  of  $\varphi$ , defined by  $\varphi(p) = (x^1(p), \dots, x^n(p)) \in \mathbf{R}^n$  for any point  $p \in U$ , are called the *coordinates* of the chart  $(U, \varphi)$ .

INPUT:

- `domain` – open subset  $U$  on which the chart is defined
- `coordinates` – (default: '' (empty string)) single string defining the coordinate symbols and ranges, with ' ' (whitespace) as a separator; each item has at most three fields, separated by ':':
  1. The coordinate symbol (a letter or a few letters)
  2. (optional) The interval  $I$  defining the coordinate range: if not provided, the coordinate is assumed to span all  $\mathbf{R}$ ; otherwise  $I$  must be provided in the form  $(a, b)$  (or equivalently  $]a, b[$ ). The bounds  $a$  and  $b$  can be `+/-Infinity`, `Inf`, `infinity`, `inf` or `oo`. For *singular* coordinates, non-open intervals such as  $[a, b]$  and  $(a, b]$  (or equivalently  $]a, b]$ ) are allowed. Note that the interval declaration must not contain any whitespace.
  3. (optional) The LaTeX spelling of the coordinate; if not provided the coordinate symbol given in the first field will be used.

The order of the fields 2 and 3 does not matter and each of them can be omitted. If it contains any LaTeX expression, the string `coordinates` must be declared with the prefix 'r' (for "raw") to allow for a proper treatment of LaTeX backslash characters (see examples below). If no interval range and no LaTeX spelling is to be set for any coordinate, the argument `coordinates` can be omitted when the shortcut operator `<, >` is used via Sage parser (see examples below)

- `names` – (default: None) unused argument, except if `coordinates` is not provided; it must then be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator `<, >` is used).

EXAMPLES:

Cartesian coordinates on  $\mathbf{R}^3$ :

```
sage: M = Manifold(3, 'R^3', r'\RR^3', start_index=1)
sage: c_cart = M.chart('x y z'); c_cart
Chart (R^3, (x, y, z))
sage: type(c_cart)
<class 'sage.manifolds.differentiable.chart.RealDiffChart'>
```

To have the coordinates accessible as global variables, one has to set:

```
sage: (x, y, z) = c_cart[:]
```

However, a shortcut is to use the declarator `<x, y, z>` in the left-hand side of the chart declaration (there is then no need to pass the string 'x y z' to `chart()`):

```
sage: M = Manifold(3, 'R^3', r'\RR^3', start_index=1)
sage: c_cart.<x,y,z> = M.chart(); c_cart
Chart (R^3, (x, y, z))
```

The coordinates are then immediately accessible:

```
sage: y
y
```

```
sage: y is c_cart[2]
True
```

The trick is performed by Sage preparser:

```
sage: prepare("c_cart.<x,y,z> = M.chart()")
"c_cart = M.chart(names=('x', 'y', 'z',)); (x, y, z) = c_cart._first_ngens(3)"
```

Note that  $x$ ,  $y$ ,  $z$  declared in  $\langle x, y, z \rangle$  are mere Python variable names and do not have to coincide with the coordinate symbols; for instance, one may write:

```
sage: M = Manifold(3, 'R^3', r'\RR^3', start_index=1)
sage: c_cart.<x1,y1,z1> = M.chart('x y z'); c_cart
Chart (R^3, (x, y, z))
```

Then  $y$  is not known as a global variable and the coordinate  $y$  is accessible only through the global variable  $y1$ :

```
sage: y1
y
sage: y1 is c_cart[2]
True
```

However, having the name of the Python variable coincide with the coordinate symbol is quite convenient; so it is recommended to declare:

```
sage: forget() # for doctests only
sage: M = Manifold(3, 'R^3', r'\RR^3', start_index=1)
sage: c_cart.<x,y,z> = M.chart()
```

Spherical coordinates on the subset  $U$  of  $\mathbf{R}^3$  that is the complement of the half-plane  $\{y = 0, x \geq 0\}$ :

```
sage: U = M.open_subset('U')
sage: c_spher.<r,th,ph> = U.chart(r'r:(0,+oo) th:(0,pi):\theta ph:(0,2*pi):\phi')
sage: c_spher
Chart (U, (r, th, ph))
```

Note the prefix 'r' for the string defining the coordinates in the arguments of `chart`.

Coordinates are Sage symbolic variables (see `sage.symbolic.expression`):

```
sage: type(th)
<type 'sage.symbolic.expression.Expression'>
sage: latex(th)
{\theta}
sage: assumptions(th)
[th is real, th > 0, th < pi]
```

Coordinate are also accessible by their indices:

```
sage: x1 = c_spher[1]; x2 = c_spher[2]; x3 = c_spher[3]
sage: [x1, x2, x3]
[r, th, ph]
sage: (x1, x2, x3) == (r, th, ph)
True
```

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

```
sage: c_cart[:]
(x, y, z)
sage: c_spher[:]
(r, th, ph)
```

Let us check that the declared coordinate ranges have been taken into account:

```
sage: c_cart.coord_range()
x: (-oo, +oo); y: (-oo, +oo); z: (-oo, +oo)
sage: c_spher.coord_range()
r: (0, +oo); th: (0, pi); ph: (0, 2*pi)
sage: bool(th>0 and th<pi)
True
sage: assumptions() # list all current symbolic assumptions
[x is real, y is real, z is real, r is real, r > 0, th is real,
 th > 0, th < pi, ph is real, ph > 0, ph < 2*pi]
```

The coordinate ranges are used for simplifications:

```
sage: simplify(abs(r)) # r has been declared to lie in the interval (0,+oo)
r
sage: simplify(abs(x)) # no positive range has been declared for x
abs(x)
```

Each constructed chart is automatically added to the manifold's user atlas:

```
sage: M.atlas()
[Chart (R^3, (x, y, z)), Chart (U, (r, th, ph))]
```

and to the atlas of its domain:

```
sage: U.atlas()
[Chart (U, (r, th, ph))]
```

Manifold subsets have a *default chart*, which, unless changed via the method `set_default_chart()`, is the first defined chart on the subset (or on a open subset of it):

```
sage: M.default_chart()
Chart (R^3, (x, y, z))
sage: U.default_chart()
Chart (U, (r, th, ph))
```

The default charts are not privileged charts on the manifold, but rather charts whose name can be skipped in the argument list of functions having an optional `chart=` argument.

The action of the chart map  $\varphi$  on a point is obtained by means of the call operator, i.e. the operator `()`:

```
sage: p = M.point((1,0,-2)); p
Point on the 3-dimensional differentiable manifold R^3
sage: c_cart(p)
(1, 0, -2)
sage: c_cart(p) == p.coord(c_cart)
True
sage: q = M.point((2,pi/2,pi/3), chart=c_spher) # point defined by its spherical_
↪coordinates
sage: c_spher(q)
(2, 1/2*pi, 1/3*pi)
sage: c_spher(q) == q.coord(c_spher)
```

```

True
sage: a = U.point((1,pi/2,pi)) # the default coordinates on U are the spherical_
↳ ones
sage: c_spher(a)
(1, 1/2*pi, pi)
sage: c_spher(a) == a.coord(c_spher)
True

```

Cartesian coordinates on  $U$  as an example of chart construction with coordinate restrictions: since  $U$  is the complement of the half-plane  $\{y = 0, x \geq 0\}$ , we must have  $y \neq 0$  or  $x < 0$  on  $U$ . Accordingly, we set:

```

sage: c_cartU.<x,y,z> = U.chart()
sage: c_cartU.add_restrictions((y!=0, x<0)) # the tuple (y!=0, x<0) means y!=0 or_
↳ x<0
sage: # c_cartU.add_restrictions([y!=0, x<0]) would have meant y!=0 AND x<0
sage: U.atlas()
[Chart (U, (r, th, ph)), Chart (U, (x, y, z))]
sage: M.atlas()
[Chart (R^3, (x, y, z)), Chart (U, (r, th, ph)), Chart (U, (x, y, z))]
sage: c_cartU.valid_coordinates(-1,0,2)
True
sage: c_cartU.valid_coordinates(1,0,2)
False
sage: c_cart.valid_coordinates(1,0,2)
True

```

A vector frame is naturally associated to each chart:

```

sage: c_cart.frame()
Coordinate frame (R^3, (d/dx,d/dy,d/dz))
sage: c_spher.frame()
Coordinate frame (U, (d/dr,d/dth,d/dph))

```

as well as a dual frame (basis of 1-forms):

```

sage: c_cart.coframe()
Coordinate coframe (R^3, (dx,dy,dz))
sage: c_spher.coframe()
Coordinate coframe (U, (dr,dth,dph))

```

Chart grids can be drawn in 2D or 3D graphics thanks to the method `plot()`.

**restrict** (*subset, restrictions=None*)

Return the restriction of the chart to some subset.

If the current chart is  $(U, \varphi)$ , a *restriction* (or *subchart*) is a chart  $(V, \psi)$  such that  $V \subset U$  and  $\psi = \varphi|_V$ .

If such subchart has not been defined yet, it is constructed here.

The coordinates of the subchart bare the same names as the coordinates of the original chart.

INPUT:

- *subset* – open subset  $V$  of the chart domain  $U$
- *restrictions* – (default: None) list of coordinate restrictions defining the subset  $V$

A restriction can be any symbolic equality or inequality involving the coordinates, such as  $x > y$  or  $x^2 + y^2 \neq 0$ . The items of the list `restrictions` are combined with the `and` operator; if some

restrictions are to be combined with the `or` operator instead, they have to be passed as a tuple in some single item of the list `restrictions`. For example:

```
restrictions = [x > y, (x != 0, y != 0), z^2 < x]
```

means  $(x > y)$  and  $((x \neq 0) \text{ or } (y \neq 0))$  and  $(z^2 < x)$ . If the list `restrictions` contains only one item, this item can be passed as such, i.e. writing  $x > y$  instead of the single element list  $[x > y]$ .

OUTPUT:

- a *RealDiffChart*  $(V, \psi)$

EXAMPLES:

Cartesian coordinates on the unit open disc in  $\mathbf{R}^2$  as a subchart of the global Cartesian coordinates:

```
sage: M = Manifold(2, 'R^2')
sage: c_cart.<x,y> = M.chart() # Cartesian coordinates on R^2
sage: D = M.open_subset('D') # the unit open disc
sage: c_cart_D = c_cart.restrict(D, x^2+y^2<1)
sage: p = M.point((1/2, 0))
sage: p in D
True
sage: q = M.point((1, 2))
sage: q in D
False
```

Cartesian coordinates on the annulus  $1 < \sqrt{x^2 + y^2} < 2$ :

```
sage: A = M.open_subset('A')
sage: c_cart_A = c_cart.restrict(A, [x^2+y^2>1, x^2+y^2<4])
sage: p in A, q in A
(False, False)
sage: a = M.point((3/2, 0))
sage: a in A
True
```

## 2.3 The Real Line and Open Intervals

The class *OpenInterval* implement open intervals as 1-dimensional differentiable manifolds over  $\mathbf{R}$ . The derived class *RealLine* is devoted to  $\mathbf{R}$  itself, as the open interval  $(-\infty, +\infty)$ .

AUTHORS:

- Eric Gourgoulhon (2015): initial version
- Travis Scrimshaw (2016): review tweaks

REFERENCES:

- [Lee2013]

**class** `sage.manifolds.differentiable.real_line.OpenInterval` (*lower*, *upper*, *ambient=None*, *name=None*, *latex\_name=None*, *coordinate=None*, *names=None*, *start\_index=0*)

Bases: `sage.manifolds.differentiable.manifold.DifferentiableManifold`

Open interval as a 1-dimensional differentiable manifold over  $\mathbf{R}$ .

INPUT:

- `lower` – lower bound of the interval (possibly `-Infinity`)
- `upper` – upper bound of the interval (possibly `+Infinity`)
- `ambient` – (default: `None`) another open interval, to which the constructed interval is a subset of
- `name` – (default: `None`) string; name (symbol) given to the interval; if `None`, the name is constructed from `lower` and `upper`
- `latex_name` – (default: `None`) string; LaTeX symbol to denote the interval; if `None`, the LaTeX symbol is constructed from `lower` and `upper` if `name` is `None`, otherwise, it is set to `name`
- `coordinate` – (default: `None`) string defining the symbol of the canonical coordinate set on the interval; if none is provided and `names` is `None`, the symbol ‘t’ is used
- `names` – (default: `None`) used only when `coordinate` is `None`: it must be a single-element tuple containing the canonical coordinate symbol (this is guaranteed if the shortcut `<names>` is used, see examples below)
- `start_index` – (default: `0`) unique value of the index for vectors and forms on the interval manifold

EXAMPLES:

The interval  $(0, \pi)$ :

```
sage: I = OpenInterval(0, pi); I
Real interval (0, pi)
sage: latex(I)
\left(0, \pi\right)
```

`I` is a 1-dimensional smooth manifold over  $\mathbf{R}$ :

```
sage: I.category()
Category of smooth manifolds over Real Field with 53 bits of precision
sage: I.base_field()
Real Field with 53 bits of precision
sage: dim(I)
1
```

It is infinitely differentiable (smooth manifold):

```
sage: I.diff_degree()
+Infinity
```

The instance is unique (as long as the constructor arguments are the same):

```
sage: I is OpenInterval(0, pi)
True
sage: I is OpenInterval(0, pi, name='I')
False
```

The display of the interval can be customized:

```
sage: I # default display
Real interval (0, pi)
sage: latex(I) # default LaTeX display
```

```

\left(0, \pi\right)
sage: I1 = OpenInterval(0, pi, name='I'); I1
Real interval I
sage: latex(I1)
I
sage: I2 = OpenInterval(0, pi, name='I', latex_name=r'\mathcal{I}'); I2
Real interval I
sage: latex(I2)
\mathcal{I}

```

I is endowed with a canonical chart:

```

sage: I.canonical_chart()
Chart ((0, pi), (t,))
sage: I.canonical_chart() is I.default_chart()
True
sage: I.atlas()
[Chart ((0, pi), (t,))]

```

The canonical coordinate is returned by the method `canonical_coordinate()`:

```

sage: I.canonical_coordinate()
t
sage: t = I.canonical_coordinate()
sage: type(t)
<type 'sage.symbolic.expression.Expression'>

```

However, it can be obtained in the same step as the interval construction by means of the shortcut `I.<names>`:

```

sage: I.<t> = OpenInterval(0, pi)
sage: t
t
sage: type(t)
<type 'sage.symbolic.expression.Expression'>

```

The trick is performed by the Sage preparer:

```

sage: prepare("I.<t> = OpenInterval(0, pi)")
"I = OpenInterval(Integer(0), pi, names=('t',)); (t,) = I._first_ngens(1)"

```

In particular the shortcut can be used to set a canonical coordinate symbol different from 't':

```

sage: J.<x> = OpenInterval(0, pi)
sage: J.canonical_chart()
Chart ((0, pi), (x,))
sage: J.canonical_coordinate()
x

```

The LaTeX symbol of the canonical coordinate can be adjusted via the same syntax as a chart declaration (see [RealChart](#)):

```

sage: J.<x> = OpenInterval(0, pi, coordinate=r'x:\xi')
sage: latex(x)
{\xi}
sage: latex(J.canonical_chart())
\left(\left(0, \pi\right), ({\xi})\right)

```

An element of the open interval I:

```
sage: x = I.an_element(); x
Point on the Real interval (0, pi)
sage: x.coord() # coordinates in the default chart = canonical chart
(1/2*pi,)
```

As for any manifold subset, a specific element of  $I$  can be created by providing a tuple containing its coordinate(s) in a given chart:

```
sage: x = I((2,)) # (2,) = tuple of coordinates in the canonical chart
sage: x
Point on the Real interval (0, pi)
```

But for convenience, it can also be created directly from the coordinate:

```
sage: x = I(2); x
Point on the Real interval (0, pi)
sage: x.coord()
(2,)
sage: I(2) == I((2,))
True
```

By default, the coordinates passed for the element  $x$  are those relative to the canonical chart:

```
sage: I(2) == I((2,), chart=I.canonical_chart())
True
```

The lower and upper bounds of the interval  $I$ :

```
sage: I.lower_bound()
0
sage: I.upper_bound()
pi
```

One of the endpoint can be infinite:

```
sage: J = OpenInterval(1, +oo); J
Real interval (1, +Infinity)
sage: J.an_element().coord()
(2,)
```

The construction of a subinterval can be performed via the argument `ambient` of `OpenInterval`:

```
sage: J = OpenInterval(0, 1, ambient=I); J
Real interval (0, 1)
```

However, it is recommended to use the method `open_interval()` instead:

```
sage: J = I.open_interval(0, 1); J
Real interval (0, 1)
sage: J.is_subset(I)
True
sage: J.manifold() is I
True
```

A subinterval of a subinterval:

```

sage: K = J.open_interval(1/2, 1); K
Real interval (1/2, 1)
sage: K.is_subset(J)
True
sage: K.is_subset(I)
True
sage: K.manifold() is I
True

```

We have:

```

sage: I.list_of_subsets()
[Real interval (0, 1), Real interval (0, pi), Real interval (1/2, 1)]
sage: J.list_of_subsets()
[Real interval (0, 1), Real interval (1/2, 1)]
sage: K.list_of_subsets()
[Real interval (1/2, 1)]

```

As any open subset of a manifold, open subintervals are created in a category of subobjects of smooth manifolds:

```

sage: J.category()
Join of Category of subobjects of sets and Category of smooth manifolds
over Real Field with 53 bits of precision
sage: K.category()
Join of Category of subobjects of sets and Category of smooth manifolds
over Real Field with 53 bits of precision

```

On the contrary, I, which has not been created as a subinterval, is in the category of smooth manifolds (see Manifolds):

```

sage: I.category()
Category of smooth manifolds over Real Field with 53 bits of precision

```

and we have:

```

sage: J.category() is I.category().Subobjects()
True

```

All intervals are parents:

```

sage: x = J(1/2); x
Point on the Real interval (0, pi)
sage: x.parent() is J
True
sage: y = K(3/4); y
Point on the Real interval (0, pi)
sage: y.parent() is K
True

```

We have:

```

sage: x in I, x in J, x in K
(True, True, False)
sage: y in I, y in J, y in K
(True, True, True)

```

The canonical chart of subintervals is inherited from the canonical chart of the parent interval:

```

sage: XI = I.canonical_chart(); XI
Chart ((0, pi), (t,))
sage: XI.coord_range()
t: (0, pi)
sage: XJ = J.canonical_chart(); XJ
Chart ((0, 1), (t,))
sage: XJ.coord_range()
t: (0, 1)
sage: XK = K.canonical_chart(); XK
Chart ((1/2, 1), (t,))
sage: XK.coord_range()
t: (1/2, 1)

```

**canonical\_chart()**

Return the canonical chart defined on `self`.

OUTPUT:

- RealDiffChart*

EXAMPLES:

Canonical chart on the interval  $(0, \pi)$ :

```

sage: I = OpenInterval(0, pi)
sage: I.canonical_chart()
Chart ((0, pi), (t,))
sage: I.canonical_chart().coord_range()
t: (0, pi)

```

The symbol used for the coordinate of the canonical chart is that defined during the construction of the interval:

```

sage: I.<x> = OpenInterval(0, pi)
sage: I.canonical_chart()
Chart ((0, pi), (x,))

```

**canonical\_coordinate()**

Return the canonical coordinate defined on the interval.

OUTPUT:

- the symbolic variable representing the canonical coordinate

EXAMPLES:

Canonical coordinate on the interval  $(0, \pi)$ :

```

sage: I = OpenInterval(0, pi)
sage: I.canonical_coordinate()
t
sage: type(I.canonical_coordinate())
<type 'sage.symbolic.expression.Expression'>
sage: I.canonical_coordinate().is_real()
True

```

The canonical coordinate is the first (unique) coordinate of the canonical chart:

```

sage: I.canonical_coordinate() is I.canonical_chart()[0]
True

```

Its default symbol is  $t$ ; but it can be customized during the creation of the interval:

```
sage: I = OpenInterval(0, pi, coordinate='x')
sage: I.canonical_coordinate()
x
sage: I.<x> = OpenInterval(0, pi)
sage: I.canonical_coordinate()
x
```

### **inf()**

Return the lower bound (infimum) of the interval.

EXAMPLES:

```
sage: I = OpenInterval(1/4, 3)
sage: I.lower_bound()
1/4
sage: J = OpenInterval(-oo, 2)
sage: J.lower_bound()
-Infinity
```

An alias of `lower_bound()` is `inf()`:

```
sage: I.inf()
1/4
sage: J.inf()
-Infinity
```

### **lower\_bound()**

Return the lower bound (infimum) of the interval.

EXAMPLES:

```
sage: I = OpenInterval(1/4, 3)
sage: I.lower_bound()
1/4
sage: J = OpenInterval(-oo, 2)
sage: J.lower_bound()
-Infinity
```

An alias of `lower_bound()` is `inf()`:

```
sage: I.inf()
1/4
sage: J.inf()
-Infinity
```

### **open\_interval(lower, upper, name=None, latex\_name=None)**

Define an open subinterval of `self`.

INPUT:

- `lower` – lower bound of the subinterval (possibly `-Infinity`)
- `upper` – upper bound of the subinterval (possibly `+Infinity`)
- `name` – (default: `None`) string; name (symbol) given to the subinterval; if `None`, the name is constructed from `lower` and `upper`
- `latex_name` – (default: `None`) string; LaTeX symbol to denote the subinterval; if `None`, the LaTeX symbol is constructed from `lower` and `upper` if `name` is `None`, otherwise, it is set to `name`

OUTPUT:

- *OpenInterval* representing the open interval (lower, upper)

EXAMPLES:

The interval  $(0, \pi)$  as a subinterval of  $(-4, 4)$ :

```
sage: I = OpenInterval(-4, 4)
sage: J = I.open_interval(0, pi); J
Real interval (0, pi)
sage: J.is_subset(I)
True
sage: I.list_of_subsets()
[Real interval (-4, 4), Real interval (0, pi)]
```

J is considered as an open submanifold of I:

```
sage: J.manifold() is I
True
```

The subinterval  $(-4, 4)$  is I itself:

```
sage: I.open_interval(-4, 4) is I
True
```

**sup()**

Return the upper bound (supremum) of the interval.

EXAMPLES:

```
sage: I = OpenInterval(1/4, 3)
sage: I.upper_bound()
3
sage: J = OpenInterval(1, +oo)
sage: J.upper_bound()
+Infinity
```

An alias of *upper\_bound()* is *sup()*:

```
sage: I.sup()
3
sage: J.sup()
+Infinity
```

**upper\_bound()**

Return the upper bound (supremum) of the interval.

EXAMPLES:

```
sage: I = OpenInterval(1/4, 3)
sage: I.upper_bound()
3
sage: J = OpenInterval(1, +oo)
sage: J.upper_bound()
+Infinity
```

An alias of *upper\_bound()* is *sup()*:

```
sage: I.sup()
3
sage: J.sup()
+Infinity
```

```
class sage.manifolds.differentiable.real_line.RealLine (name='R', latex_name='\Bold{R}',
coordinate=None, names=None, start_index=0)
```

Bases: *sage.manifolds.differentiable.real\_line.OpenInterval*

Field of real numbers, as a differentiable manifold of dimension 1 (real line) with a canonical coordinate chart.

INPUT:

- `name` – (default: 'R') string; name (symbol) given to the real line
- `latex_name` – (default: `r'\Bold{R}'`) string; LaTeX symbol to denote the real line
- `coordinate` – (default: None) string defining the symbol of the canonical coordinate set on the real line; if none is provided and `names` is None, the symbol 't' is used
- `names` – (default: None) used only when `coordinate` is None: it must be a single-element tuple containing the canonical coordinate symbol (this is guaranteed if the shortcut `<names>` is used, see examples below)
- `start_index` – (default: 0) unique value of the index for vectors and forms on the real line manifold

EXAMPLES:

Constructing the real line without any argument:

```
sage: R = RealLine() ; R
Real number line R
sage: latex(R)
\Bold{R}
```

R is a 1-dimensional real smooth manifold:

```
sage: R.category()
Category of smooth manifolds over Real Field with 53 bits of precision
sage: isinstance(R, sage.manifolds.differentiable.manifold.DifferentiableManifold)
True
sage: dim(R)
1
```

It is endowed with a canonical chart:

```
sage: R.canonical_chart()
Chart (R, (t,))
sage: R.canonical_chart() is R.default_chart()
True
sage: R.atlas()
[Chart (R, (t,))]
```

The instance is unique (as long as the constructor arguments are the same):

```
sage: R is RealLine()
True
sage: R is RealLine(latex_name='R')
False
```

The canonical coordinate is returned by the method `canonical_coordinate()`:

```
sage: R.canonical_coordinate()
t
sage: t = R.canonical_coordinate()
sage: type(t)
<type 'sage.symbolic.expression.Expression'>
```

However, it can be obtained in the same step as the real line construction by means of the shortcut `R.<names>`:

```
sage: R.<t> = RealLine()
sage: t
t
sage: type(t)
<type 'sage.symbolic.expression.Expression'>
```

The trick is performed by Sage preparer:

```
sage: prepare("R.<t> = RealLine() ")
"R = RealLine(names=('t',)); (t,) = R._first_ngens(1) "
```

In particular the shortcut is to be used to set a canonical coordinate symbol different from 't':

```
sage: R.<x> = RealLine()
sage: R.canonical_chart()
Chart (R, (x,))
sage: R.atlas()
[Chart (R, (x,))]
sage: R.canonical_coordinate()
x
```

The LaTeX symbol of the canonical coordinate can be adjusted via the same syntax as a chart declaration (see [RealChart](#)):

```
sage: R.<x> = RealLine(coordinate=r'x:\xi')
sage: latex(x)
{\xi}
sage: latex(R.canonical_chart())
\left(\mathbf{R}, (\xi)\right)
```

The LaTeX symbol of the real line itself can also be customized:

```
sage: R.<x> = RealLine(latex_name=r'\mathbb{R}')
sage: latex(R)
\mathbb{R}
```

Elements of the real line can be constructed directly from a number:

```
sage: p = R(2) ; p
Point on the Real number line R
sage: p.coord()
(2,)
sage: p = R(1.742) ; p
Point on the Real number line R
sage: p.coord()
(1.7420000000000000,)
```

Symbolic variables can also be used:

```

sage: p = R(pi, name='pi') ; p
Point pi on the Real number line R
sage: p.coord()
(pi,)
sage: a = var('a')
sage: p = R(a) ; p
Point on the Real number line R
sage: p.coord()
(a,)

```

The real line is considered as the open interval  $(-\infty, +\infty)$ :

```

sage: isinstance(R, sage.manifolds.differentiable.real_line.OpenInterval)
True
sage: R.lower_bound()
-Infinity
sage: R.upper_bound()
+Infinity

```

A real interval can be created from R means of the method `open_interval()`:

```

sage: I = R.open_interval(0, 1); I
Real interval (0, 1)
sage: I.manifold()
Real number line R
sage: R.list_of_subsets()
[Real interval (0, 1), Real number line R]

```

## 2.4 Scalar Fields

### 2.4.1 Algebra of Differentiable Scalar Fields

The class `DiffScalarFieldAlgebra` implements the commutative algebra  $C^k(M)$  of differentiable scalar fields on a differentiable manifold  $M$  of class  $C^k$  over a topological field  $K$  (in most applications,  $K = \mathbf{R}$  or  $K = \mathbf{C}$ ). By *differentiable scalar field*, it is meant a function  $M \rightarrow K$  that is  $k$ -times continuously differentiable.  $C^k(M)$  is an algebra over  $K$ , whose ring product is the pointwise multiplication of  $K$ -valued functions, which is clearly commutative.

AUTHORS:

- Ericourgoulhon, Michal Bejger (2014-2015): initial version

REFERENCES:

- [KN1963]
- [Lee2013]
- [ONe1983]

**class** `sage.manifolds.differentiable.scalarfield_algebra.DiffScalarFieldAlgebra` (*domain*)  
 Bases: `sage.manifolds.scalarfield_algebra.ScalarFieldAlgebra`

Commutative algebra of differentiable scalar fields on a differentiable manifold.

If  $M$  is a differentiable manifold of class  $C^k$  over a topological field  $K$ , the *commutative algebra of scalar fields on  $M$*  is the set  $C^k(M)$  of all  $k$ -times continuously differentiable maps  $M \rightarrow K$ . The set  $C^k(M)$  is

an algebra over  $K$ , whose ring product is the pointwise multiplication of  $K$ -valued functions, which is clearly commutative.

If  $K = \mathbf{R}$  or  $K = \mathbf{C}$ , the field  $K$  over which the algebra  $C^k(M)$  is constructed is represented by Sage's Symbolic Ring SR, since there is no exact representation of  $\mathbf{R}$  nor  $\mathbf{C}$  in Sage.

Via its base class `ScalarFieldAlgebra`, the class `DiffScalarFieldAlgebra` inherits from `Parent`, with the category set to `CommutativeAlgebras`. The corresponding *element* class is `DiffScalarField`.

INPUT:

- `domain` – the differentiable manifold  $M$  on which the scalar fields are defined (must be an instance of class `DifferentiableManifold`)

EXAMPLES:

Algebras of scalar fields on the sphere  $S^2$  and on some open subset of it:

```
sage: M = Manifold(2, 'M') # the 2-dimensional sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                                     intersection_name='W', restrictions1= x^2+y^2!=0,
....:                                     restrictions2= u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: CM = M.scalar_field_algebra() ; CM
Algebra of differentiable scalar fields on the 2-dimensional
differentiable manifold M
sage: W = U.intersection(V) # S^2 minus the two poles
sage: CW = W.scalar_field_algebra() ; CW
Algebra of differentiable scalar fields on the Open subset W of the
2-dimensional differentiable manifold M
```

$C^k(M)$  and  $C^k(W)$  belong to the category of commutative algebras over  $\mathbf{R}$  (represented here by Sage's Symbolic Ring):

```
sage: CM.category()
Category of commutative algebras over Symbolic Ring
sage: CM.base_ring()
Symbolic Ring
sage: CW.category()
Category of commutative algebras over Symbolic Ring
sage: CW.base_ring()
Symbolic Ring
```

The elements of  $C^k(M)$  are scalar fields on  $M$ :

```
sage: CM.an_element()
Scalar field on the 2-dimensional differentiable manifold M
sage: CM.an_element().display() # this sample element is a constant field
M --> R
on U: (x, y) |--> 2
on V: (u, v) |--> 2
```

Those of  $C^k(W)$  are scalar fields on  $W$ :

```

sage: CW.an_element()
Scalar field on the Open subset W of the 2-dimensional differentiable
manifold M
sage: CW.an_element().display() # this sample element is a constant field
W --> R
(x, y) |--> 2
(u, v) |--> 2

```

The zero element:

```

sage: CM.zero()
Scalar field zero on the 2-dimensional differentiable manifold M
sage: CM.zero().display()
zero: M --> R
on U: (x, y) |--> 0
on V: (u, v) |--> 0

```

```

sage: CW.zero()
Scalar field zero on the Open subset W of the 2-dimensional
differentiable manifold M
sage: CW.zero().display()
zero: W --> R
(x, y) |--> 0
(u, v) |--> 0

```

The unit element:

```

sage: CM.one()
Scalar field 1 on the 2-dimensional differentiable manifold M
sage: CM.one().display()
1: M --> R
on U: (x, y) |--> 1
on V: (u, v) |--> 1

```

```

sage: CW.one()
Scalar field 1 on the Open subset W of the 2-dimensional differentiable
manifold M
sage: CW.one().display()
1: W --> R
(x, y) |--> 1
(u, v) |--> 1

```

A generic element can be constructed as for any parent in Sage, namely by means of the `__call__` operator on the parent (here with the dictionary of the coordinate expressions defining the scalar field):

```

sage: f = CM({c_xy: atan(x^2+y^2), c_uv: pi/2 - atan(u^2+v^2)}); f
Scalar field on the 2-dimensional differentiable manifold M
sage: f.display()
M --> R
on U: (x, y) |--> arctan(x^2 + y^2)
on V: (u, v) |--> 1/2*pi - arctan(u^2 + v^2)
sage: f.parent()
Algebra of differentiable scalar fields on the 2-dimensional
differentiable manifold M

```

Specific elements can also be constructed in this way:

```
sage: CM(0) == CM.zero()
True
sage: CM(1) == CM.one()
True
```

Note that the zero scalar field is cached:

```
sage: CM(0) is CM.zero()
True
```

Elements can also be constructed by means of the method `scalar_field()` acting on the domain (this allows one to set the name of the scalar field at the construction):

```
sage: f1 = M.scalar_field({c_xy: atan(x^2+y^2), c_uv: pi/2 - atan(u^2+v^2)},
.....:                    name='f')
sage: f1.parent()
Algebra of differentiable scalar fields on the 2-dimensional
differentiable manifold M
sage: f1 == f
True
sage: M.scalar_field(0, chart='all') == CM.zero()
True
```

The algebra  $C^k(M)$  coerces to  $C^k(W)$  since  $W$  is an open subset of  $M$ :

```
sage: CW.has_coerce_map_from(CM)
True
```

The reverse is of course false:

```
sage: CM.has_coerce_map_from(CW)
False
```

The coercion map is nothing but the restriction to  $W$  of scalar fields on  $M$ :

```
sage: fW = CW(f) ; fW
Scalar field on the Open subset W of the 2-dimensional differentiable
manifold M
sage: fW.display()
W --> R
(x, y) |--> arctan(x^2 + y^2)
(u, v) |--> 1/2*pi - arctan(u^2 + v^2)
```

```
sage: CW(CM.one()) == CW.one()
True
```

The coercion map allows for the addition of elements of  $C^k(W)$  with elements of  $C^k(M)$ , the result being an element of  $C^k(W)$ :

```
sage: s = fW + f
sage: s.parent()
Algebra of differentiable scalar fields on the Open subset W of the
2-dimensional differentiable manifold M
sage: s.display()
W --> R
(x, y) |--> 2*arctan(x^2 + y^2)
(u, v) |--> pi - 2*arctan(u^2 + v^2)
```

Another coercion is that from the Symbolic Ring, the parent of all symbolic expressions (cf. `SymbolicRing`). Since the Symbolic Ring is the base ring for the algebra `CM`, the coercion of a symbolic expression `s` is performed by the operation `s*CM.one()`, which invokes the reflected multiplication operator `sage.manifolds.scalarfield.ScalarField.__mul__()`. If the symbolic expression does not involve any chart coordinate, the outcome is a constant scalar field:

```
sage: h = CM(pi*sqrt(2)) ; h
Scalar field on the 2-dimensional differentiable manifold M
sage: h.display()
M --> R
on U: (x, y) |--> sqrt(2)*pi
on V: (u, v) |--> sqrt(2)*pi
sage: a = var('a')
sage: h = CM(a); h.display()
M --> R
on U: (x, y) |--> a
on V: (u, v) |--> a
```

If the symbolic expression involves some coordinate of one of the manifold's charts, the outcome is initialized only on the chart domain:

```
sage: h = CM(a+x); h.display()
M --> R
on U: (x, y) |--> a + x
sage: h = CM(a+u); h.display()
M --> R
on V: (u, v) |--> a + u
```

If the symbolic expression involves coordinates of different charts, the scalar field is created as a Python object, but is not initialized, in order to avoid any ambiguity:

```
sage: h = CM(x+u); h.display()
M --> R
```

## TESTS OF THE ALGEBRA LAWS:

Ring laws:

```
sage: h = CM(pi*sqrt(2))
sage: s = f + h ; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M --> R
on U: (x, y) |--> sqrt(2)*pi + arctan(x^2 + y^2)
on V: (u, v) |--> 1/2*pi*(2*sqrt(2) + 1) - arctan(u^2 + v^2)
```

```
sage: s = f - h ; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M --> R
on U: (x, y) |--> -sqrt(2)*pi + arctan(x^2 + y^2)
on V: (u, v) |--> -1/2*pi*(2*sqrt(2) - 1) - arctan(u^2 + v^2)
```

```
sage: s = f*h ; s
Scalar field on the 2-dimensional differentiable manifold M
```

```

sage: s.display()
M --> R
on U: (x, y) |--> sqrt(2)*pi*arctan(x^2 + y^2)
on V: (u, v) |--> 1/2*sqrt(2)*(pi^2 - 2*pi*arctan(u^2 + v^2))

```

```

sage: s = f/h ; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M --> R
on U: (x, y) |--> 1/2*sqrt(2)*arctan(x^2 + y^2)/pi
on V: (u, v) |--> 1/4*(sqrt(2)*pi - 2*sqrt(2)*arctan(u^2 + v^2))/pi

```

```

sage: f*(h+f) == f*h + f*f
True

```

Ring laws with coercion:

```

sage: f - fW == CW.zero()
True
sage: f/fW == CW.one()
True
sage: s = f*fW ; s
Scalar field on the Open subset W of the 2-dimensional differentiable
manifold M
sage: s.display()
W --> R
(x, y) |--> arctan(x^2 + y^2)^2
(u, v) |--> 1/4*pi^2 - pi*arctan(u^2 + v^2) + arctan(u^2 + v^2)^2
sage: s/f == fW
True

```

Multiplication by a number:

```

sage: s = 2*f ; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M --> R
on U: (x, y) |--> 2*arctan(x^2 + y^2)
on V: (u, v) |--> pi - 2*arctan(u^2 + v^2)

```

```

sage: 0*f == CM.zero()
True
sage: 1*f == f
True
sage: 2*(f/2) == f
True
sage: (f+2*f)/3 == f
True
sage: 1/3*(f+2*f) == f
True

```

The Sage test suite for algebras is passed:

```

sage: TestSuite(CM).run()

```

It is passed also for  $C^k(W)$ :

```
sage: TestSuite(CW).run()
```

**Element**alias of `DiffScalarField`

## 2.4.2 Differentiable Scalar Fields

Given a differentiable manifold  $M$  of class  $C^k$  over a topological field  $K$  (in most applications,  $K = \mathbf{R}$  or  $K = \mathbf{C}$ ), a *differentiable scalar field* on  $M$  is a map

$$f : M \longrightarrow K$$

of class  $C^k$ .

Differentiable scalar fields are implemented by the class `DiffScalarField`.

**AUTHORS:**

- Eric Gourgoulhon, Michal Bejger (2013-2015): initial version

**REFERENCES:**

- [KN1963]
- [Lee2013]
- [ONe1983]

```
class sage.manifolds.differentiable.scalarfield.DiffScalarField(parent, coord_expression=None,
                                                                chart=None,
                                                                name=None, latex_name=None)
```

Bases: `sage.manifolds.scalarfield.ScalarField`

Differentiable scalar field on a differentiable manifold.

Given a differentiable manifold  $M$  of class  $C^k$  over a topological field  $K$  (in most applications,  $K = \mathbf{R}$  or  $K = \mathbf{C}$ ), a *differentiable scalar field* defined on  $M$  is a map

$$f : M \longrightarrow K$$

that is  $k$ -times continuously differentiable.

The class `DiffScalarField` is a Sage *element* class, whose *parent* class is `DiffScalarFieldAlgebra`. It inherits from the class `ScalarField` devoted to generic continuous scalar fields on topological manifolds.

**INPUT:**

- `parent` – the algebra of scalar fields containing the scalar field (must be an instance of class `DiffScalarFieldAlgebra`)
- `coord_expression` – (default: `None`) coordinate expression(s) of the scalar field; this can be either
  - a dictionary of coordinate expressions in various charts on the domain, with the charts as keys;
  - a single coordinate expression; if the argument `chart` is `'all'`, this expression is set to all the charts defined on the open set; otherwise, the expression is set in the specific chart provided by the argument `chart`

NB: If `coord_expression` is `None` or `incomplete`, coordinate expressions can be added after the creation of the object, by means of the methods `add_expr()`, `add_expr_by_continuation()` and `set_expr()`

- `chart` – (default: `None`) chart defining the coordinates used in `coord_expression` when the latter is a single coordinate expression; if none is provided (default), the default chart of the open set is assumed. If `chart=='all'`, `coord_expression` is assumed to be independent of the chart (constant scalar field).
- `name` – (default: `None`) string; name (symbol) given to the scalar field
- `latex_name` – (default: `None`) string; LaTeX symbol to denote the scalar field; if none is provided, the LaTeX symbol is set to `name`

#### EXAMPLES:

A scalar field on the 2-sphere:

```
sage: M = Manifold(2, 'M') # the 2-dimensional sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                               intersection_name='W',
....:                               restrictions1= x^2+y^2!=0,
....:                               restrictions2= u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: f = M.scalar_field({c_xy: 1/(1+x^2+y^2), c_uv: (u^2+v^2)/(1+u^2+v^2)},
....:                   name='f') ; f
Scalar field f on the 2-dimensional differentiable manifold M
sage: f.display()
f: M --> R
on U: (x, y) |--> 1/(x^2 + y^2 + 1)
on V: (u, v) |--> (u^2 + v^2)/(u^2 + v^2 + 1)
```

For scalar fields defined by a single coordinate expression, the latter can be passed instead of the dictionary over the charts:

```
sage: g = U.scalar_field(x*y, chart=c_xy, name='g') ; g
Scalar field g on the Open subset U of the 2-dimensional differentiable
manifold M
```

The above is indeed equivalent to:

```
sage: g = U.scalar_field({c_xy: x*y}, name='g') ; g
Scalar field g on the Open subset U of the 2-dimensional differentiable
manifold M
```

Since `c_xy` is the default chart of `U`, the argument `chart` can be skipped:

```
sage: g = U.scalar_field(x*y, name='g') ; g
Scalar field g on the Open subset U of the 2-dimensional differentiable
manifold M
```

The scalar field  $g$  is defined on  $U$  and has an expression in terms of the coordinates  $(u, v)$  on  $W = U \cap V$ :

```
sage: g.display()
g: U --> R
```

```
(x, y) |--> x*y
on W: (u, v) |--> u*v/(u^4 + 2*u^2*v^2 + v^4)
```

Scalar fields on  $M$  can also be declared with a single chart:

```
sage: f = M.scalar_field(1/(1+x^2+y^2), chart=c_xy, name='f') ; f
Scalar field f on the 2-dimensional differentiable manifold M
```

Their definition must then be completed by providing the expressions on other charts, via the method `add_expr()`, to get a global cover of the manifold:

```
sage: f.add_expr((u^2+v^2)/(1+u^2+v^2), chart=c_uv)
sage: f.display()
f: M --> R
on U: (x, y) |--> 1/(x^2 + y^2 + 1)
on V: (u, v) |--> (u^2 + v^2)/(u^2 + v^2 + 1)
```

We can even first declare the scalar field without any coordinate expression and provide them subsequently:

```
sage: f = M.scalar_field(name='f')
sage: f.add_expr(1/(1+x^2+y^2), chart=c_xy)
sage: f.add_expr((u^2+v^2)/(1+u^2+v^2), chart=c_uv)
sage: f.display()
f: M --> R
on U: (x, y) |--> 1/(x^2 + y^2 + 1)
on V: (u, v) |--> (u^2 + v^2)/(u^2 + v^2 + 1)
```

We may also use the method `add_expr_by_continuation()` to complete the coordinate definition using the analytic continuation from domains in which charts overlap:

```
sage: f = M.scalar_field(1/(1+x^2+y^2), chart=c_xy, name='f') ; f
Scalar field f on the 2-dimensional differentiable manifold M
sage: f.add_expr_by_continuation(c_uv, U.intersection(V))
sage: f.display()
f: M --> R
on U: (x, y) |--> 1/(x^2 + y^2 + 1)
on V: (u, v) |--> (u^2 + v^2)/(u^2 + v^2 + 1)
```

A scalar field can also be defined by some unspecified function of the coordinates:

```
sage: h = U.scalar_field(function('H')(x, y), name='h') ; h
Scalar field h on the Open subset U of the 2-dimensional differentiable
manifold M
sage: h.display()
h: U --> R
(x, y) |--> H(x, y)
on W: (u, v) |--> H(u/(u^2 + v^2), v/(u^2 + v^2))
```

We may use the argument `latex_name` to specify the LaTeX symbol denoting the scalar field if the latter is different from name:

```
sage: latex(f)
f
sage: f = M.scalar_field({c_xy: 1/(1+x^2+y^2), c_uv: (u^2+v^2)/(1+u^2+v^2)},
....:                      name='f', latex_name=r'\mathcal{F}')
sage: latex(f)
\mathcal{F}
```

The coordinate expression in a given chart is obtained via the method `expr()`, which returns a symbolic expression:

```
sage: f.expr(c_uv)
(u^2 + v^2)/(u^2 + v^2 + 1)
sage: type(f.expr(c_uv))
<type 'sage.symbolic.expression.Expression'>
```

The method `coord_function()` returns instead a function of the chart coordinates, i.e. an instance of `CoordFunction`:

```
sage: f.coord_function(c_uv)
(u^2 + v^2)/(u^2 + v^2 + 1)
sage: type(f.coord_function(c_uv))
<class 'sage.manifolds.coord_func_symb.CoordFunctionSymbRing_with_category.
  ↳element_class'>
sage: f.coord_function(c_uv).display()
(u, v) |--> (u^2 + v^2)/(u^2 + v^2 + 1)
```

The value returned by the method `expr()` is actually the coordinate expression of the chart function:

```
sage: f.expr(c_uv) is f.coord_function(c_uv).expr()
True
```

A constant scalar field is declared by setting the argument `chart` to `'all'`:

```
sage: c = M.scalar_field(2, chart='all', name='c') ; c
Scalar field c on the 2-dimensional differentiable manifold M
sage: c.display()
c: M --> R
on U: (x, y) |--> 2
on V: (u, v) |--> 2
```

A shortcut is to use the method `constant_scalar_field()`:

```
sage: c == M.constant_scalar_field(2)
True
```

The constant value can be some unspecified parameter:

```
sage: var('a')
a
sage: c = M.constant_scalar_field(a, name='c') ; c
Scalar field c on the 2-dimensional differentiable manifold M
sage: c.display()
c: M --> R
on U: (x, y) |--> a
on V: (u, v) |--> a
```

A special case of constant field is the zero scalar field:

```
sage: zer = M.constant_scalar_field(0) ; zer
Scalar field zero on the 2-dimensional differentiable manifold M
sage: zer.display()
zero: M --> R
on U: (x, y) |--> 0
on V: (u, v) |--> 0
```

It can be obtained directly by means of the function `zero_scalar_field()`:

```
sage: zer is M.zero_scalar_field()
True
```

A third way is to get it as the zero element of the algebra  $C^k(M)$  of scalar fields on  $M$  (see below):

```
sage: zer is M.scalar_field_algebra().zero()
True
```

By definition, a scalar field acts on the manifold's points, sending them to elements of the manifold's base field (real numbers in the present case):

```
sage: N = M.point((0,0), chart=c_uv) # the North pole
sage: S = M.point((0,0), chart=c_xy) # the South pole
sage: E = M.point((1,0), chart=c_xy) # a point at the equator
sage: f(N)
0
sage: f(S)
1
sage: f(E)
1/2
sage: h(E)
H(1, 0)
sage: c(E)
a
sage: zer(E)
0
```

A scalar field can be compared to another scalar field:

```
sage: f == g
False
```

...to a symbolic expression:

```
sage: f == x*y
False
sage: g == x*y
True
sage: c == a
True
```

...to a number:

```
sage: f == 2
False
sage: zer == 0
True
```

...to anything else:

```
sage: f == M
False
```

Standard mathematical functions are implemented:

```

sage: sqrt(f)
Scalar field sqrt(f) on the 2-dimensional differentiable manifold M
sage: sqrt(f).display()
sqrt(f): M --> R
on U: (x, y) |--> 1/sqrt(x^2 + y^2 + 1)
on V: (u, v) |--> sqrt(u^2 + v^2)/sqrt(u^2 + v^2 + 1)

```

```

sage: tan(f)
Scalar field tan(f) on the 2-dimensional differentiable manifold M
sage: tan(f).display()
tan(f): M --> R
on U: (x, y) |--> sin(1/(x^2 + y^2 + 1))/cos(1/(x^2 + y^2 + 1))
on V: (u, v) |--> sin((u^2 + v^2)/(u^2 + v^2 + 1))/cos((u^2 + v^2)/(u^2 + v^2 + 1))

```

### Arithmetics of scalar fields

Scalar fields on  $M$  (resp.  $U$ ) belong to the algebra  $C^k(M)$  (resp.  $C^k(U)$ ):

```

sage: f.parent()
Algebra of differentiable scalar fields on the 2-dimensional
differentiable manifold M
sage: f.parent() is M.scalar_field_algebra()
True
sage: g.parent()
Algebra of differentiable scalar fields on the Open subset U of the
2-dimensional differentiable manifold M
sage: g.parent() is U.scalar_field_algebra()
True

```

Consequently, scalar fields can be added:

```

sage: s = f + c ; s
Scalar field f+c on the 2-dimensional differentiable manifold M
sage: s.display()
f+c: M --> R
on U: (x, y) |--> (a*x^2 + a*y^2 + a + 1)/(x^2 + y^2 + 1)
on V: (u, v) |--> ((a + 1)*u^2 + (a + 1)*v^2 + a)/(u^2 + v^2 + 1)

```

and subtracted:

```

sage: s = f - c ; s
Scalar field f-c on the 2-dimensional differentiable manifold M
sage: s.display()
f-c: M --> R
on U: (x, y) |--> -(a*x^2 + a*y^2 + a - 1)/(x^2 + y^2 + 1)
on V: (u, v) |--> -((a - 1)*u^2 + (a - 1)*v^2 + a)/(u^2 + v^2 + 1)

```

Some tests:

```

sage: f + zer == f
True
sage: f - f == zer
True
sage: f + (-f) == zer
True

```

```
sage: (f+c)-f == c
True
sage: (f-c)+c == f
True
```

We may add a number (interpreted as a constant scalar field) to a scalar field:

```
sage: s = f + 1 ; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M --> R
on U: (x, y) |--> (x^2 + y^2 + 2)/(x^2 + y^2 + 1)
on V: (u, v) |--> (2*u^2 + 2*v^2 + 1)/(u^2 + v^2 + 1)
sage: (f+1)-1 == f
True
```

The number can be represented by a symbolic variable:

```
sage: s = a + f ; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s == c + f
True
```

However if the symbolic variable is a chart coordinate, the addition is performed only on the chart domain:

```
sage: s = f + x; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M --> R
on U: (x, y) |--> (x^3 + x*y^2 + x + 1)/(x^2 + y^2 + 1)
sage: s = f + u; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M --> R
on V: (u, v) |--> (u^3 + (u + 1)*v^2 + u^2 + u)/(u^2 + v^2 + 1)
```

The addition of two scalar fields with different domains is possible if the domain of one of them is a subset of the domain of the other; the domain of the result is then this subset:

```
sage: f.domain()
2-dimensional differentiable manifold M
sage: g.domain()
Open subset U of the 2-dimensional differentiable manifold M
sage: s = f + g ; s
Scalar field on the Open subset U of the 2-dimensional differentiable
manifold M
sage: s.domain()
Open subset U of the 2-dimensional differentiable manifold M
sage: s.display()
U --> R
(x, y) |--> (x*y^3 + (x^3 + x)*y + 1)/(x^2 + y^2 + 1)
on W: (u, v) |--> (u^6 + 3*u^4*v^2 + 3*u^2*v^4 + v^6 + u*v^3
+ (u^3 + u)*v)/(u^6 + v^6 + (3*u^2 + 1)*v^4 + u^4 + (3*u^4 + 2*u^2)*v^2)
```

The operation actually performed is  $f|_U + g$ :

```
sage: s == f.restrict(U) + g
True
```

In Sage framework, the addition of  $f$  and  $g$  is permitted because there is a *coercion* of the parent of  $f$ , namely  $C^k(M)$ , to the parent of  $g$ , namely  $C^k(U)$  (see [DiffScalarFieldAlgebra](#)):

```
sage: CM = M.scalar_field_algebra()
sage: CU = U.scalar_field_algebra()
sage: CU.has_coerce_map_from(CM)
True
```

The coercion map is nothing but the restriction to domain  $U$ :

```
sage: CU.coerce(f) == f.restrict(U)
True
```

Since the algebra  $C^k(M)$  is a vector space over  $\mathbf{R}$ , scalar fields can be multiplied by a number, either an explicit one:

```
sage: s = 2*f ; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M --> R
on U: (x, y) |--> 2/(x^2 + y^2 + 1)
on V: (u, v) |--> 2*(u^2 + v^2)/(u^2 + v^2 + 1)
```

or a symbolic one:

```
sage: s = a*f ; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M --> R
on U: (x, y) |--> a/(x^2 + y^2 + 1)
on V: (u, v) |--> (u^2 + v^2)*a/(u^2 + v^2 + 1)
```

However, if the symbolic variable is a chart coordinate, the multiplication is performed only in the corresponding chart:

```
sage: s = x*f; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M --> R
on U: (x, y) |--> x/(x^2 + y^2 + 1)
sage: s = u*f; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M --> R
on V: (u, v) |--> (u^2 + v^2)*u/(u^2 + v^2 + 1)
```

Some tests:

```
sage: 0*f == 0
True
sage: 0*f == zer
True
sage: 1*f == f
True
sage: (-2)*f == - f - f
True
```

The ring multiplication of the algebras  $C^k(M)$  and  $C^k(U)$  is the pointwise multiplication of functions:

```

sage: s = f*f ; s
Scalar field f*f on the 2-dimensional differentiable manifold M
sage: s.display()
f*f: M --> R
on U: (x, y) |--> 1/(x^4 + y^4 + 2*(x^2 + 1)*y^2 + 2*x^2 + 1)
on V: (u, v) |--> (u^4 + 2*u^2*v^2 + v^4)/(u^4 + v^4 + 2*(u^2 + 1)*v^2 + 2*u^2 +
↪1)
sage: s = g*h ; s
Scalar field g*h on the Open subset U of the 2-dimensional
differentiable manifold M
sage: s.display()
g*h: U --> R
(x, y) |--> x*y*H(x, y)
on W: (u, v) |--> u*v*H(u/(u^2 + v^2), v/(u^2 + v^2))/(u^4 + 2*u^2*v^2 + v^4)

```

Thanks to the coercion  $C^k(M) \rightarrow C^k(U)$  mentioned above, it is possible to multiply a scalar field defined on  $M$  by a scalar field defined on  $U$ , the result being a scalar field defined on  $U$ :

```

sage: f.domain(), g.domain()
(2-dimensional differentiable manifold M,
Open subset U of the 2-dimensional differentiable manifold M)
sage: s = f*g ; s
Scalar field on the Open subset U of the 2-dimensional differentiable
manifold M
sage: s.display()
U --> R
(x, y) |--> x*y/(x^2 + y^2 + 1)
on W: (u, v) |--> u*v/(u^4 + v^4 + (2*u^2 + 1)*v^2 + u^2)
sage: s == f.restrict(U)*g
True

```

Scalar fields can be divided (pointwise division):

```

sage: s = f/c ; s
Scalar field f/c on the 2-dimensional differentiable manifold M
sage: s.display()
f/c: M --> R
on U: (x, y) |--> 1/(a*x^2 + a*y^2 + a)
on V: (u, v) |--> (u^2 + v^2)/(a*u^2 + a*v^2 + a)
sage: s = g/h ; s
Scalar field g/h on the Open subset U of the 2-dimensional
differentiable manifold M
sage: s.display()
g/h: U --> R
(x, y) |--> x*y/H(x, y)
on W: (u, v) |--> u*v/((u^4 + 2*u^2*v^2 + v^4)*H(u/(u^2 + v^2), v/(u^2 + v^2)))
sage: s = f/g ; s
Scalar field on the Open subset U of the 2-dimensional differentiable
manifold M
sage: s.display()
U --> R
(x, y) |--> 1/(x*y^3 + (x^3 + x)*y)
on W: (u, v) |--> (u^6 + 3*u^4*v^2 + 3*u^2*v^4 + v^6)/(u*v^3 + (u^3 + u)*v)
sage: s == f.restrict(U)/g
True

```

For scalar fields defined on a single chart domain, we may perform some arithmetics with symbolic expressions involving the chart coordinates:

```

sage: s = g + x^2 - y ; s
Scalar field on the Open subset U of the 2-dimensional differentiable
manifold M
sage: s.display()
U --> R
(x, y) |--> x^2 + (x - 1)*y
on W: (u, v) |--> -(v^3 - u^2 + (u^2 - u)*v)/(u^4 + 2*u^2*v^2 + v^4)

```

```

sage: s = g*x ; s
Scalar field on the Open subset U of the 2-dimensional differentiable
manifold M
sage: s.display()
U --> R
(x, y) |--> x^2*y
on W: (u, v) |--> u^2*v/(u^6 + 3*u^4*v^2 + 3*u^2*v^4 + v^6)

```

```

sage: s = g/x ; s
Scalar field on the Open subset U of the 2-dimensional differentiable
manifold M
sage: s.display()
U --> R
(x, y) |--> y
on W: (u, v) |--> v/(u^2 + v^2)
sage: s = x/g ; s
Scalar field on the Open subset U of the 2-dimensional differentiable
manifold M
sage: s.display()
U --> R
(x, y) |--> 1/y
on W: (u, v) |--> (u^2 + v^2)/v

```

The test suite is passed:

```

sage: TestSuite(f).run()
sage: TestSuite(zer).run()

```

### **differential()**

Return the differential of self.

OUTPUT:

- a *DiffForm* (or of *DiffFormParal* if the scalar field's domain is parallelizable) representing the 1-form that is the differential of the scalar field

EXAMPLES:

Differential of a scalar field on a 3-dimensional differentiable manifold:

```

sage: M = Manifold(3, 'M')
sage: c_xyz.<x,y,z> = M.chart()
sage: f = M.scalar_field(cos(x)*z^3 + exp(y)*z^2, name='f')
sage: df = f.differential() ; df
1-form df on the 3-dimensional differentiable manifold M
sage: df.display()
df = -z^3*sin(x) dx + z^2*e^y dy + (3*z^2*cos(x) + 2*z*e^y) dz
sage: latex(df)
\mathrm{d}f
sage: df.parent()

```

```
Free module /\^1(M) of 1-forms on the 3-dimensional differentiable
manifold M
```

The result is cached, i.e. is not recomputed unless `f` is changed:

```
sage: f.differential() is df
True
```

Since the exterior derivative of a scalar field (considered a 0-form) is nothing but its differential, `exterior_derivative()` is an alias of `differential()`:

```
sage: df = f.exterior_derivative() ; df
1-form df on the 3-dimensional differentiable manifold M
sage: df.display()
df = -z^3*sin(x) dx + z^2*e^y dy + (3*z^2*cos(x) + 2*z*e^y) dz
sage: latex(df)
\mathrm{d}f
```

One may also use the global function `exterior_derivative()` or its alias `xder()` instead of the method `exterior_derivative()`:

```
sage: from sage.manifolds.utilities import xder
sage: xder(f) is f.exterior_derivative()
True
```

Differential computed on a chart that is not the default one:

```
sage: c_uvw.<u,v,w> = M.chart()
sage: g = M.scalar_field(u*v^2*w^3, c_uvw, name='g')
sage: dg = g.differential() ; dg
1-form dg on the 3-dimensional differentiable manifold M
sage: dg._components
{Coordinate frame (M, (d/du,d/dv,d/dw)): 1-index components w.r.t.
Coordinate frame (M, (d/du,d/dv,d/dw))}
sage: dg.comp(c_uvw.frame())[:, c_uvw]
[v^2*w^3, 2*u*v*w^3, 3*u*v^2*w^2]
sage: dg.display(c_uvw.frame(), c_uvw)
dg = v^2*w^3 du + 2*u*v*w^3 dv + 3*u*v^2*w^2 dw
```

The exterior derivative is nilpotent:

```
sage: ddf = df.exterior_derivative() ; ddf
2-form ddf on the 3-dimensional differentiable manifold M
sage: ddf == 0
True
sage: ddf[:] # for the incredule
[0 0 0]
[0 0 0]
[0 0 0]
sage: ddg = dg.exterior_derivative() ; ddg
2-form ddg on the 3-dimensional differentiable manifold M
sage: ddg == 0
True
```

#### `exterior_derivative()`

Return the differential of `self`.

OUTPUT:

- a *DiffForm* (or of *DiffFormParal* if the scalar field's domain is parallelizable) representing the 1-form that is the differential of the scalar field

EXAMPLES:

Differential of a scalar field on a 3-dimensional differentiable manifold:

```
sage: M = Manifold(3, 'M')
sage: c_xyz.<x,y,z> = M.chart()
sage: f = M.scalar_field(cos(x)*z^3 + exp(y)*z^2, name='f')
sage: df = f.differential() ; df
1-form df on the 3-dimensional differentiable manifold M
sage: df.display()
df = -z^3*sin(x) dx + z^2*e^y dy + (3*z^2*cos(x) + 2*z*e^y) dz
sage: latex(df)
\mathrm{d}f
sage: df.parent()
Free module /\^1(M) of 1-forms on the 3-dimensional differentiable
manifold M
```

The result is cached, i.e. is not recomputed unless *f* is changed:

```
sage: f.differential() is df
True
```

Since the exterior derivative of a scalar field (considered a 0-form) is nothing but its differential, `exterior_derivative()` is an alias of `differential()`:

```
sage: df = f.exterior_derivative() ; df
1-form df on the 3-dimensional differentiable manifold M
sage: df.display()
df = -z^3*sin(x) dx + z^2*e^y dy + (3*z^2*cos(x) + 2*z*e^y) dz
sage: latex(df)
\mathrm{d}f
```

One may also use the global function `exterior_derivative()` or its alias `xder()` instead of the method `exterior_derivative()`:

```
sage: from sage.manifolds.utilities import xder
sage: xder(f) is f.exterior_derivative()
True
```

Differential computed on a chart that is not the default one:

```
sage: c_uvw.<u,v,w> = M.chart()
sage: g = M.scalar_field(u*v^2*w^3, c_uvw, name='g')
sage: dg = g.differential() ; dg
1-form dg on the 3-dimensional differentiable manifold M
sage: dg._components
{Coordinate frame (M, (d/du,d/dv,d/dw)): 1-index components w.r.t.
 Coordinate frame (M, (d/du,d/dv,d/dw))}
sage: dg.comp(c_uvw.frame())[ :, c_uvw]
[v^2*w^3, 2*u*v*w^3, 3*u*v^2*w^2]
sage: dg.display(c_uvw.frame(), c_uvw)
dg = v^2*w^3 du + 2*u*v*w^3 dv + 3*u*v^2*w^2 dw
```

The exterior derivative is nilpotent:

```

sage: ddf = df.exterior_derivative() ; ddf
2-form ddf on the 3-dimensional differentiable manifold M
sage: ddf == 0
True
sage: ddf[:] # for the incredule
[0 0 0]
[0 0 0]
[0 0 0]
sage: ddg = dg.exterior_derivative() ; ddg
2-form ddg on the 3-dimensional differentiable manifold M
sage: ddg == 0
True

```

**hodge\_dual** (*metric*)

Compute the Hodge dual of the scalar field with respect to some metric.

If  $M$  is the domain of the scalar field (denoted by  $f$ ),  $n$  is the dimension of  $M$  and  $g$  is a pseudo-Riemannian metric on  $M$ , the *Hodge dual* of  $f$  w.r.t.  $g$  is the  $n$ -form  $*f$  defined by

$$*f = f\epsilon,$$

where  $\epsilon$  is the volume  $n$ -form associated with  $g$  (see `volume_form()`).

INPUT:

- `metric`: a pseudo-Riemannian metric defined on the same manifold as the current scalar field; must be an instance of `PseudoRiemannianMetric`

OUTPUT:

- the  $n$ -form  $*f$

EXAMPLES:

Hodge dual of a scalar field in the Euclidean space  $R^3$ :

```

sage: M = Manifold(3, 'M', start_index=1)
sage: X.<x,y,z> = M.chart()
sage: g = M.metric('g')
sage: g[1,1], g[2,2], g[3,3] = 1, 1, 1
sage: f = M.scalar_field(function('F')(x,y,z), name='f')
sage: sf = f.hodge_dual(g) ; sf
3-form *f on the 3-dimensional differentiable manifold M
sage: sf.display()
*f = F(x, y, z) dx/\dy/\dz
sage: ssf = sf.hodge_dual(g) ; ssf
Scalar field **f on the 3-dimensional differentiable manifold M
sage: ssf.display()
**f: M --> R
      (x, y, z) |--> F(x, y, z)
sage: ssf == f # must hold for a Riemannian metric
True

```

Instead of calling the method `hodge_dual()` on the scalar field, one can invoke the method `hodge_star()` of the metric:

```

sage: f.hodge_dual(g) == g.hodge_star(f)
True

```

**lie\_der** (*vector*)

Compute the Lie derivative with respect to a vector field.

In the present case (scalar field), the Lie derivative is equal to the scalar field resulting from the action of the vector field on the scalar field.

INPUT:

- *vector* – vector field with respect to which the Lie derivative is to be taken

OUTPUT:

- the scalar field that is the Lie derivative of the scalar field with respect to *vector*

EXAMPLES:

Lie derivative on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart()
sage: f = M.scalar_field(x^2*cos(y))
sage: v = M.vector_field(name='v')
sage: v[:] = (-y, x)
sage: f.lie_derivative(v)
Scalar field on the 2-dimensional differentiable manifold M
sage: f.lie_derivative(v).expr()
-x^3*sin(y) - 2*x*y*cos(y)
```

The result is cached:

```
sage: f.lie_derivative(v) is f.lie_derivative(v)
True
```

An alias is `lie_der`:

```
sage: f.lie_der(v) is f.lie_derivative(v)
True
```

Alternative expressions of the Lie derivative of a scalar field:

```
sage: f.lie_der(v) == v(f) # the vector acting on f
True
sage: f.lie_der(v) == f.differential()(v) # the differential of f acting on
↳ the vector
True
```

A vanishing Lie derivative:

```
sage: f.set_expr(x^2 + y^2)
sage: f.lie_der(v).display()
M --> R
(x, y) |--> 0
```

**lie\_derivative** (*vector*)

Compute the Lie derivative with respect to a vector field.

In the present case (scalar field), the Lie derivative is equal to the scalar field resulting from the action of the vector field on the scalar field.

INPUT:

- *vector* – vector field with respect to which the Lie derivative is to be taken

OUTPUT:

- the scalar field that is the Lie derivative of the scalar field with respect to `vector`

EXAMPLES:

Lie derivative on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart()
sage: f = M.scalar_field(x^2*cos(y))
sage: v = M.vector_field(name='v')
sage: v[:] = (-y, x)
sage: f.lie_derivative(v)
Scalar field on the 2-dimensional differentiable manifold M
sage: f.lie_derivative(v).expr()
-x^3*sin(y) - 2*x*y*cos(y)
```

The result is cached:

```
sage: f.lie_derivative(v) is f.lie_derivative(v)
True
```

An alias is `lie_der`:

```
sage: f.lie_der(v) is f.lie_derivative(v)
True
```

Alternative expressions of the Lie derivative of a scalar field:

```
sage: f.lie_der(v) == v(f) # the vector acting on f
True
sage: f.lie_der(v) == f.differential()(v) # the differential of f acting on
↳the vector
True
```

A vanishing Lie derivative:

```
sage: f.set_expr(x^2 + y^2)
sage: f.lie_der(v).display()
M --> R
(x, y) |--> 0
```

**tensor\_type()**

Return the tensor type of `self`, when the latter is considered as a tensor field on the manifold. This is always  $(0,0)$ .

OUTPUT:

- always  $(0,0)$

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart()
sage: f = M.scalar_field(x+2*y)
sage: f.tensor_type()
(0, 0)
```

## 2.5 Differentiable Maps and Curves

### 2.5.1 Sets of Morphisms between Differentiable Manifolds

The class `DifferentiableManifoldHomset` implements sets of morphisms between two differentiable manifolds over the same topological field  $K$  (in most applications,  $K = \mathbf{R}$  or  $K = \mathbf{C}$ ), a morphism being a *differentiable map* for the category of differentiable manifolds.

The subclass `DifferentiableCurveSet` is devoted to the specific case of differential curves, i.e. morphisms whose domain is an open interval of  $\mathbf{R}$ .

AUTHORS:

- Eric Gourgoulhon (2015): initial version
- Travis Scrimshaw (2016): review tweaks

REFERENCES:

- [Lee2013]
- [KN1963]

```
class sage.manifolds.differentiable.manifold_homset.DifferentiableCurveSet (domain,
                                                                              codomain,
                                                                              name=None,
                                                                              la-
                                                                              tex_name=None)
```

Bases: `sage.manifolds.differentiable.manifold_homset.DifferentiableManifoldHomset`

Set of differentiable curves in a differentiable manifold.

Given an open interval  $I$  of  $\mathbf{R}$  (possibly  $I = \mathbf{R}$ ) and a differentiable manifold  $M$  over  $\mathbf{R}$ , this is the set  $\text{Hom}(I, M)$  of morphisms (i.e. differentiable curves)  $I \rightarrow M$ .

INPUT:

- `domain` – `OpenInterval` if an open interval  $I \subset \mathbf{R}$  (domain of the morphisms), or `RealLine` if  $I = \mathbf{R}$
- `codomain` – `DifferentiableManifold`; differentiable manifold  $M$  (codomain of the morphisms)
- `name` – (default: `None`) string; name given to the set of curves; if `None`,  $\text{Hom}(I, M)$  will be used
- `latex_name` – (default: `None`) string; LaTeX symbol to denote the set of curves; if `None`,  $\text{Hom}(I, M)$  will be used

EXAMPLES:

Set of curves  $\mathbf{R} \rightarrow M$ , where  $M$  is a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: R.<t> = RealLine() ; R
Real number line R
sage: H = Hom(R, M) ; H
Set of Morphisms from Real number line R to 2-dimensional
differentiable manifold M in Category of smooth manifolds over Real
Field with 53 bits of precision
sage: H.category()
Category of homsets of topological spaces
sage: latex(H)
```

```

\mathrm{Hom}\left(\mathbf{R},M\right)
sage: H.domain()
Real number line R
sage: H.codomain()
2-dimensional differentiable manifold M

```

An element of  $H$  is a curve in  $M$ :

```

sage: c = H.an_element(); c
Curve in the 2-dimensional differentiable manifold M
sage: c.display()
R --> M
t |--> (x, y) = (1/(t^2 + 1) - 1/2, 0)

```

The test suite is passed:

```

sage: TestSuite(H).run()

```

The set of curves  $(0, 1) \rightarrow U$ , where  $U$  is an open subset of  $M$ :

```

sage: I = R.open_interval(0, 1) ; I
Real interval (0, 1)
sage: U = M.open_subset('U', coord_def={X: x^2+y^2<1}) ; U
Open subset U of the 2-dimensional differentiable manifold M
sage: H = Hom(I, U) ; H
Set of Morphisms from Real interval (0, 1) to Open subset U of the
2-dimensional differentiable manifold M in Join of Category of
subobjects of sets and Category of smooth manifolds over Real Field
with 53 bits of precision

```

An element of  $H$  is a curve in  $U$ :

```

sage: c = H.an_element() ; c
Curve in the Open subset U of the 2-dimensional differentiable
manifold M
sage: c.display()
(0, 1) --> U
t |--> (x, y) = (1/(t^2 + 1) - 1/2, 0)

```

The set of curves  $\mathbf{R} \rightarrow \mathbf{R}$  is a set of (manifold) endomorphisms:

```

sage: E = Hom(R, R) ; E
Set of Morphisms from Real number line R to Real number line R in
Category of smooth manifolds over Real Field with 53 bits of precision
sage: E.category()
Category of endsets of topological spaces
sage: E.is_endomorphism_set()
True
sage: E is End(R)
True

```

It is a monoid for the law of morphism composition:

```

sage: E in Monoids()
True

```

The identity element of the monoid is the identity map of  $\mathbf{R}$ :

```

sage: E.one()
Identity map Id_R of the Real number line R
sage: E.one() is R.identity_map()
True
sage: E.one().display()
Id_R: R --> R
      t |--> t

```

A “typical” element of the monoid:

```

sage: E.an_element().display()
R --> R
      t |--> 1/(t^2 + 1) - 1/2

```

The test suite is passed by E:

```

sage: TestSuite(E).run()

```

Similarly, the set of curves  $I \rightarrow I$  is a monoid, whose elements are (manifold) endomorphisms:

```

sage: EI = Hom(I, I) ; EI
Set of Morphisms from Real interval (0, 1) to Real interval (0, 1) in
Join of Category of subobjects of sets and
      Category of smooth manifolds over Real Field with 53 bits of precision
sage: EI.category()
Category of endsets of subobjects of sets and topological spaces
sage: EI is End(I)
True
sage: EI in Monoids()
True

```

The identity element and a “typical” element of this monoid:

```

sage: EI.one()
Identity map Id_(0, 1) of the Real interval (0, 1)
sage: EI.one().display()
Id_(0, 1): (0, 1) --> (0, 1)
           t |--> t
sage: EI.an_element().display()
(0, 1) --> (0, 1)
           t |--> 1/2/(t^2 + 1) + 1/4

```

The test suite is passed by EI:

```

sage: TestSuite(EI).run()

```

### Element

alias of `DifferentiableCurve`

**class** `sage.manifolds.differentiable.manifold_homset.DifferentiableManifoldHomset` (*domain*, *codomain*, *name=None*, *la-  
tex\_name=None*)

Bases: `sage.manifolds.manifold_homset.TopologicalManifoldHomset`

Set of differentiable maps between two differentiable manifolds.

Given two differentiable manifolds  $M$  and  $N$  over a topological field  $K$ , the class `DifferentiableManifoldHomset` implements the set  $\text{Hom}(M, N)$  of morphisms (i.e. differentiable maps)  $M \rightarrow N$ .

This is a Sage *parent* class, whose *element* class is `DiffMap`.

INPUT:

- `domain` – differentiable manifold  $M$  (domain of the morphisms), as an instance of `DifferentiableManifold`
- `codomain` – differentiable manifold  $N$  (codomain of the morphisms), as an instance of `DifferentiableManifold`
- `name` – (default: None) string; name given to the homset; if None,  $\text{Hom}(M, N)$  will be used
- `latex_name` – (default: None) string; LaTeX symbol to denote the homset; if None,  $\text{Hom}(M, N)$  will be used

EXAMPLES:

Set of differentiable maps between a 2-dimensional differentiable manifold and a 3-dimensional one:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: N = Manifold(3, 'N')
sage: Y.<u,v,w> = N.chart()
sage: H = Hom(M, N) ; H
Set of Morphisms from 2-dimensional differentiable manifold M to
3-dimensional differentiable manifold N in Category of smooth
manifolds over Real Field with 53 bits of precision
sage: type(H)
<class 'sage.manifolds.differentiable.manifold_homset.
↪DifferentiableManifoldHomset_with_category'>
sage: H.category()
Category of homsets of topological spaces
sage: latex(H)
\mathrm{Hom}\left(M,N\right)
sage: H.domain()
2-dimensional differentiable manifold M
sage: H.codomain()
3-dimensional differentiable manifold N
```

An element of  $H$  is a differentiable map from  $M$  to  $N$ :

```
sage: H.Element
<class 'sage.manifolds.differentiable.diff_map.DiffMap'>
sage: f = H.an_element() ; f
Differentiable map from the 2-dimensional differentiable manifold M to the
3-dimensional differentiable manifold N
sage: f.display()
M --> N
(x, y) |--> (u, v, w) = (0, 0, 0)
```

The test suite is passed:

```
sage: TestSuite(H).run()
```

When the codomain coincides with the domain, the homset is a set of *endomorphisms* in the category of differentiable manifolds:

```

sage: E = Hom(M, M) ; E
Set of Morphisms from 2-dimensional differentiable manifold M to
2-dimensional differentiable manifold M in Category of smooth
manifolds over Real Field with 53 bits of precision
sage: E.category()
Category of endsets of topological spaces
sage: E.is_endomorphism_set()
True
sage: E is End(M)
True

```

In this case, the homset is a monoid for the law of morphism composition:

```

sage: E in Monoids()
True

```

This was of course not the case for  $H = \text{Hom}(M, N)$ :

```

sage: H in Monoids()
False

```

The identity element of the monoid is of course the identity map of  $M$ :

```

sage: E.one()
Identity map Id_M of the 2-dimensional differentiable manifold M
sage: E.one() is M.identity_map()
True
sage: E.one().display()
Id_M: M --> M
      (x, y) |--> (x, y)

```

The test suite is passed by  $E$ :

```

sage: TestSuite(E).run()

```

This test suite includes more tests than in the case of  $H$ , since  $E$  has some extra structure (monoid).

### Element

alias of `DiffMap`

## 2.5.2 Differentiable Maps between Differentiable Manifolds

The class `DiffMap` implements differentiable maps from a differentiable manifold  $M$  to a differentiable manifold  $N$  over the same topological field  $K$  as  $M$  (in most applications,  $K = \mathbf{R}$  or  $K = \mathbf{C}$ ):

$$\Phi : M \longrightarrow N$$

### AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2013-2015): initial version

### REFERENCES:

- Chap. 1 of [KN1963]
- Chaps. 2 and 3 of [Lee2013]

```
class sage.manifolds.differentiable.diff_map.DiffMap (parent, coord_functions=None,
                                                    name=None, latex_name=None,
                                                    is_isomorphism=False,
                                                    is_identity=False)
```

Bases: `sage.manifolds.continuous_map.ContinuousMap`

Differentiable map between two differentiable manifolds.

This class implements differentiable maps of the type

$$\Phi : M \longrightarrow N$$

where  $M$  and  $N$  are differentiable manifolds over the same topological field  $K$  (in most applications,  $K = \mathbf{R}$  or  $K = \mathbf{C}$ ).

Differentiable maps are the *morphisms* of the *category* of differentiable manifolds. The set of all differentiable maps from  $M$  to  $N$  is therefore the homset between  $M$  and  $N$ , which is denoted by  $\text{Hom}(M, N)$ .

The class `DiffMap` is a Sage *element* class, whose *parent* class is `DifferentiableManifoldHomset`. It inherits from the class `ContinuousMap` since a differentiable map is obviously a continuous one.

INPUT:

- `parent` – homset  $\text{Hom}(M, N)$  to which the differentiable map belongs
- `coord_functions` – (default: `None`) if not `None`, must be a dictionary of the coordinate expressions (as lists (or tuples) of the coordinates of the image expressed in terms of the coordinates of the considered point) with the pairs of charts (`chart1`, `chart2`) as keys (`chart1` being a chart on  $M$  and `chart2` a chart on  $N$ ). If the dimension of the map's codomain is 1, a single coordinate expression can be passed instead of a tuple with a single element
- `name` – (default: `None`) name given to the differentiable map
- `latex_name` – (default: `None`) LaTeX symbol to denote the differentiable map; if `None`, the LaTeX symbol is set to `name`
- `is_isomorphism` – (default: `False`) determines whether the constructed object is a isomorphism (i.e. a diffeomorphism); if set to `True`, then the manifolds  $M$  and  $N$  must have the same dimension.
- `is_identity` – (default: `False`) determines whether the constructed object is the identity map; if set to `True`, then  $N$  must be  $M$  and the entry `coord_functions` is not used.

---

**Note:** If the information passed by means of the argument `coord_functions` is not sufficient to fully specify the differentiable map, further coordinate expressions, in other charts, can be subsequently added by means of the method `add_expr()`

---

EXAMPLES:

The standard embedding of the sphere  $S^2$  into  $\mathbf{R}^3$ :

```
sage: M = Manifold(2, 'S^2') # the 2-dimensional sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                                     intersection_name='W', restrictions1= x^2+y^2!=0,
....:                                     restrictions2= u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
```

```

sage: N = Manifold(3, 'R^3', r'\RR^3') # R^3
sage: c_cart.<X,Y,Z> = N.chart() # Cartesian coordinates on R^3
sage: Phi = M.diff_map(N,
....: {(c_xy, c_cart): [2*x/(1+x^2+y^2), 2*y/(1+x^2+y^2), (x^2+y^2-1)/(1+x^2+y^
↪2)],
....: (c_uv, c_cart): [2*u/(1+u^2+v^2), 2*v/(1+u^2+v^2), (1-u^2-v^2)/(1+u^2+v^
↪2)]}),
....: name='Phi', latex_name=r'\Phi')
sage: Phi
Differentiable map Phi from the 2-dimensional differentiable manifold
S^2 to the 3-dimensional differentiable manifold R^3
sage: Phi.parent()
Set of Morphisms from 2-dimensional differentiable manifold S^2 to
3-dimensional differentiable manifold R^3 in Category of smooth
manifolds over Real Field with 53 bits of precision
sage: Phi.parent() is Hom(M, N)
True
sage: type(Phi)
<class 'sage.manifolds.differentiable.manifold_homset.
↪DifferentiableManifoldHomset_with_category.element_class'>
sage: Phi.display()
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1),
(x^2 + y^2 - 1)/(x^2 + y^2 + 1))
on V: (u, v) |--> (X, Y, Z) = (2*u/(u^2 + v^2 + 1), 2*v/(u^2 + v^2 + 1),
-(u^2 + v^2 - 1)/(u^2 + v^2 + 1))

```

It is possible to create the map via the method `diff_map()` only in a single pair of charts: the argument `coord_functions` is then a mere list of coordinate expressions (and not a dictionary) and the arguments `chart1` and `chart2` have to be provided if the charts differ from the default ones on the domain and/or the codomain:

```

sage: Phi1 = M.diff_map(N, [2*x/(1+x^2+y^2), 2*y/(1+x^2+y^2),
....: (x^2+y^2-1)/(1+x^2+y^2)],
....: chart1=c_xy, chart2=c_cart, name='Phi',
....: latex_name=r'\Phi')

```

Since `c_xy` and `c_cart` are the default charts on respectively `M` and `N`, they can be omitted, so that the above declaration is equivalent to:

```

sage: Phi1 = M.diff_map(N, [2*x/(1+x^2+y^2), 2*y/(1+x^2+y^2),
....: (x^2+y^2-1)/(1+x^2+y^2)],
....: name='Phi', latex_name=r'\Phi')

```

With such a declaration, the differentiable map is only partially defined on the manifold  $S^2$ , being known in only one chart:

```

sage: Phi1.display()
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1),
(x^2 + y^2 - 1)/(x^2 + y^2 + 1))

```

The definition can be completed by means of the method `add_expr()`:

```

sage: Phi1.add_expr(c_uv, c_cart, [2*u/(1+u^2+v^2), 2*v/(1+u^2+v^2),
....: (1-u^2-v^2)/(1+u^2+v^2)])
sage: Phi1.display()

```

```
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1),
(x^2 + y^2 - 1)/(x^2 + y^2 + 1))
on V: (u, v) |--> (X, Y, Z) = (2*u/(u^2 + v^2 + 1), 2*v/(u^2 + v^2 + 1),
-(u^2 + v^2 - 1)/(u^2 + v^2 + 1))
```

At this stage, `Phi1` and `Phi` are fully equivalent:

```
sage: Phi1 == Phi
True
```

The test suite is passed:

```
sage: TestSuite(Phi).run()
sage: TestSuite(Phi1).run()
```

The map acts on points:

```
sage: np = M.point((0,0), chart=c_uv, name='N') # the North pole
sage: Phi(np)
Point Phi(N) on the 3-dimensional differentiable manifold R^3
sage: Phi(np).coord() # Cartesian coordinates
(0, 0, 1)
sage: sp = M.point((0,0), chart=c_xy, name='S') # the South pole
sage: Phi(sp).coord() # Cartesian coordinates
(0, 0, -1)
```

The differential  $d\Phi$  of the map  $\Phi$  at the North pole and at the South pole:

```
sage: Phi.differential(np)
Generic morphism:
  From: Tangent space at Point N on the 2-dimensional differentiable manifold S^2
  To:   Tangent space at Point Phi(N) on the 3-dimensional differentiable_
↳manifold R^3
sage: Phi.differential(sp)
Generic morphism:
  From: Tangent space at Point S on the 2-dimensional differentiable manifold S^2
  To:   Tangent space at Point Phi(S) on the 3-dimensional differentiable_
↳manifold R^3
```

The matrix of the linear map  $d\Phi_N$  with respect to the default bases of  $T_N S^2$  and  $T_{\Phi(N)} \mathbf{R}^3$ :

```
sage: Phi.differential(np).matrix()
[2 0]
[0 2]
[0 0]
```

the default bases being:

```
sage: Phi.differential(np).domain().default_basis()
Basis (d/du,d/dv) on the Tangent space at Point N on the 2-dimensional
differentiable manifold S^2
sage: Phi.differential(np).codomain().default_basis()
Basis (d/dX,d/dY,d/dZ) on the Tangent space at Point Phi(N) on the
3-dimensional differentiable manifold R^3
```

Differentiable maps can be composed by means of the operator `*`: let us introduce the map  $\mathbf{R}^3 \rightarrow \mathbf{R}^2$  corresponding to the projection from the point  $(X, Y, Z) = (0, 0, 1)$  onto the equatorial plane  $Z = 0$ :

```

sage: P = Manifold(2, 'R^2', r'\RR^2') # R^2 (equatorial plane)
sage: cP.<xP, yP> = P.chart()
sage: Psi = N.diff_map(P, (X/(1-Z), Y/(1-Z)), name='Psi',
....:                    latex_name=r'\Psi')
sage: Psi
Differentiable map Psi from the 3-dimensional differentiable manifold
R^3 to the 2-dimensional differentiable manifold R^2
sage: Psi.display()
Psi: R^3 --> R^2
      (X, Y, Z) |--> (xP, yP) = (-X/(Z - 1), -Y/(Z - 1))

```

Then we compose Psi with Phi, thereby getting a map  $S^2 \rightarrow \mathbf{R}^2$ :

```

sage: ster = Psi*Phi ; ster
Differentiable map from the 2-dimensional differentiable manifold S^2
to the 2-dimensional differentiable manifold R^2

```

Let us test on the South pole (sp) that ster is indeed the composite of Psi and Phi:

```

sage: ster(sp) == Psi(Phi(sp))
True

```

Actually ster is the stereographic projection from the North pole, as its coordinate expression reveals:

```

sage: ster.display()
S^2 --> R^2
on U: (x, y) |--> (xP, yP) = (x, y)
on V: (u, v) |--> (xP, yP) = (u/(u^2 + v^2), v/(u^2 + v^2))

```

If its codomain is 1-dimensional, a differentiable map must be defined by a single symbolic expression for each pair of charts, and not by a list/tuple with a single element:

```

sage: N = Manifold(1, 'N')
sage: c_N = N.chart('X')
sage: Phi = M.diff_map(N, {(c_xy, c_N): x^2+y^2,
....: (c_uv, c_N): 1/(u^2+v^2)}) # not ...[1/(u^2+v^2)] or (1/(u^2+v^2),)

```

An example of differentiable map  $\mathbf{R} \rightarrow \mathbf{R}^2$ :

```

sage: R = Manifold(1, 'R') # field R
sage: T.<t> = R.chart() # canonical chart on R
sage: R2 = Manifold(2, 'R^2') # R^2
sage: c_xy.<x,y> = R2.chart() # Cartesian coordinates on R^2
sage: Phi = R.diff_map(R2, [cos(t), sin(t)], name='Phi') ; Phi
Differentiable map Phi from the 1-dimensional differentiable manifold R
to the 2-dimensional differentiable manifold R^2
sage: Phi.parent()
Set of Morphisms from 1-dimensional differentiable manifold R to
2-dimensional differentiable manifold R^2 in Category of smooth
manifolds over Real Field with 53 bits of precision
sage: Phi.parent() is Hom(R, R2)
True
sage: Phi.display()
Phi: R --> R^2
      t |--> (x, y) = (cos(t), sin(t))

```

An example of diffeomorphism between the unit open disk and the Euclidean plane  $\mathbf{R}^2$ :

```

sage: D = R2.open_subset('D', coord_def={c_xy: x^2+y^2<1}) # the open unit disk
sage: Phi = D.diffeomorphism(R2, [x/sqrt(1-x^2-y^2), y/sqrt(1-x^2-y^2)],
.....:                               name='Phi', latex_name=r'\Phi')
sage: Phi
Diffeomorphism Phi from the Open subset D of the 2-dimensional
differentiable manifold R^2 to the 2-dimensional differentiable
manifold R^2
sage: Phi.parent()
Set of Morphisms from Open subset D of the 2-dimensional differentiable
manifold R^2 to 2-dimensional differentiable manifold R^2 in Join of
Category of subobjects of sets and Category of smooth manifolds over
Real Field with 53 bits of precision
sage: Phi.parent() is Hom(D, R2)
True
sage: Phi.display()
Phi: D --> R^2
(x, y) |--> (x, y) = (x/sqrt(-x^2 - y^2 + 1), y/sqrt(-x^2 - y^2 + 1))

```

The image of a point:

```

sage: p = D.point((1/2, 0))
sage: q = Phi(p) ; q
Point on the 2-dimensional differentiable manifold R^2
sage: q.coord()
(1/3*sqrt(3), 0)

```

The inverse diffeomorphism is computed by means of the method `inverse()`:

```

sage: Phi.inverse()
Diffeomorphism Phi^(-1) from the 2-dimensional differentiable manifold R^2
to the Open subset D of the 2-dimensional differentiable manifold R^2
sage: Phi.inverse().display()
Phi^(-1): R^2 --> D
(x, y) |--> (x, y) = (x/sqrt(x^2 + y^2 + 1), y/sqrt(x^2 + y^2 + 1))

```

Equivalently, one may use the notations  $\wedge(-1)$  or  $\sim$  to get the inverse:

```

sage: Phi^(-1) is Phi.inverse()
True
sage: ~Phi is Phi.inverse()
True

```

Check that  $\sim$ Phi is indeed the inverse of Phi:

```

sage: (~Phi)(q) == p
True
sage: Phi * ~Phi == R2.identity_map()
True
sage: ~Phi * Phi == D.identity_map()
True

```

The coordinate expression of the inverse diffeomorphism:

```

sage: (~Phi).display()
Phi^(-1): R^2 --> D
(x, y) |--> (x, y) = (x/sqrt(x^2 + y^2 + 1), y/sqrt(x^2 + y^2 + 1))

```

A special case of diffeomorphism: the identity map of the open unit disk:

```

sage: id = D.identity_map() ; id
Identity map Id_D of the Open subset D of the 2-dimensional
differentiable manifold R^2
sage: latex(id)
\mathrm{Id}_{D}
sage: id.parent()
Set of Morphisms from Open subset D of the 2-dimensional differentiable
manifold R^2 to Open subset D of the 2-dimensional differentiable
manifold R^2 in Join of Category of subobjects of sets and Category of
smooth manifolds over Real Field with 53 bits of precision
sage: id.parent() is Hom(D, D)
True
sage: id is Hom(D,D).one() # the identity element of the monoid Hom(D,D)
True

```

The identity map acting on a point:

```

sage: id(p)
Point on the 2-dimensional differentiable manifold R^2
sage: id(p) == p
True
sage: id(p) is p
True

```

The coordinate expression of the identity map:

```

sage: id.display()
Id_D: D --> D
      (x, y) |--> (x, y)

```

The identity map is its own inverse:

```

sage: id^(-1) is id
True
sage: ~id is id
True

```

### **differential** (*point*)

Return the differential of `self` at a given point.

If the differentiable map `self` is

$$\Phi : M \longrightarrow N,$$

where  $M$  and  $N$  are differentiable manifolds, the *differential* of  $\Phi$  at a point  $p \in M$  is the tangent space linear map:

$$d\Phi_p : T_p M \longrightarrow T_{\Phi(p)} N$$

defined by

$$\forall v \in T_p M, \quad d\Phi_p(v) : \begin{array}{ccc} C^k(N) & \longrightarrow & \mathbb{R} \\ f & \longmapsto & v(f \circ \Phi) \end{array}$$

INPUT:

- point – point  $p$  in the domain  $M$  of the differentiable map  $\Phi$

OUTPUT:

•  $d\Phi_p$ , the differential of  $\Phi$  at  $p$ , as a `FiniteRankFreeModuleMorphism`

EXAMPLES:

Differential of a differentiable map between a 2-dimensional manifold and a 3-dimensional one:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: N = Manifold(3, 'N')
sage: Y.<u,v,w> = N.chart()
sage: Phi = M.diff_map(N, {(X,Y): (x-2*y, x*y, x^2-y^3)}, name='Phi',
....:                        latex_name = r'\Phi')
sage: p = M.point((2,-1), name='p')
sage: dPhip = Phi.differential(p) ; dPhip
Generic morphism:
  From: Tangent space at Point p on the 2-dimensional differentiable manifold M
  To:   Tangent space at Point Phi(p) on the 3-dimensional differentiable manifold N
sage: latex(dPhip)
{\mathrm{d}\Phi}_{p}
sage: dPhip.parent()
Set of Morphisms from Tangent space at Point p on the 2-dimensional
differentiable manifold M to Tangent space at Point Phi(p) on the
3-dimensional differentiable manifold N in Category of finite
dimensional vector spaces over Symbolic Ring
```

The matrix of  $d\Phi_p$  w.r.t. to the default bases of  $T_pM$  and  $T_{\Phi(p)}N$ :

```
sage: dPhip.matrix()
[ 1 -2]
[-1  2]
[ 4 -3]
```

**differential\_functions** (*chart1=None, chart2=None*)

Return the coordinate expression of the differential of the differentiable map with respect to a pair of charts.

If the differentiable map is

$$\Phi : M \longrightarrow N,$$

where  $M$  and  $N$  are differentiable manifolds, the *differential* of  $\Phi$  at a point  $p \in M$  is the tangent space linear map:

$$d\Phi_p : T_pM \longrightarrow T_{\Phi(p)}N$$

defined by

$$\forall v \in T_pM, \quad d\Phi_p(v) : \begin{array}{ccc} C^k(N) & \longrightarrow & \mathbb{R}, \\ f & \longmapsto & v(f \circ \Phi). \end{array}$$

If the coordinate expression of  $\Phi$  is

$$y^i = Y^i(x^1, \dots, x^n), \quad 1 \leq i \leq m,$$

where  $(x^1, \dots, x^n)$  are coordinates of a chart on  $M$  and  $(y^1, \dots, y^m)$  are coordinates of a chart on  $\Phi(M)$ , the expression of the differential of  $\Phi$  with respect to these coordinates is

$$J_{ij} = \frac{\partial Y^i}{\partial x^j} \quad 1 \leq i \leq m, \quad 1 \leq j \leq n.$$

$J_{ij}|_p$  is then the matrix of the linear map  $d\Phi_p$  with respect to the bases of  $T_pM$  and  $T_{\Phi(p)}N$  associated to the above charts:

$$d\Phi_p \left( \frac{\partial}{\partial x^j} \Big|_p \right) = J_{ij}|_p \frac{\partial}{\partial y^i} \Big|_{\Phi(p)} .$$

INPUT:

- `chart1` – (default: `None`) chart on the domain  $M$  of  $\Phi$  (coordinates denoted by  $(x^j)$  above); if `None`, the domain's default chart is assumed
- `chart2` – (default: `None`) chart on the codomain of  $\Phi$  (coordinates denoted by  $(y^i)$  above); if `None`, the codomain's default chart is assumed

OUTPUT:

- the functions  $J_{ij}$  as a double array,  $J_{ij}$  being the element `[i][j]` represented by a *CoordFunction*

To get symbolic expressions, use the method `jacobian_matrix()` instead.

EXAMPLES:

Differential functions of a map between a 2-dimensional manifold and a 3-dimensional one:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: N = Manifold(3, 'N')
sage: Y.<u,v,w> = N.chart()
sage: Phi = M.diff_map(N, {(X,Y): (x-2*y, x*y, x^2-y^3)}, name='Phi',
....:                      latex_name = r'\Phi')
sage: J = Phi.differential_functions(X, Y) ; J
[  1      -2]
[  y       x]
[ 2*x  -3*y^2]
```

The result is cached:

```
sage: Phi.differential_functions(X, Y) is J
True
```

The elements of `J` are functions of the coordinates of the chart `X`:

```
sage: J[2][0]
2*x
sage: type(J[2][0])
<class 'sage.manifolds.coord_func_symb.CoordFunctionSymbRing_with_category.
↳element_class'>
sage: J[2][0].display()
(x, y) |--> 2*x
```

In contrast, the method `jacobian_matrix()` leads directly to symbolic expressions:

```
sage: JJ = Phi.jacobian_matrix(X,Y) ; JJ
[  1      -2]
[  y       x]
[ 2*x  -3*y^2]
sage: JJ[2,0]
2*x
sage: type(JJ[2,0])
<type 'sage.symbolic.expression.Expression'>
```

```
sage: bool( JJ[2,0] == J[2][0].expr() )
True
```

**jacobian\_matrix** (*chart1=None, chart2=None*)

Return the Jacobian matrix resulting from the coordinate expression of the differentiable map with respect to a pair of charts.

If  $\Phi$  is the current differentiable map and its coordinate expression is

$$y^i = Y^i(x^1, \dots, x^n), \quad 1 \leq i \leq m,$$

where  $(x^1, \dots, x^n)$  are coordinates of a chart  $X$  on the domain of  $\Phi$  and  $(y^1, \dots, y^m)$  are coordinates of a chart  $Y$  on the codomain of  $\Phi$ , the *Jacobian matrix* of the differentiable map  $\Phi$  w.r.t. to charts  $X$  and  $Y$  is

$$J = \left( \frac{\partial Y^i}{\partial x^j} \right)_{\substack{1 \leq i \leq m \\ 1 \leq j \leq n}},$$

where  $i$  is the row index and  $j$  the column one.

INPUT:

- `chart1` – (default: `None`) chart  $X$  on the domain of  $\Phi$ ; if none is provided, the domain's default chart is assumed
- `chart2` – (default: `None`) chart  $Y$  on the codomain of  $\Phi$ ; if none is provided, the codomain's default chart is assumed

OUTPUT:

- the matrix  $J$  defined above

EXAMPLES:

Jacobian matrix of a map between a 2-dimensional manifold and a 3-dimensional one:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: N = Manifold(3, 'N')
sage: Y.<u,v,w> = N.chart()
sage: Phi = M.diff_map(N, {(X,Y): (x-2*y, x*y, x^2-y^3)}, name='Phi',
....:                      latex_name=r'\Phi')
sage: Phi.display()
Phi: M --> N
      (x, y) |--> (u, v, w) = (x - 2*y, x*y, -y^3 + x^2)
sage: J = Phi.jacobian_matrix(X, Y) ; J
[  1  -2]
[  y  x ]
[ 2*x -3*y^2]
sage: J.parent()
Full MatrixSpace of 3 by 2 dense matrices over Symbolic Ring
```

**pullback** (*tensor*)

Pullback operator associated with `self`.

In what follows, let  $\Phi$  denote a differentiable map,  $M$  its domain and  $N$  its codomain.

INPUT:

- `tensor` – *TensorField*; a fully covariant tensor field  $T$  on  $N$ , i.e. a tensor field of type  $(0, p)$ , with  $p$  a positive or zero integer; the case  $p = 0$  corresponds to a scalar field

OUTPUT:

- a *TensorField* representing a fully covariant tensor field on  $M$  that is the pullback of  $T$  by  $\Phi$

EXAMPLES:

Pullback on  $S^2$  of a scalar field defined on  $R^3$ :

```
sage: M = Manifold(2, 'S^2', start_index=1)
sage: U = M.open_subset('U') # the complement of a meridian (domain of
↳spherical coordinates)
sage: c_spher.<th,ph> = U.chart(r'th:(0,pi):\theta ph:(0,2*pi):\phi') #
↳spherical coord. on U
sage: N = Manifold(3, 'R^3', r'\RR^3', start_index=1)
sage: c_cart.<x,y,z> = N.chart() # Cartesian coord. on R^3
sage: Phi = U.diff_map(N, (sin(th)*cos(ph), sin(th)*sin(ph), cos(th)),
....:                      name='Phi', latex_name=r'\Phi')
sage: f = N.scalar_field(x*y*z, name='f') ; f
Scalar field f on the 3-dimensional differentiable manifold R^3
sage: f.display()
f: R^3 --> R
   (x, y, z) |--> x*y*z
sage: pf = Phi.pullback(f) ; pf
Scalar field Phi_*(f) on the Open subset U of the 2-dimensional
differentiable manifold S^2
sage: pf.display()
Phi_*(f): U --> R
   (th, ph) |--> cos(ph)*cos(th)*sin(ph)*sin(th)^2
```

Pullback on  $S^2$  of the standard Euclidean metric on  $R^3$ :

```
sage: g = N.sym_bilin_form_field('g')
sage: g[1,1], g[2,2], g[3,3] = 1, 1, 1
sage: g.display()
g = dx*dx + dy*dy + dz*dz
sage: pg = Phi.pullback(g) ; pg
Field of symmetric bilinear forms Phi_*(g) on the Open subset U of
the 2-dimensional differentiable manifold S^2
sage: pg.display()
Phi_*(g) = dth*dth + sin(th)^2 dph*dph
```

Pullback on  $S^2$  of a 3-form on  $R^3$ :

```
sage: a = N.diff_form(3, 'A')
sage: a[1,2,3] = f
sage: a.display()
A = x*y*z dx/\dy/\dz
sage: pa = Phi.pullback(a) ; pa
3-form Phi_*(A) on the Open subset U of the 2-dimensional
differentiable manifold S^2
sage: pa.display() # should be zero (as any 3-form on a 2-dimensional
↳manifold)
Phi_*(A) = 0
```

### pushforward (tensor)

Pushforward operator associated with `self`.

In what follows, let  $\Phi$  denote the differentiable map,  $M$  its domain and  $N$  its codomain.

INPUT:

- `tensor` – *TensorField*; a fully contrariant tensor field  $T$  on  $M$ , i.e. a tensor field of type  $(p, 0)$ , with  $p$  a positive integer

OUTPUT:

- a *TensorField* representing a fully contravariant tensor field along  $M$  with values in  $N$ , which is the pushforward of  $T$  by  $\Phi$

EXAMPLES:

Pushforward of a vector field on the 2-sphere  $S^2$  to the Euclidean 3-space  $\mathbf{R}^3$ , via the standard embedding of  $S^2$ :

```
sage: S2 = Manifold(2, 'S^2', start_index=1)
sage: U = S2.open_subset('U') # domain of spherical coordinates
sage: spher.<th,ph> = U.chart(r'th:(0,pi):\theta ph:(0,2*pi):\phi')
sage: R3 = Manifold(3, 'R^3', start_index=1)
sage: cart.<x,y,z> = R3.chart()
sage: Phi = U.diff_map(R3, {(spher, cart): [sin(th)*cos(ph),
.....:   sin(th)*sin(ph), cos(th)]}, name='Phi', latex_name=r'\Phi')
sage: v = U.vector_field(name='v')
sage: v[:] = 0, 1
sage: v.display()
v = d/dph
sage: pv = Phi.pushforward(v); pv
Vector field Phi^*(v) along the Open subset U of the 2-dimensional
differentiable manifold S^2 with values on the 3-dimensional
differentiable manifold R^3
sage: pv.display()
Phi^*(v) = -sin(ph)*sin(th) d/dx + cos(ph)*sin(th) d/dy
```

Pushforward of a vector field on the real line to the  $\mathbf{R}^3$ , via a helix embedding:

```
sage: R.<t> = RealLine()
sage: Psi = R.diff_map(R3, [cos(t), sin(t), t], name='Psi',
.....:   latex_name=r'\Psi')
sage: u = R.vector_field(name='u')
sage: u[0] = 1
sage: u.display()
u = d/dt
sage: pu = Psi.pushforward(u); pu
Vector field Psi^*(u) along the Real number line R with values on
the 3-dimensional differentiable manifold R^3
sage: pu.display()
Psi^*(u) = -sin(t) d/dx + cos(t) d/dy + d/dz
```

### 2.5.3 Curves in Manifolds

Given a differentiable manifold  $M$ , a *differentiable curve*  $\text{curve}$  in  $M$  is a differentiable mapping

$$\gamma : I \longrightarrow M,$$

where  $I$  is an interval of  $\mathbf{R}$ .

Differentiable curves are implemented by *DifferentiableCurve*.

AUTHORS:

- Eric Gourgoulhon (2015): initial version

- Travis Scrimshaw (2016): review tweaks

## REFERENCES:

- Chap. 1 of [KN1963]
- Chap. 3 of [Lee2013]

```
class sage.manifolds.differentiable.curve.DifferentiableCurve (parent,
                                                             coord_expression=None,
                                                             name=None,
                                                             latex_name=None,
                                                             is_isomorphism=False,
                                                             is_identity=False)
```

Bases: `sage.manifolds.differentiable.diff_map.DiffMap`

Curve in a differentiable manifold.

Given a differentiable manifold  $M$ , a *differentiable curve* in  $M$  is a differentiable map

$$\gamma : I \longrightarrow M,$$

where  $I$  is an interval of  $\mathbf{R}$ .

## INPUT:

- `parent` – `DifferentiableCurveSet` the set of curves  $\text{Hom}(I, M)$  to which the curve belongs
- `coord_expression` – (default: `None`) dictionary (possibly empty) of the functions of the curve parameter  $t$  expressing the curve in various charts of  $M$ , the keys of the dictionary being the charts and the values being lists or tuples of  $n$  symbolic expressions of  $t$ , where  $n$  is the dimension of  $M$
- `name` – (default: `None`) string; symbol given to the curve
- `latex_name` – (default: `None`) string; LaTeX symbol to denote the the curve; if none is provided, `name` will be used
- `is_isomorphism` – (default: `False`) determines whether the constructed object is a diffeomorphism; if set to `True`, then  $M$  must have dimension one
- `is_identity` – (default: `False`) determines whether the constructed object is the identity map; if set to `True`, then  $M$  must be the interval  $I$

## EXAMPLES:

The lemniscate of Geronon in the 2-dimensional Euclidean plane:

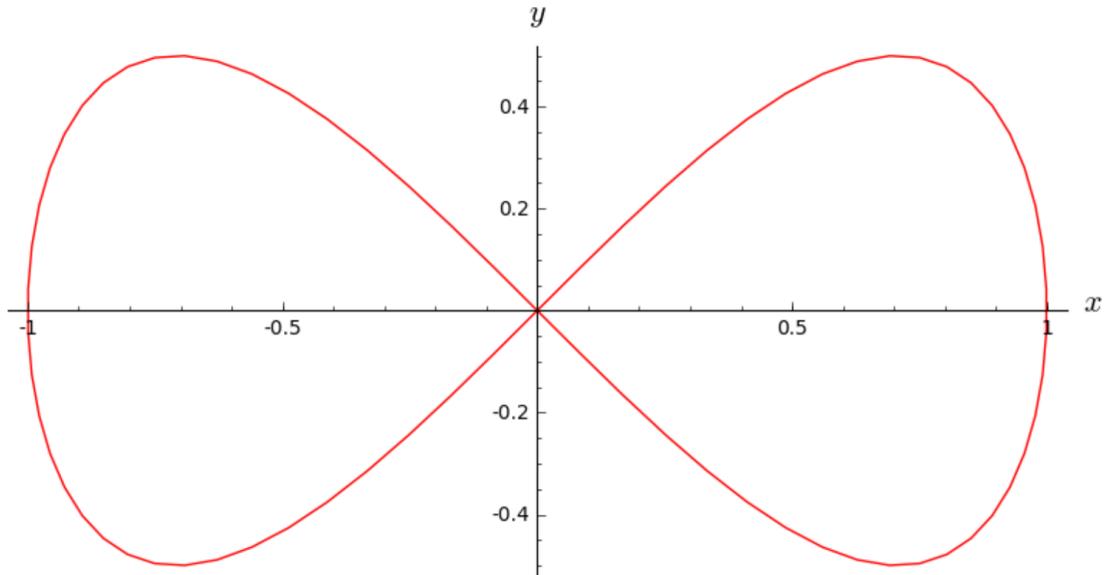
```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: t = var('t')
sage: c = M.curve({X: [sin(t), sin(2*t)/2]}, (t, 0, 2*pi), name='c') ; c
Curve c in the 2-dimensional differentiable manifold M
sage: type(c)
<class 'sage.manifolds.differentiable.manifold_homset.DifferentiableCurveSet_with_
↪category.element_class'>
```

Instead of declaring the parameter  $t$  as a symbolic variable by means of `var('t')`, it is equivalent to get it as the canonical coordinate of the real number line (see `RealLine`):

```
sage: R.<t> = RealLine()
sage: c = M.curve({X: [sin(t), sin(2*t)/2]}, (t, 0, 2*pi), name='c') ; c
Curve c in the 2-dimensional differentiable manifold M
```

A graphical view of the curve is provided by the method `plot()`:

```
sage: c.plot(aspect_ratio=1)
Graphics object consisting of 1 graphics primitive
```



Curves are considered as (manifold) morphisms from real intervals to differentiable manifolds:

```
sage: c.parent()
Set of Morphisms from Real interval (0, 2*pi) to 2-dimensional
differentiable manifold M in Join of Category of subobjects of sets
and Category of smooth manifolds over Real Field with 53 bits of
precision
sage: I = R.open_interval(0, 2*pi)
sage: c.parent() is Hom(I, M)
True
sage: c.domain()
Real interval (0, 2*pi)
sage: c.domain() is I
True
sage: c.codomain()
2-dimensional differentiable manifold M
```

Accordingly, all methods of *DiffMap* are available for them. In particular, the method `display()` shows the coordinate representations in various charts of manifold *M*:

```
sage: c.display()
c: (0, 2*pi) --> M
t |--> (x, y) = (sin(t), 1/2*sin(2*t))
```

Another map method is using the usual call syntax, which returns the image of a point in the curve's domain:

```
sage: t0 = pi/2
sage: I(t0)
Point on the Real number line R
sage: c(I(t0))
Point on the 2-dimensional differentiable manifold M
sage: c(I(t0)).coord(X)
(1, 0)
```

For curves, the value of the parameter, instead of the corresponding point in the real line manifold, can be passed directly:

```
sage: c(t0)
Point c(1/2*pi) on the 2-dimensional differentiable manifold M
sage: c(t0).coord(X)
(1, 0)
sage: c(t0) == c(I(t0))
True
```

Instead of a dictionary of coordinate expressions, the curve can be defined by a single coordinate expression in a given chart:

```
sage: c = M.curve([sin(t), sin(2*t)/2], (t, 0, 2*pi), chart=X, name='c') ; c
Curve c in the 2-dimensional differentiable manifold M
sage: c.display()
c: (0, 2*pi) --> M
   t |--> (x, y) = (sin(t), 1/2*sin(2*t))
```

Since  $X$  is the default chart on  $M$ , it can be omitted:

```
sage: c = M.curve([sin(t), sin(2*t)/2], (t, 0, 2*pi), name='c') ; c
Curve c in the 2-dimensional differentiable manifold M
sage: c.display()
c: (0, 2*pi) --> M
   t |--> (x, y) = (sin(t), 1/2*sin(2*t))
```

Note that a curve in  $M$  can also be created as a differentiable map  $I \rightarrow M$ :

```
sage: c1 = I.diff_map(M, coord_functions={X: [sin(t), sin(2*t)/2]},
....:                      name='c') ; c1
Curve c in the 2-dimensional differentiable manifold M
sage: c1.parent() is c.parent()
True
sage: c1 == c
True
```

LaTeX symbols representing a curve:

```
sage: c = M.curve([sin(t), sin(2*t)/2], (t, 0, 2*pi))
sage: latex(c)
\mbox{Curve in the 2-dimensional differentiable manifold M}
sage: c = M.curve([sin(t), sin(2*t)/2], (t, 0, 2*pi), name='c')
sage: latex(c)
c
sage: c = M.curve([sin(t), sin(2*t)/2], (t, 0, 2*pi), name='c',
....:              latex_name=r'\gamma')
```

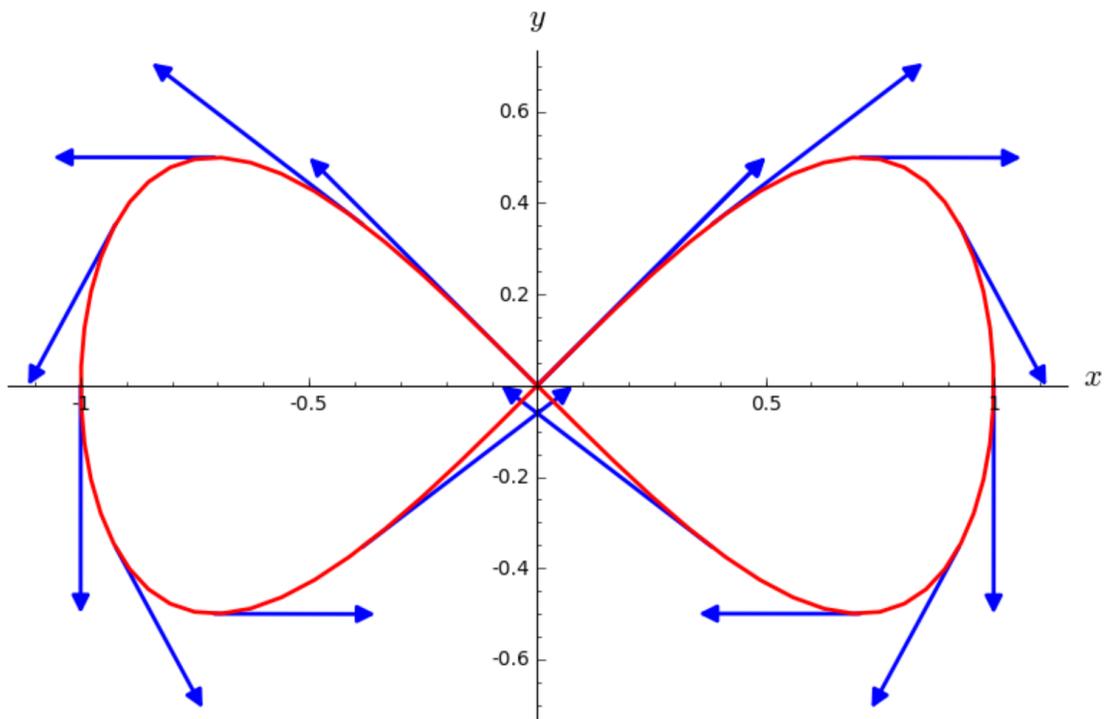
```
sage: latex(c)
\gamma
```

The curve's tangent vector field (velocity vector):

```
sage: v = c.tangent_vector_field() ; v
Vector field c' along the Real interval (0, 2*pi) with values on the
2-dimensional differentiable manifold M
sage: v.display()
c' = cos(t) d/dx + (2*cos(t)^2 - 1) d/dy
```

Plot of the curve and its tangent vector field:

```
sage: show(c.plot(thickness=2, aspect_ratio=1) +
.....:       v.plot(chart=X, number_values=17, scale=0.5))
```



Value of the tangent vector field at  $t = \pi$ :

```
sage: v.at(R(pi))
Tangent vector c' at Point on the 2-dimensional differentiable
manifold M
sage: v.at(R(pi)) in M.tangent_space(c(R(pi)))
True
sage: v.at(R(pi)).display()
c' = -d/dx + d/dy
```

Curves  $\mathbf{R} \rightarrow \mathbf{R}$  can be composed: the operator  $\circ$  is given by  $*$ :

```

sage: f = R.curve(t^2, (t,-oo,+oo))
sage: g = R.curve(cos(t), (t,-oo,+oo))
sage: s = g*f ; s
Differentiable map from the Real number line R to itself
sage: s.display()
R --> R
  t |--> cos(t^2)
sage: s = f*g ; s
Differentiable map from the Real number line R to itself
sage: s.display()
R --> R
  t |--> cos(t)^2

```

**coord\_expr** (*chart=None*)

Return the coordinate functions expressing the curve in a given chart.

## INPUT:

- *chart* – (default: None) chart on the curve’s codomain; if None, the codomain’s default chart is assumed

## OUTPUT:

- symbolic expression representing the curve in the above chart

## EXAMPLES:

Cartesian and polar expression of a curve in the Euclidean plane:

```

sage: M = Manifold(2, 'R^2', r'\RR^2') # the Euclidean plane R^2
sage: c_xy.<x,y> = M.chart() # Cartesian coordinate on R^2
sage: U = M.open_subset('U', coord_def={c_xy: (y!=0, x<0)}) # the complement
↳ of the segment y=0 and x>0
sage: c_cart = c_xy.restrict(U) # Cartesian coordinates on U
sage: c_spher.<r,ph> = U.chart(r'r:(0,+oo) ph:(0,2*pi):\phi') # spherical
↳ coordinates on U
sage: # Links between spherical coordinates and Cartesian ones:
sage: ch_cart_spher = c_cart.transition_map(c_spher, [sqrt(x*x+y*y), atan2(y,
↳ x)])
sage: ch_cart_spher.set_inverse(r*cos(ph), r*sin(ph))
sage: R.<t> = RealLine()
sage: c = U.curve({c_spher: (1,t)}, (t, 0, 2*pi), name='c')
sage: c.coord_expr(c_spher)
(1, t)
sage: c.coord_expr(c_cart)
(cos(t), sin(t))

```

Since *c\_cart* is the default chart on *U*, it can be omitted:

```

sage: c.coord_expr()
(cos(t), sin(t))

```

Cartesian expression of a cardioid:

```

sage: c = U.curve({c_spher: (2*(1+cos(t)), t)}, (t, 0, 2*pi), name='c')
sage: c.coord_expr(c_cart)
(2*cos(t)^2 + 2*cos(t), 2*(cos(t) + 1)*sin(t))

```

**plot** (*chart=None, ambient\_coords=None, mapping=None, prange=None, include\_end\_point=(True, True), end\_point\_offset=(0.001, 0.001), parameters=None, color='red', style='-', label\_axes=True, aspect\_ratio='automatic', max\_range=8, plot\_points=75, thickness=1, \*\*kwds*)  
 Plot the current curve in a Cartesian graph based on the coordinates of some ambient chart.

The curve is drawn in terms of two (2D graphics) or three (3D graphics) coordinates of a given chart, called hereafter the *ambient chart*. The ambient chart's domain must overlap with the curve's codomain or with the codomain of the composite curve  $\Phi \circ c$ , where  $c$  is the current curve and  $\Phi$  some manifold differential map (argument mapping below).

INPUT:

- **chart** – (default: None) the ambient chart (see above); if None, the default chart of the codomain of the curve (or of the curve composed with  $\Phi$ ) is used
- **ambient\_coords** – (default: None) tuple containing the 2 or 3 coordinates of the ambient chart in terms of which the plot is performed; if None, all the coordinates of the ambient chart are considered
- **mapping** – (default: None) differentiable mapping  $\Phi$  (instance of *DiffMap*) providing the link between the curve and the ambient chart *chart* (cf. above); if None, the ambient chart is supposed to be defined on the codomain of the curve.
- **prange** – (default: None) range of the curve parameter for the plot; if None, the entire parameter range declared during the curve construction is considered (with  $-\infty$  replaced by  $-\text{max\_range}$  and  $+\infty$  by  $\text{max\_range}$ )
- **include\_end\_point** – (default: (True, True)) determines whether the end points of *prange* are included in the plot
- **end\_point\_offset** – (default: (0.001, 0.001)) offsets from the end points when they are not included in the plot: if `include_end_point[0] == False`, the minimal value of the curve parameter used for the plot is `prange[0] + end_point_offset[0]`, while if `include_end_point[1] == False`, the maximal value is `prange[1] - end_point_offset[1]`.
- **max\_range** – (default: 8) numerical value substituted to  $+\infty$  if the latter is the upper bound of the parameter range; similarly  $-\text{max\_range}$  is the numerical valued substituted for  $-\infty$
- **parameters** – (default: None) dictionary giving the numerical values of the parameters that may appear in the coordinate expression of the curve
- **color** – (default: 'red') color of the drawn curve
- **style** – (default: '-') color of the drawn curve; NB: *style* is effective only for 2D plots
- **thickness** – (default: 1) thickness of the drawn curve
- **plot\_points** – (default: 75) number of points to plot the curve
- **label\_axes** – (default: True) boolean determining whether the labels of the coordinate axes of *chart* shall be added to the graph; can be set to `False` if the graph is 3D and must be superposed with another graph.
- **aspect\_ratio** – (default: 'automatic') aspect ratio of the plot; the default value ('automatic') applies only for 2D plots; for 3D plots, the default value is 1 instead

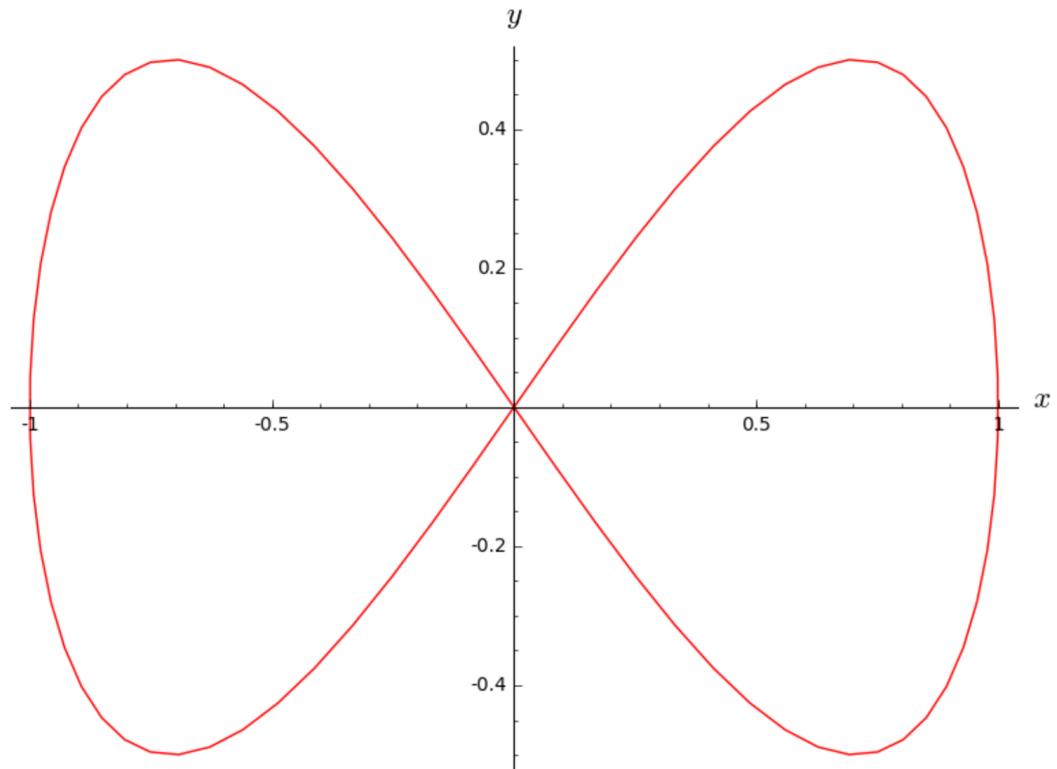
OUTPUT:

- a graphic object, either an instance of *Graphics* for a 2D plot (i.e. based on 2 coordinates of *chart*) or an instance of *Graphics3d* for a 3D plot (i.e. based on 3 coordinates of *chart*)

EXAMPLES:

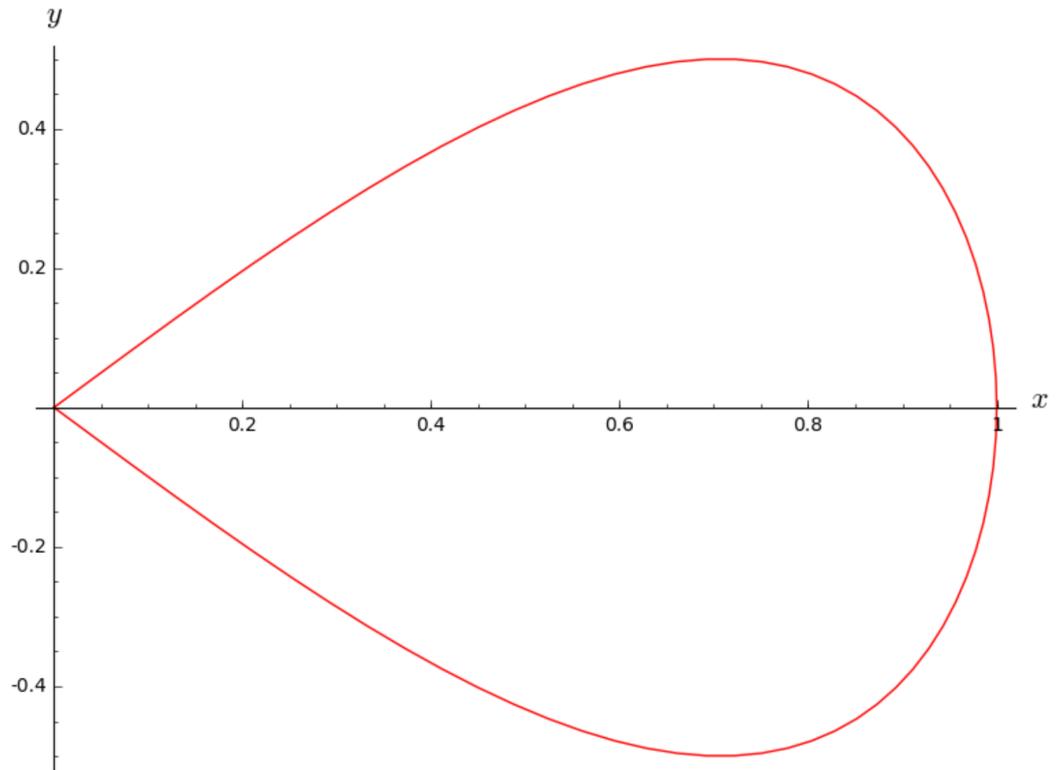
Plot of the lemniscate of Gerono:

```
sage: R2 = Manifold(2, 'R^2')
sage: X.<x,y> = R2.chart()
sage: R.<t> = RealLine()
sage: c = R2.curve([sin(t), sin(2*t)/2], (t, 0, 2*pi), name='c')
sage: c.plot() # 2D plot
Graphics object consisting of 1 graphics primitive
```



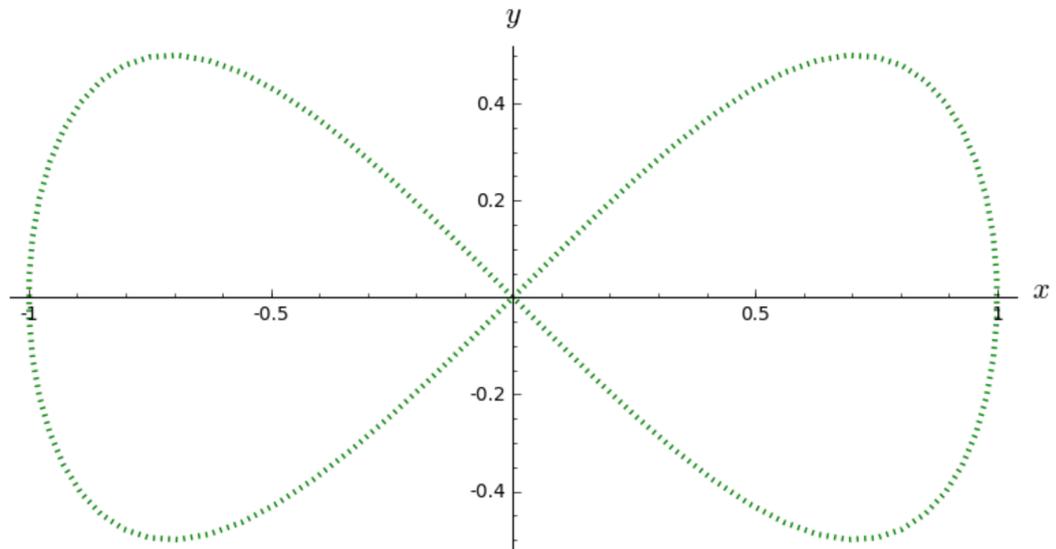
Plot for a subinterval of the curve's domain:

```
sage: c.plot(prange=(0,pi))
Graphics object consisting of 1 graphics primitive
```



Plot with various options:

```
sage: c.plot(color='green', style=':', thickness=3, aspect_ratio=1)
Graphics object consisting of 1 graphics primitive
```

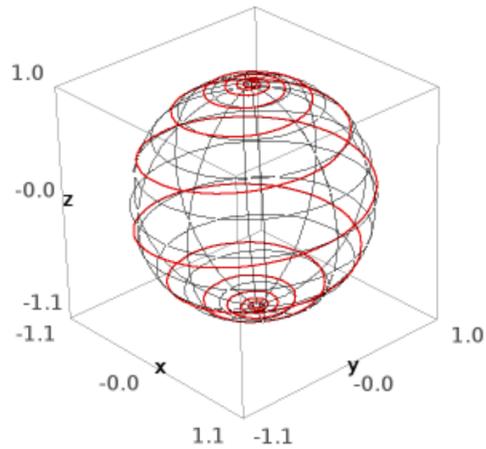


Plot via a mapping to another manifold: loxodrome of a sphere viewed in  $\mathbf{R}^3$ :

```

sage: S2 = Manifold(2, 'S^2')
sage: U = S2.open_subset('U')
sage: XS.<th,ph> = U.chart(r'th:(0,pi):\theta ph:(0,2*pi):\phi')
sage: R3 = Manifold(3, 'R^3')
sage: X3.<x,y,z> = R3.chart()
sage: F = S2.diff_map(R3, {(XS, X3): [sin(th)*cos(ph),
....:                               sin(th)*sin(ph), cos(th)]}, name='F')
sage: F.display()
F: S^2 --> R^3
on U: (th, ph) |--> (x, y, z) = (cos(ph)*sin(th), sin(ph)*sin(th), cos(th))
sage: c = S2.curve([2*atan(exp(-t/10)), t], (t, -oo, +oo), name='c')
sage: graph_c = c.plot(mapping=F, max_range=40,
....:                  plot_points=200, thickness=2, label_axes=False) # 3D
↳plot
sage: graph_S2 = XS.plot(X3, mapping=F, number_values=11, color='black') #
↳plot of the sphere
sage: show(graph_c + graph_S2) # the loxodrome + the sphere

```

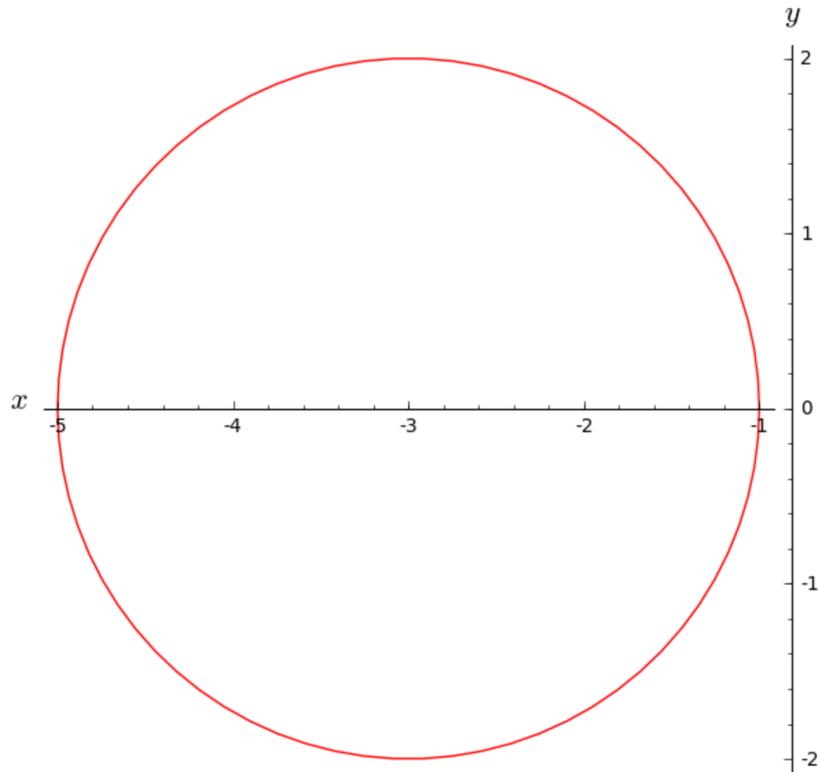


Example of use of the argument `parameters`: we define a curve with some symbolic parameters `a` and `b`:

```
sage: a, b = var('a b')
sage: c = R2.curve([a*cos(t) + b, a*sin(t)], (t, 0, 2*pi), name='c')
```

To make a plot, we set specific values for `a` and `b` by means of the Python dictionary parameters:

```
sage: c.plot(parameters={a: 2, b: -3}, aspect_ratio=1)
Graphics object consisting of 1 graphics primitive
```



**tangent\_vector\_field** (*name=None, latex\_name=None*)

Return the tangent vector field to the curve (velocity vector).

INPUT:

- *name* – (default: *None*) string; symbol given to the tangent vector field; if none is provided, the primed curve symbol (if any) will be used
- *latex\_name* – (default: *None*) string; LaTeX symbol to denote the tangent vector field; if *None* then (i) if *name* is *None* as well, the primed curve LaTeX symbol (if any) will be used or (ii) if *name* is not *None*, *name* will be used

OUTPUT:

- the tangent vector field, as an instance of *VectorField*

EXAMPLES:

Tangent vector field to a circle curve in  $\mathbf{R}^2$ :

```
sage: M = Manifold(2, 'R^2')
sage: X.<x,y> = M.chart()
sage: R.<t> = RealLine()
sage: c = M.curve([cos(t), sin(t)], (t, 0, 2*pi), name='c')
sage: v = c.tangent_vector_field() ; v
Vector field c' along the Real interval (0, 2*pi) with values on
the 2-dimensional differentiable manifold R^2
sage: v.display()
c' = -sin(t) d/dx + cos(t) d/dy
sage: latex(v)
{c'}
```

```

sage: v.parent()
Free module X((0, 2*pi),c) of vector fields along the Real interval
(0, 2*pi) mapped into the 2-dimensional differentiable manifold R^2

```

Value of the tangent vector field for some specific value of the curve parameter ( $t = \pi$ ):

```

sage: R(pi) in c.domain() # pi in (0, 2*pi)
True
sage: vp = v.at(R(pi)) ; vp
Tangent vector c' at Point on the 2-dimensional differentiable
manifold R^2
sage: vp.parent() is M.tangent_space(c(R(pi)))
True
sage: vp.display()
c' = -d/dy

```

Tangent vector field to a curve in a non-parallelizable manifold (the 2-sphere  $S^2$ ): first, we introduce the 2-sphere:

```

sage: M = Manifold(2, 'M') # the 2-dimensional sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                                     intersection_name='W', restrictions1= x^2+y^2!=0,
....:                                     restrictions2= u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: W = U.intersection(V)
sage: A = W.open_subset('A', coord_def={c_xy.restrict(W): (y!=0, x<0)})
sage: c_spher.<th,ph> = A.chart(r'th:(0,pi):\theta ph:(0,2*pi):\phi') #
↳spherical coordinates
sage: spher_to_xy = c_spher.transition_map(c_xy.restrict(A),
....:                                     (sin(th)*cos(ph)/(1-cos(th)), sin(th)*sin(ph)/(1-cos(th))) )
sage: spher_to_xy.set_inverse(2*atan(1/sqrt(x^2+y^2)), atan2(y, x),
↳check=False)

```

Then we define a curve (a loxodrome) by its expression in terms of spherical coordinates and evaluate the tangent vector field:

```

sage: R.<t> = RealLine()
sage: c = M.curve({c_spher: [2*atan(exp(-t/10)), t]}, (t, -oo, +oo),
....:                                     name='c') ; c
Curve c in the 2-dimensional differentiable manifold M
sage: vc = c.tangent_vector_field() ; vc
Vector field c' along the Real number line R with values on
the 2-dimensional differentiable manifold M
sage: vc.parent()
Module X(R,c) of vector fields along the Real number line R
mapped into the 2-dimensional differentiable manifold M
sage: vc.display(c_spher.frame().along(c.restrict(R,A)))
c' = -1/5*e^(1/10*t)/(e^(1/5*t) + 1) d/dth + d/dph

```

## 2.6 Tangent Spaces

### 2.6.1 Tangent Spaces

The class `TangentSpace` implements tangent vector spaces to a differentiable manifold.

AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2014-2015): initial version
- Travis Scrimshaw (2016): review tweaks

REFERENCES:

- Chap. 3 of [Lee2013]

**class** `sage.manifolds.differentiable.tangent_space.TangentSpace` (*point*)  
**Bases:** `sage.tensor.modules.finite_rank_free_module.FiniteRankFreeModule`

Tangent space to a differentiable manifold at a given point.

Let  $M$  be a differentiable manifold of dimension  $n$  over a topological field  $K$  and  $p \in M$ . The tangent space  $T_p M$  is an  $n$ -dimensional vector space over  $K$  (without a distinguished basis).

INPUT:

- `point` – *ManifoldPoint*; point  $p$  at which the tangent space is defined

EXAMPLES:

Tangent space on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart()
sage: p = M.point((-1,2), name='p')
sage: Tp = M.tangent_space(p) ; Tp
Tangent space at Point p on the 2-dimensional differentiable manifold M
```

Tangent spaces are free modules of finite rank over `SymbolicRing` (actually vector spaces of finite dimension over the manifold base field  $K$ , with  $K = \mathbf{R}$  here):

```
sage: Tp.base_ring()
Symbolic Ring
sage: Tp.category()
Category of finite dimensional vector spaces over Symbolic Ring
sage: Tp.rank()
2
sage: dim(Tp)
2
```

The tangent space is automatically endowed with bases deduced from the vector frames around the point:

```
sage: Tp.bases()
[Basis (d/dx,d/dy) on the Tangent space at Point p on the 2-dimensional
differentiable manifold M]
sage: M.frames()
[Coordinate frame (M, (d/dx,d/dy))]
```

At this stage, only one basis has been defined in the tangent space, but new bases can be added from vector frames on the manifold by means of the method `at()`, for instance, from the frame associated with some new coordinates:

```

sage: c_uv.<u,v> = M.chart()
sage: c_uv.frame().at(p)
Basis (d/du,d/dv) on the Tangent space at Point p on the 2-dimensional
differentiable manifold M
sage: Tp.bases()
[Basis (d/dx,d/dy) on the Tangent space at Point p on the 2-dimensional
differentiable manifold M,
Basis (d/du,d/dv) on the Tangent space at Point p on the 2-dimensional
differentiable manifold M]

```

All the bases defined on  $Tp$  are on the same footing. Accordingly the tangent space is not in the category of modules with a distinguished basis:

```

sage: Tp in ModulesWithBasis(SR)
False

```

It is simply in the category of modules:

```

sage: Tp in Modules(SR)
True

```

Since the base ring is a field, it is actually in the category of vector spaces:

```

sage: Tp in VectorSpaces(SR)
True

```

A typical element:

```

sage: v = Tp.an_element() ; v
Tangent vector at Point p on the
2-dimensional differentiable manifold M
sage: v.display()
d/dx + 2 d/dy
sage: v.parent()
Tangent space at Point p on the
2-dimensional differentiable manifold M

```

The zero vector:

```

sage: Tp.zero()
Tangent vector zero at Point p on the
2-dimensional differentiable manifold M
sage: Tp.zero().display()
zero = 0
sage: Tp.zero().parent()
Tangent space at Point p on the
2-dimensional differentiable manifold M

```

Tangent spaces are unique:

```

sage: M.tangent_space(p) is Tp
True
sage: p1 = M.point((-1,2))
sage: M.tangent_space(p1) is Tp
True

```

even if points are not:

```
sage: p1 is p
False
```

Actually `p1` and `p` share the same tangent space because they compare equal:

```
sage: p1 == p
True
```

The tangent-space uniqueness holds even if the points are created in different coordinate systems:

```
sage: xy_to_uv = c_xy.transition_map(c_uv, (x+y, x-y))
sage: uv_to_xv = xy_to_uv.inverse()
sage: p2 = M.point((1, -3), chart=c_uv, name='p_2')
sage: p2 is p
False
sage: M.tangent_space(p2) is Tp
True
sage: p2 == p
True
```

#### See also:

`FiniteRankFreeModule` for more documentation.

#### Element

alias of `TangentVector`

#### `base_point()`

Return the manifold point at which `self` is defined.

#### EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: p = M.point((1,-2), name='p')
sage: Tp = M.tangent_space(p)
sage: Tp.base_point()
Point p on the 2-dimensional differentiable manifold M
sage: Tp.base_point() is p
True
```

#### `dim()`

Return the vector space dimension of `self`.

#### EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: p = M.point((1,-2), name='p')
sage: Tp = M.tangent_space(p)
sage: Tp.dimension()
2
```

A shortcut is `dim()`:

```
sage: Tp.dim()
2
```

One can also use the global function `dim`:

```
sage: dim(Tp)
2
```

**dimension()**

Return the vector space dimension of `self`.

## EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: p = M.point((1,-2), name='p')
sage: Tp = M.tangent_space(p)
sage: Tp.dimension()
2
```

A shortcut is `dim()`:

```
sage: Tp.dim()
2
```

One can also use the global function `dim`:

```
sage: dim(Tp)
2
```

## 2.6.2 Tangent Vectors

The class *TangentVector* implements tangent vectors to a differentiable manifold.

## AUTHORS:

- Ericourgoulhon, Michal Bejger (2014-2015): initial version
- Travis Scrimshaw (2016): review tweaks

## REFERENCES:

- Chap. 3 of [Lee2013]

**class** `sage.manifolds.differentiable.tangent_vector.TangentVector` (*parent*,  
*name=None, latex\_name=None*)

Bases: `sage.tensor.modules.free_module_tensor.FiniteRankFreeModuleElement`

Tangent vector to a differentiable manifold at a given point.

## INPUT:

- *parent* – *TangentSpace*; the tangent space to which the vector belongs
- *name* – (default: `None`) string; symbol given to the vector
- *latex\_name* – (default: `None`) string; LaTeX symbol to denote the vector; if `None`, *name* will be used

## EXAMPLES:

Tangent vector on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart()
sage: p = M.point((2,3), name='p')
```

```

sage: Tp = M.tangent_space(p)
sage: v = Tp((-2,1), name='v') ; v
Tangent vector v at Point p on the 2-dimensional differentiable
manifold M
sage: v.display()
v = -2 d/dx + d/dy
sage: v.parent()
Tangent space at Point p on the 2-dimensional differentiable manifold M
sage: v in Tp
True

```

**See also:**

`FiniteRankFreeModuleElement` for more documentation.

**plot** (*chart=None, ambient\_coords=None, mapping=None, color='blue', print\_label=True, label=None, label\_color=None, fontsize=10, label\_offset=0.1, parameters=None, scale=1, \*\*extra\_options*)

Plot the vector in a Cartesian graph based on the coordinates of some ambient chart.

The vector is drawn in terms of two (2D graphics) or three (3D graphics) coordinates of a given chart, called hereafter the *ambient chart*. The vector's base point  $p$  (or its image  $\Phi(p)$  by some differentiable mapping  $\Phi$ ) must lie in the ambient chart's domain. If  $\Phi$  is different from the identity mapping, the vector actually depicted is  $d\Phi_p(v)$ , where  $v$  is the current vector (`self`) (see the example of a vector tangent to the 2-sphere below, where  $\Phi : S^2 \rightarrow \mathbf{R}^3$ ).

**INPUT:**

- `chart` – (default: `None`) the ambient chart (see above); if `None`, it is set to the default chart of the open set containing the point at which the vector (or the vector image via the differential  $d\Phi_p$  of mapping) is defined
- `ambient_coords` – (default: `None`) tuple containing the 2 or 3 coordinates of the ambient chart in terms of which the plot is performed; if `None`, all the coordinates of the ambient chart are considered
- `mapping` – (default: `None`) *DiffMap*; differentiable mapping  $\Phi$  providing the link between the point  $p$  at which the vector is defined and the ambient chart `chart`: the domain of `chart` must contain  $\Phi(p)$ ; if `None`, the identity mapping is assumed
- `scale` – (default: 1) value by which the length of the arrow representing the vector is multiplied
- `color` – (default: 'blue') color of the arrow representing the vector
- `print_label` – (boolean; default: `True`) determines whether a label is printed next to the arrow representing the vector
- `label` – (string; default: `None`) label printed next to the arrow representing the vector; if `None`, the vector's symbol is used, if any
- `label_color` – (default: `None`) color to print the label; if `None`, the value of `color` is used
- `fontsize` – (default: 10) size of the font used to print the label
- `label_offset` – (default: 0.1) determines the separation between the vector arrow and the label
- `parameters` – (default: `None`) dictionary giving the numerical values of the parameters that may appear in the coordinate expression of `self` (see example below)
- `**extra_options` – extra options for the arrow plot, like `linestyle`, `width` or `arrowsize` (see `arrow2d()` and `arrow3d()` for details)

**OUTPUT:**

- a graphic object, either an instance of `Graphics` for a 2D plot (i.e. based on 2 coordinates of chart) or an instance of `Graphics3d` for a 3D plot (i.e. based on 3 coordinates of chart)

EXAMPLES:

Vector tangent to a 2-dimensional manifold:

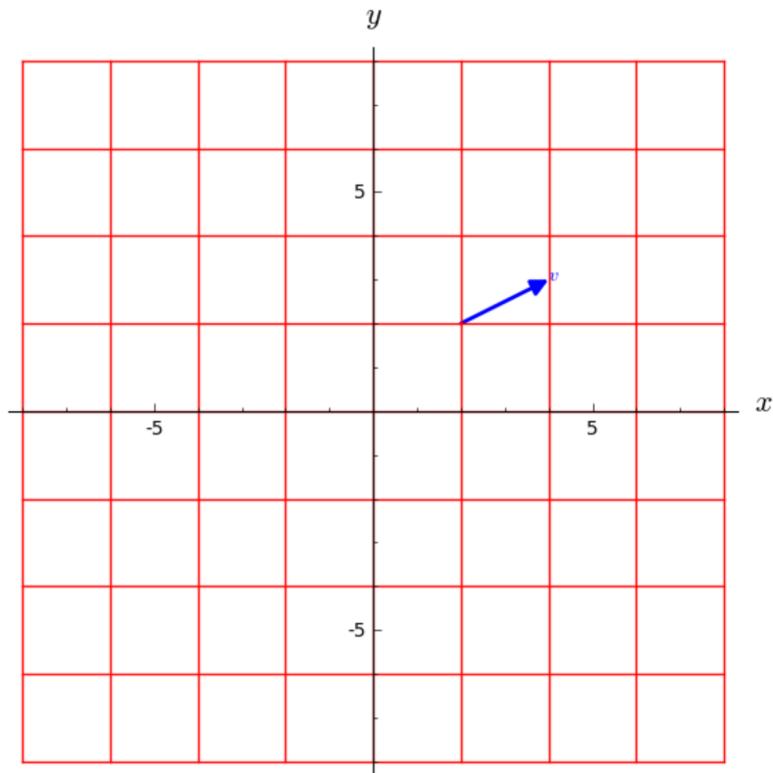
```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: p = M((2,2), name='p')
sage: Tp = M.tangent_space(p)
sage: v = Tp((2, 1), name='v') ; v
Tangent vector v at Point p on the 2-dimensional differentiable
manifold M
```

Plot of the vector alone (arrow + label):

```
sage: v.plot()
Graphics object consisting of 2 graphics primitives
```

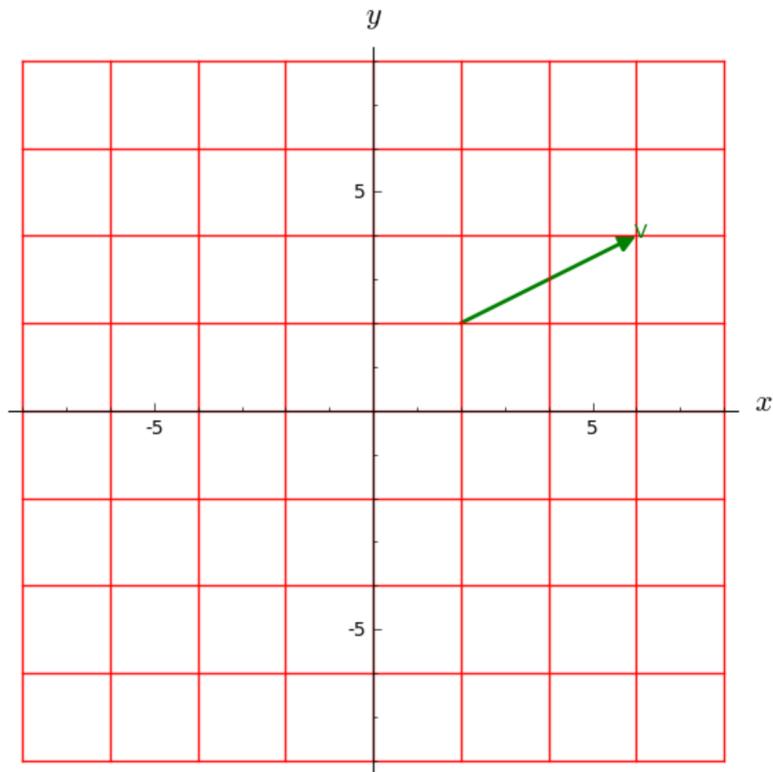
Plot atop of the chart grid:

```
sage: X.plot() + v.plot()
Graphics object consisting of 20 graphics primitives
```

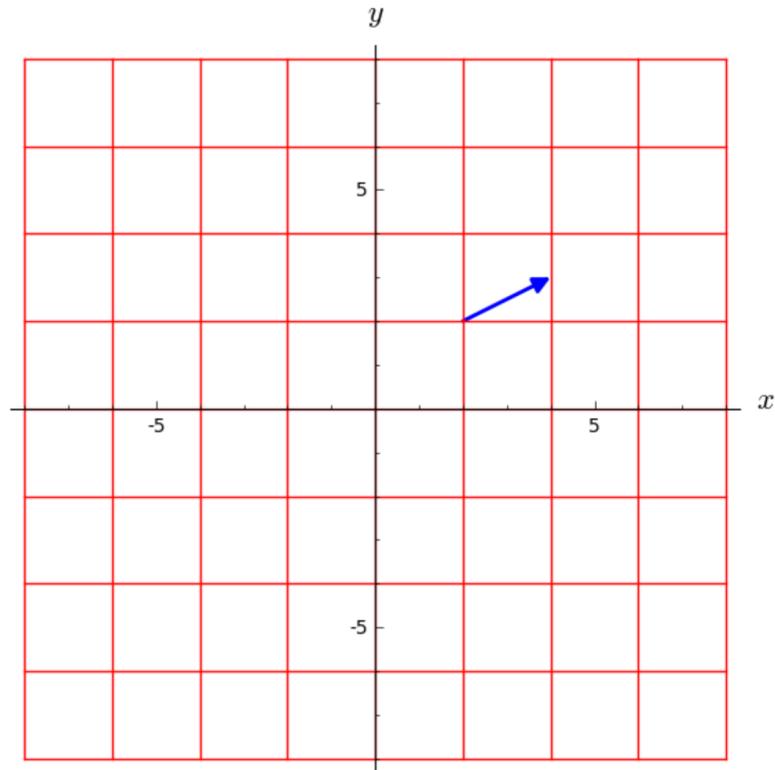


Plots with various options:

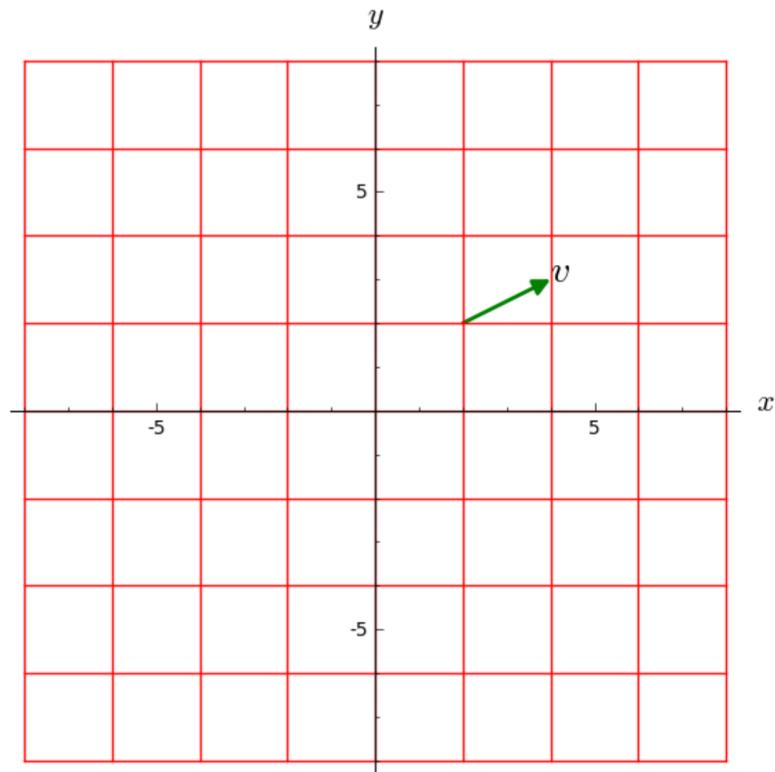
```
sage: X.plot() + v.plot(color='green', scale=2, label='V')
Graphics object consisting of 20 graphics primitives
```



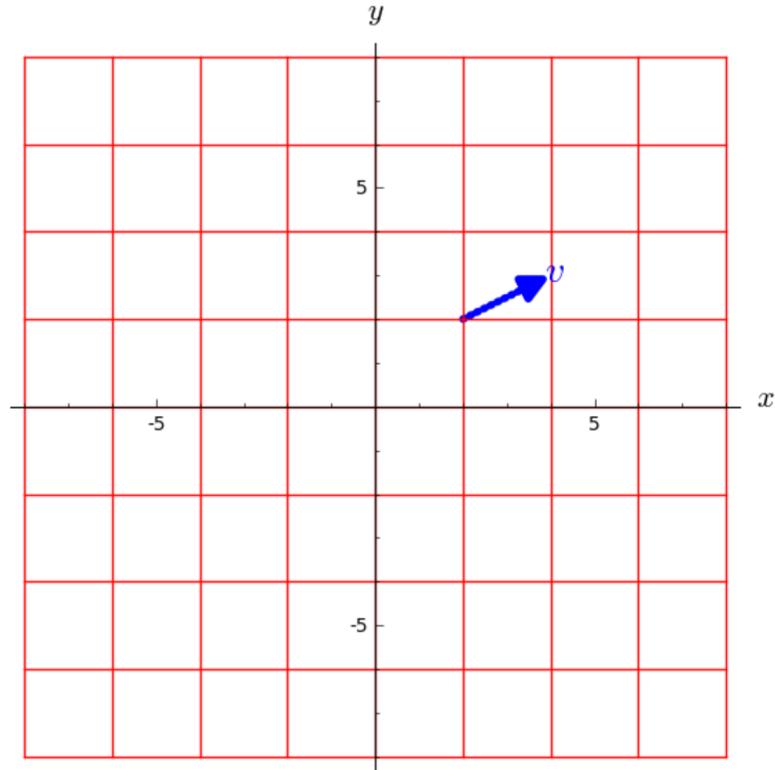
```
sage: X.plot() + v.plot(print_label=False)
Graphics object consisting of 19 graphics primitives
```



```
sage: X.plot() + v.plot(color='green', label_color='black',
....:                  fontsize=20, label_offset=0.2)
Graphics object consisting of 20 graphics primitives
```



```
sage: X.plot() + v.plot(linestyle=':', width=4, arrowsize=8,  
.....:                  fontsize=20)  
Graphics object consisting of 20 graphics primitives
```



Plot with specific values of some free parameters:

```
sage: var('a b')
(a, b)
sage: v = Tp((1+a, -b^2), name='v') ; v.display()
v = (a + 1) d/dx - b^2 d/dy
sage: X.plot() + v.plot(parameters={a: -2, b: 3})
Graphics object consisting of 20 graphics primitives
```

Special case of the zero vector:

```
sage: v = Tp.zero() ; v
Tangent vector zero at Point p on the 2-dimensional differentiable
manifold M
sage: X.plot() + v.plot()
Graphics object consisting of 19 graphics primitives
```

Vector tangent to a 4-dimensional manifold:

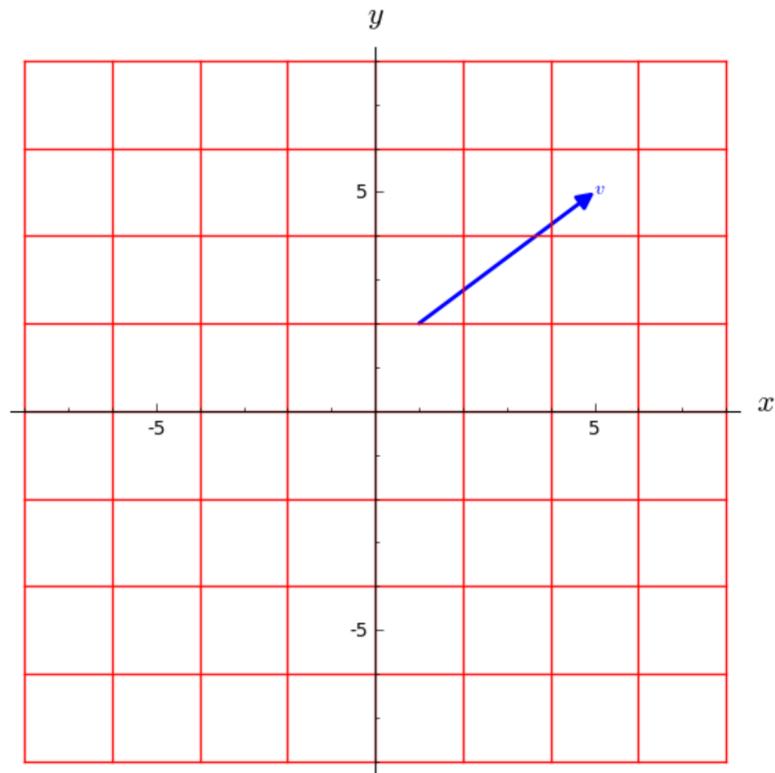
```
sage: M = Manifold(4, 'M')
sage: X.<t,x,y,z> = M.chart()
sage: p = M((0,1,2,3), name='p')
sage: Tp = M.tangent_space(p)
sage: v = Tp((5,4,3,2), name='v') ; v
Tangent vector v at Point p on the 4-dimensional differentiable
manifold M
```

We cannot make a 4D plot directly:

```
sage: v.plot()
Traceback (most recent call last):
...
ValueError: the number of coordinates involved in the plot must
be either 2 or 3, not 4
```

Rather, we have to select some chart coordinates for the plot, via the argument `ambient_coords`. For instance, for a 2-dimensional plot in terms of the coordinates  $(x, y)$ :

```
sage: v.plot(ambient_coords=(x, y))
Graphics object consisting of 2 graphics primitives
```

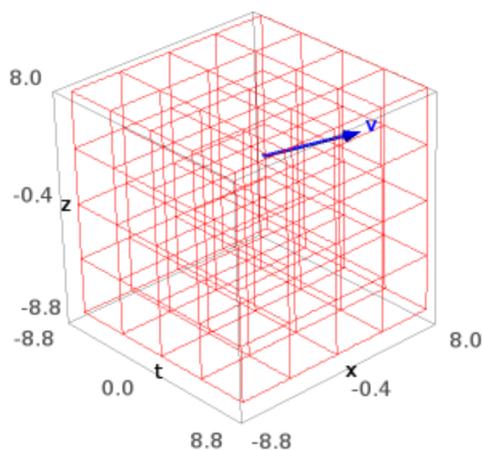


This plot involves only the components  $v^x$  and  $v^y$  of  $v$ . Similarly, for a 3-dimensional plot in terms of the coordinates  $(t, x, y)$ :

```
sage: g = v.plot(ambient_coords=(t, x, z))
sage: print(g)
Graphics3d Object
```

This plot involves only the components  $v^t$ ,  $v^x$  and  $v^z$  of  $v$ . A nice 3D view atop the coordinate grid is obtained via:

```
sage: (X.plot(ambient_coords=(t, x, z)) # long time
....: + v.plot(ambient_coords=(t, x, z),
....:         label_offset=0.5, width=6))
Graphics3d Object
```

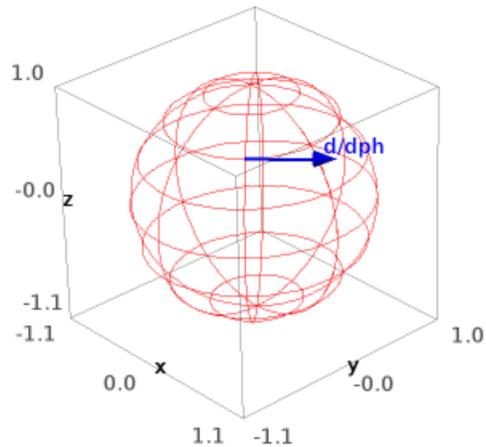


An example of plot via a differential mapping: plot of a vector tangent to a 2-sphere viewed in  $\mathbb{R}^3$ :

```

sage: S2 = Manifold(2, 'S^2')
sage: U = S2.open_subset('U') # the open set covered by spherical coord.
sage: XS.<th,ph> = U.chart(r'th:(0,pi):\theta ph:(0,2*pi):\phi')
sage: R3 = Manifold(3, 'R^3')
sage: X3.<x,y,z> = R3.chart()
sage: F = S2.diff_map(R3, {(XS, X3): [sin(th)*cos(ph),
....:                               sin(th)*sin(ph),
....:                               cos(th)]}, name='F')
sage: F.display() # the standard embedding of S^2 into R^3
F: S^2 --> R^3
on U: (th, ph) |--> (x, y, z) = (cos(ph)*sin(th), sin(ph)*sin(th), cos(th))
sage: p = U.point((pi/4, 7*pi/4), name='p')
sage: v = XS.frame()[1].at(p) ; v # the coordinate vector d/dphi at p
Tangent vector d/dph at Point p on the 2-dimensional differentiable
manifold S^2
sage: graph_v = v.plot(mapping=F)
sage: graph_S2 = XS.plot(chart=X3, mapping=F, number_values=9) # long time
sage: graph_v + graph_S2 # long time
Graphics3d Object

```



## 2.7 Vector Fields

### 2.7.1 Vector Field Modules

The set of vector fields along a differentiable manifold  $U$  with values on a differentiable manifold  $M$  via a differentiable map  $\Phi : U \rightarrow M$  (possibly  $U = M$  and  $\Phi = \text{Id}_M$ ) is a module over the algebra  $C^k(U)$  of differentiable scalar fields on  $U$ . If  $\Phi$  is the identity map, this module is considered a Lie algebroid under the Lie bracket  $[\ , \ ]$  (cf. [Wikipedia article Lie\\_algebroid](#)). It is a free module if and only if  $M$  is parallelizable. Accordingly, there are two classes for vector field modules:

- *VectorFieldModule* for vector fields with values on a generic (in practice, not parallelizable) differentiable manifold  $M$ .
- *VectorFieldFreeModule* for vector fields with values on a parallelizable manifold  $M$ .

AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2014-2015): initial version
- Travis Scrimshaw (2016): structure of Lie algebroid ([trac ticket #20771](#))

REFERENCES:

- [KN1963]
- [Lee2013]
- [ONe1983]

**class** `sage.manifolds.differentiable.vectorfield_module.VectorFieldFreeModule` (*domain*, *dest\_map=None*)

Bases: `sage.tensor.modules.finite_rank_free_module.FiniteRankFreeModule`

Free module of vector fields along a differentiable manifold  $U$  with values on a parallelizable manifold  $M$ , via a differentiable map  $U \rightarrow M$ .

Given a differentiable map

$$\Phi : U \longrightarrow M$$

the *vector field module*  $\mathcal{X}(U, \Phi)$  is the set of all vector fields of the type

$$v : U \longrightarrow TM$$

(where  $TM$  is the tangent bundle of  $M$ ) such that

$$\forall p \in U, v(p) \in T_{\Phi(p)}M,$$

where  $T_{\Phi(p)}M$  is the tangent space to  $M$  at the point  $\Phi(p)$ .

Since  $M$  is parallelizable, the set  $\mathcal{X}(U, \Phi)$  is a free module over  $C^k(U)$ , the ring (algebra) of differentiable scalar fields on  $U$  (see *DiffScalarFieldAlgebra*). In fact, it carries the structure of a finite-dimensional Lie algebroid (cf. [Wikipedia article Lie\\_algebroid](#)).

The standard case of vector fields *on* a differentiable manifold corresponds to  $U = M$  and  $\Phi = \text{Id}_M$ ; we then denote  $\mathcal{X}(M, \text{Id}_M)$  by merely  $\mathcal{X}(M)$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $M$  ( $U$  is then an open interval of  $\mathbf{R}$ ).

---

**Note:** If  $M$  is not parallelizable, the class *VectorFieldModule* should be used instead, for  $\mathcal{X}(U, \Phi)$  is no longer a free module.

---

INPUT:

- `domain` – differentiable manifold  $U$  along which the vector fields are defined
- `dest_map` – (default: `None`) destination map  $\Phi : U \rightarrow M$  (type: *DiffMap*); if `None`, it is assumed that  $U = M$  and  $\Phi$  is the identity map of  $M$  (case of vector fields *on*  $M$ )

EXAMPLES:

Module of vector fields on  $\mathbf{R}^2$ :

```
sage: M = Manifold(2, 'R^2')
sage: cart.<x,y> = M.chart() # Cartesian coordinates on R^2
sage: XM = M.vector_field_module() ; XM
Free module X(R^2) of vector fields on the 2-dimensional differentiable
manifold R^2
sage: XM.category()
Category of finite dimensional modules
over Algebra of differentiable scalar fields
on the 2-dimensional differentiable manifold R^2
sage: XM.base_ring() is M.scalar_field_algebra()
True
```

Since  $\mathbf{R}^2$  is obviously parallelizable, `XM` is a free module:

```
sage: isinstance(XM, FiniteRankFreeModule)
True
```

Some elements:

```
sage: XM.an_element().display()
2 d/dx + 2 d/dy
sage: XM.zero().display()
zero = 0
sage: v = XM([-y,x]) ; v
Vector field on the 2-dimensional differentiable manifold R^2
sage: v.display()
-y d/dx + x d/dy
```

An example of module of vector fields with a destination map  $\Phi$  different from the identity map, namely a mapping  $\Phi : I \rightarrow \mathbf{R}^2$ , where  $I$  is an open interval of  $\mathbf{R}$ :

```
sage: I = Manifold(1, 'I')
sage: canon.<t> = I.chart('t:(0,2*pi)')
sage: Phi = I.diff_map(M, coord_functions=[cos(t), sin(t)], name='Phi',
....:                    latex_name=r'\Phi') ; Phi
Differentiable map Phi from the 1-dimensional differentiable manifold
I to the 2-dimensional differentiable manifold R^2
sage: Phi.display()
Phi: I --> R^2
    t |--> (x, y) = (cos(t), sin(t))
sage: XIM = I.vector_field_module(dest_map=Phi) ; XIM
Free module X(I,Phi) of vector fields along the 1-dimensional
differentiable manifold I mapped into the 2-dimensional differentiable
manifold R^2
sage: XIM.category()
Category of finite dimensional modules
over Algebra of differentiable scalar fields
on the 1-dimensional differentiable manifold I
```

The rank of the free module  $\mathcal{X}(I, \Phi)$  is the dimension of the manifold  $\mathbf{R}^2$ , namely two:

```
sage: XIM.rank()
2
```

A basis of it is induced by the coordinate vector frame of  $\mathbf{R}^2$ :

```
sage: XIM.bases()
[Vector frame (I, (d/dx,d/dy)) with values on the 2-dimensional
differentiable manifold R^2]
```

Some elements of this module:

```
sage: XIM.an_element().display()
2 d/dx + 2 d/dy
sage: v = XIM([t, t^2]) ; v
Vector field along the 1-dimensional differentiable manifold I with
values on the 2-dimensional differentiable manifold R^2
sage: v.display()
t d/dx + t^2 d/dy
```

The test suite is passed:

```
sage: TestSuite(XIM).run()
```

Let us now consider the module of vector fields on the circle  $S^1$ ; we start by constructing the  $S^1$  manifold:

```

sage: M = Manifold(1, 'S^1')
sage: U = M.open_subset('U') # the complement of one point
sage: c_t.<t> = U.chart('t:(0,2*pi)') # the standard angle coordinate
sage: V = M.open_subset('V') # the complement of the point t=pi
sage: M.declare_union(U,V) # S^1 is the union of U and V
sage: c_u.<u> = V.chart('u:(0,2*pi)') # the angle t-pi
sage: t_to_u = c_t.transition_map(c_u, (t-pi,), intersection_name='W',
....:                             restrictions1 = t!=pi, restrictions2 = u!=pi)
sage: u_to_t = t_to_u.inverse()
sage: W = U.intersection(V)

```

$S^1$  cannot be covered by a single chart, so it cannot be covered by a coordinate frame. It is however parallelizable and we introduce a global vector frame as follows. We notice that on their common subdomain,  $W$ , the coordinate vectors  $\partial/\partial t$  and  $\partial/\partial u$  coincide, as we can check explicitly:

```

sage: c_t.frame()[0].display(c_u.frame().restrict(W))
d/dt = d/du

```

Therefore, we can extend  $\partial/\partial t$  to all  $V$  and hence to all  $S^1$ , to form a vector field on  $S^1$  whose components w.r.t. both  $\partial/\partial t$  and  $\partial/\partial u$  are 1:

```

sage: e = M.vector_frame('e')
sage: U.set_change_of_frame(e.restrict(U), c_t.frame(),
....:                       U.tangent_identity_field())
sage: V.set_change_of_frame(e.restrict(V), c_u.frame(),
....:                       V.tangent_identity_field())
sage: e[0].display(c_t.frame())
e_0 = d/dt
sage: e[0].display(c_u.frame())
e_0 = d/du

```

Equipped with the frame  $e$ , the manifold  $S^1$  is manifestly parallelizable:

```

sage: M.is_manifestly_parallelizable()
True

```

Consequently, the module of vector fields on  $S^1$  is a free module:

```

sage: XM = M.vector_field_module() ; XM
Free module X(S^1) of vector fields on the 1-dimensional differentiable
manifold S^1
sage: isinstance(XM, FiniteRankFreeModule)
True
sage: XM.category()
Category of finite dimensional modules
over Algebra of differentiable scalar fields
on the 1-dimensional differentiable manifold S^1
sage: XM.base_ring() is M.scalar_field_algebra()
True

```

The zero element:

```

sage: z = XM.zero() ; z
Vector field zero on the 1-dimensional differentiable manifold S^1
sage: z.display()
zero = 0
sage: z.display(c_t.frame())
zero = 0

```

The module  $\mathcal{X}(S^1)$  coerces to any module of vector fields defined on a subdomain of  $S^1$ , for instance  $\mathcal{X}(U)$ :

```
sage: XU = U.vector_field_module() ; XU
Free module X(U) of vector fields on the Open subset U of the
  1-dimensional differentiable manifold S^1
sage: XU.has_coerce_map_from(XM)
True
sage: XU.coerce_map_from(XM)
Coercion map:
  From: Free module X(S^1) of vector fields on the 1-dimensional differentiable_
↪manifold S^1
  To:   Free module X(U) of vector fields on the Open subset U of the 1-
↪dimensional differentiable manifold S^1
```

The conversion map is actually the restriction of vector fields defined on  $S^1$  to  $U$ .

The Sage test suite for modules is passed:

```
sage: TestSuite(XM).run()
```

### Element

alias of `VectorFieldParal`

### ambient\_domain()

Return the manifold in which the vector fields of `self` take their values.

If the module is  $\mathcal{X}(U, \Phi)$ , returns the codomain  $M$  of  $\Phi$ .

OUTPUT:

- a *DifferentiableManifold* representing the manifold in which the vector fields of `self` take their values

EXAMPLES:

```
sage: M = Manifold(3, 'M')
sage: X.<x,y,z> = M.chart() # makes M parallelizable
sage: XM = M.vector_field_module()
sage: XM.ambient_domain()
3-dimensional differentiable manifold M
sage: U = Manifold(2, 'U')
sage: Y.<u,v> = U.chart()
sage: Phi = U.diff_map(M, {(Y,X): [u+v, u-v, u*v]}, name='Phi')
sage: XU = U.vector_field_module(dest_map=Phi)
sage: XU.ambient_domain()
3-dimensional differentiable manifold M
```

### basis (symbol=None, latex\_symbol=None, from\_frame=None)

Define a basis of `self`.

A basis of the vector field module is actually a vector frame along the differentiable manifold  $U$  over which the vector field module is defined.

If the basis specified by the given symbol already exists, it is simply returned. If no argument is provided the module's default basis is returned.

INPUT:

- `symbol` – (string; default: `None`) a letter (of a few letters) to denote a generic element of the basis; if `None` and `from_frame = None` the module's default basis is returned

- `latex_symbol` – (string; default: None) symbol to denote a generic element of the basis; if None, the value of `symbol` is used
- `from_frame` – (default: None) vector frame  $\tilde{e}$  on the codomain  $M$  of the destination map  $\Phi$  of `self`; the returned basis  $e$  is then such that for all  $p \in U$ , we have  $e(p) = \tilde{e}(\Phi(p))$

OUTPUT:

- a `VectorFrame` representing a basis on `self`

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart() # makes M parallelizable
sage: XM = M.vector_field_module()
sage: e = XM.basis('e'); e
Vector frame (M, (e_0,e_1))
```

See `VectorFrame` for more examples and documentation.

### `destination_map()`

Return the differential map associated to `self`.

The differential map associated to this module is the map

$$\Phi : U \longrightarrow M$$

such that this module is the set  $\mathcal{X}(U, \Phi)$  of all vector fields of the type

$$v : U \longrightarrow TM$$

(where  $TM$  is the tangent bundle of  $M$ ) such that

$$\forall p \in U, v(p) \in T_{\Phi(p)}M,$$

where  $T_{\Phi(p)}M$  is the tangent space to  $M$  at the point  $\Phi(p)$ .

OUTPUT:

- a `DiffMap` representing the differential map  $\Phi$

EXAMPLES:

```
sage: M = Manifold(3, 'M')
sage: X.<x,y,z> = M.chart() # makes M parallelizable
sage: XM = M.vector_field_module()
sage: XM.destination_map()
Identity map Id_M of the 3-dimensional differentiable manifold M
sage: U = Manifold(2, 'U')
sage: Y.<u,v> = U.chart()
sage: Phi = U.diff_map(M, {(Y,X): [u+v, u-v, u*v]}, name='Phi')
sage: XU = U.vector_field_module(dest_map=Phi)
sage: XU.destination_map()
Differentiable map Phi from the 2-dimensional differentiable
manifold U to the 3-dimensional differentiable manifold M
```

### `domain()`

Return the domain of the vector fields in `self`.

If the module is  $\mathcal{X}(U, \Phi)$ , returns the domain  $U$  of  $\Phi$ .

OUTPUT:

- a *DifferentiableManifold* representing the domain of the vector fields that belong to this module

EXAMPLES:

```
sage: M = Manifold(3, 'M')
sage: X.<x,y,z> = M.chart() # makes M parallelizable
sage: XM = M.vector_field_module()
sage: XM.domain()
3-dimensional differentiable manifold M
sage: U = Manifold(2, 'U')
sage: Y.<u,v> = U.chart()
sage: Phi = U.diff_map(M, {(Y,X): [u+v, u-v, u*v]}, name='Phi')
sage: XU = U.vector_field_module(dest_map=Phi)
sage: XU.domain()
2-dimensional differentiable manifold U
```

### **dual\_exterior\_power**(*p*)

Return the  $p$ -th exterior power of the dual of *self*.

If the vector field module is  $\mathcal{X}(U, \Phi)$ , the  $p$ -th exterior power of its dual is the set  $\Lambda^p(U, \Phi)$  of  $p$ -forms along  $U$  with values on  $\Phi(U)$ . It is a module over  $C^k(U)$ , the ring (algebra) of differentiable scalar fields on  $U$ .

INPUT:

- $p$  – non-negative integer

OUTPUT:

- for  $p \geq 1$ , a *DiffFormFreeModule* representing the module  $\Lambda^p(U, \Phi)$ ; for  $p = 0$ , the base ring, i.e.  $C^k(U)$ , is returned instead

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart() # makes M parallelizable
sage: XM = M.vector_field_module()
sage: XM.dual_exterior_power(2)
Free module /\^2(M) of 2-forms on the 2-dimensional differentiable
manifold M
sage: XM.dual_exterior_power(1)
Free module /\^1(M) of 1-forms on the 2-dimensional differentiable
manifold M
sage: XM.dual_exterior_power(1) is XM.dual()
True
sage: XM.dual_exterior_power(0)
Algebra of differentiable scalar fields on the 2-dimensional
differentiable manifold M
sage: XM.dual_exterior_power(0) is M.scalar_field_algebra()
True
```

**See also:**

*DiffFormFreeModule* for more examples and documentation.

### **general\_linear\_group**()

Return the general linear group of *self*.

If the vector field module is  $\mathcal{X}(U, \Phi)$ , the *general linear group* is the group  $\text{GL}(\mathcal{X}(U, \Phi))$  of automorphisms of  $\mathcal{X}(U, \Phi)$ . Note that an automorphism of  $\mathcal{X}(U, \Phi)$  can also be viewed as a *field* along  $U$  of automorphisms of the tangent spaces of  $V = \Phi(U)$ .

OUTPUT:

- a *AutomorphismFieldParalGroup* representing  $GL(\mathcal{X}(U, \Phi))$

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart() # makes M parallelizable
sage: XM = M.vector_field_module()
sage: XM.general_linear_group()
General linear group of the Free module X(M) of vector fields on
the 2-dimensional differentiable manifold M
```

See also:

*AutomorphismFieldParalGroup* for more examples and documentation.

**metric** (*name*, *signature=None*, *latex\_name=None*)

Construct a pseudo-Riemannian metric (nondegenerate symmetric bilinear form) on the current vector field module.

A pseudo-Riemannian metric of the vector field module is actually a field of tangent-space non-degenerate symmetric bilinear forms along the manifold  $U$  on which the vector field module is defined.

INPUT:

- *name* – (string) name given to the metric
- *signature* – (integer; default: None) signature  $S$  of the metric:  $S = n_+ - n_-$ , where  $n_+$  (resp.  $n_-$ ) is the number of positive terms (resp. number of negative terms) in any diagonal writing of the metric components; if *signature* is not provided,  $S$  is set to the manifold's dimension (Riemannian signature)
- *latex\_name* – (string; default: None) LaTeX symbol to denote the metric; if None, it is formed from *name*

OUTPUT:

- instance of *PseudoRiemannianMetricParal* representing the defined pseudo-Riemannian metric.

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart() # makes M parallelizable
sage: XM = M.vector_field_module()
sage: XM.metric('g')
Riemannian metric g on the 2-dimensional differentiable manifold M
sage: XM.metric('g', signature=0)
Lorentzian metric g on the 2-dimensional differentiable manifold M
```

See also:

*PseudoRiemannianMetricParal* for more documentation.

**sym\_bilinear\_form** (*name=None*, *latex\_name=None*)

Construct a symmetric bilinear form on *self*.

A symmetric bilinear form on the vector field module is actually a field of tangent-space symmetric bilinear forms along the differentiable manifold  $U$  over which the vector field module is defined.

INPUT:

- *name* – string (default: None); name given to the automorphism

- `latex_name` – string (default: None); LaTeX symbol to denote the automorphism; if None, the LaTeX symbol is set to `name`

OUTPUT:

- a *TensorFieldParal* of tensor type (0, 2) and symmetric

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart() # makes M parallelizable
sage: XM = M.vector_field_module()
sage: XM.sym_bilinear_form(name='a')
Field of symmetric bilinear forms a on the 2-dimensional
differentiable manifold M
```

See also:

*TensorFieldParal* for more examples and documentation.

**tensor** (*tensor\_type*, *name=None*, *latex\_name=None*, *sym=None*, *antisym=None*, *specific\_type=None*)  
Construct a tensor on `self`.

The tensor is actually a tensor field along the differentiable manifold  $U$  over which `self` is defined.

INPUT:

- `tensor_type` – pair (k,l) with k being the contravariant rank and l the covariant rank
- `name` – (string; default: None) name given to the tensor
- `latex_name` – (string; default: None) LaTeX symbol to denote the tensor; if none is provided, the LaTeX symbol is set to `name`
- `sym` – (default: None) a symmetry or a list of symmetries among the tensor arguments: each symmetry is described by a tuple containing the positions of the involved arguments, with the convention `position=0` for the first argument; for instance:
  - `sym = (0, 1)` for a symmetry between the 1st and 2nd arguments
  - `sym = [(0, 2), (1, 3, 4)]` for a symmetry between the 1st and 3rd arguments and a symmetry between the 2nd, 4th and 5th arguments
- `antisym` – (default: None) antisymmetry or list of antisymmetries among the arguments, with the same convention as for `sym`
- `specific_type` – (default: None) specific subclass of *TensorFieldParal* for the output

OUTPUT:

- a *TensorFieldParal* representing the tensor defined on `self` with the provided characteristics

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart() # makes M parallelizable
sage: XM = M.vector_field_module()
sage: XM.tensor((1,2), name='t')
Tensor field t of type (1,2) on the 2-dimensional differentiable
manifold M
sage: XM.tensor((1,0), name='a')
Vector field a on the 2-dimensional differentiable manifold M
sage: XM.tensor((0,2), name='a', antisym=(0,1))
2-form a on the 2-dimensional differentiable manifold M
```

See *TensorFieldParal* for more examples and documentation.

**tensor\_from\_comp** (*tensor\_type*, *comp*, *name=None*, *latex\_name=None*)

Construct a tensor on *self* from a set of components.

The tensor is actually a tensor field along the differentiable manifold  $U$  over which the vector field module is defined. The tensor symmetries are deduced from those of the components.

INPUT:

- *tensor\_type* – pair  $(k, l)$  with  $k$  being the contravariant rank and  $l$  the covariant rank
- *comp* – Components; the tensor components in a given basis
- *name* – string (default: None); name given to the tensor
- *latex\_name* – string (default: None); LaTeX symbol to denote the tensor; if None, the LaTeX symbol is set to *name*

OUTPUT:

- a *TensorFieldParal* representing the tensor defined on the vector field module with the provided characteristics

EXAMPLES:

A 2-dimensional set of components transformed into a type-(1, 1) tensor field:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: XM = M.vector_field_module()
sage: from sage.tensor.modules.comp import Components
sage: comp = Components(M.scalar_field_algebra(), X.frame(), 2,
....:                   output_formatter=XM._output_formatter)
sage: comp[:] = [[1+x, -y], [x*y, 2-y^2]]
sage: t = XM.tensor_from_comp((1,1), comp, name='t'); t
Tensor field t of type (1,1) on the 2-dimensional differentiable
manifold M
sage: t.display()
t = (x + 1) d/dx*dx - y d/dx*dy + x*y d/dy*dx + (-y^2 + 2) d/dy*dy
```

The same set of components transformed into a type-(0, 2) tensor field:

```
sage: t = XM.tensor_from_comp((0,2), comp, name='t'); t
Tensor field t of type (0,2) on the 2-dimensional differentiable
manifold M
sage: t.display()
t = (x + 1) dx*dx - y dx*dy + x*y dy*dx + (-y^2 + 2) dy*dy
```

**tensor\_module** ( $k, l$ )

Return the free module of all tensors of type  $(k, l)$  defined on *self*.

INPUT:

- $k$  – non-negative integer; the contravariant rank, the tensor type being  $(k, l)$
- $l$  – non-negative integer; the covariant rank, the tensor type being  $(k, l)$

OUTPUT:

- a *TensorFieldFreeModule* representing the free module of type- $(k, l)$  tensors on the vector field module

EXAMPLES:

A tensor field module on a 2-dimensional differentiable manifold:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart() # makes M parallelizable
sage: XM = M.vector_field_module()
sage: XM.tensor_module(1,2)
Free module T^(1,2)(M) of type-(1,2) tensors fields on the
2-dimensional differentiable manifold M
```

The special case of tensor fields of type (1,0):

```
sage: XM.tensor_module(1,0)
Free module X(M) of vector fields on the 2-dimensional
differentiable manifold M
```

The result is cached:

```
sage: XM.tensor_module(1,2) is XM.tensor_module(1,2)
True
sage: XM.tensor_module(1,0) is XM
True
```

See also:

[TensorFieldFreeModule](#) for more examples and documentation.

```
class sage.manifolds.differentiable.vectorfield_module.VectorFieldModule(domain,
                                                                           dest_map=None)
Bases: sage.structure.unique_representation.UniqueRepresentation, sage.
structure.parent.Parent
```

Module of vector fields along a differentiable manifold  $U$  with values on a differentiable manifold  $M$ , via a differentiable map  $U \rightarrow M$ .

Given a differentiable map

$$\Phi : U \longrightarrow M,$$

the *vector field module*  $\mathcal{X}(U, \Phi)$  is the set of all vector fields of the type

$$v : U \longrightarrow TM$$

(where  $TM$  is the tangent bundle of  $M$ ) such that

$$\forall p \in U, v(p) \in T_{\Phi(p)}M,$$

where  $T_{\Phi(p)}M$  is the tangent space to  $M$  at the point  $\Phi(p)$ .

The set  $\mathcal{X}(U, \Phi)$  is a module over  $C^k(U)$ , the ring (algebra) of differentiable scalar fields on  $U$  (see [DiffScalarFieldAlgebra](#)). Furthermore, it is a Lie algebroid under the Lie bracket (cf. [Wikipedia article Lie\\_algebroid](#))

$$[X, Y] = X \circ Y - Y \circ X$$

over the scalarfields if  $\Phi$  is the identity map. That is to say the Lie bracket is antisymmetric, bilinear over the base field, satisfies the Jacobi identity, and  $[X, fY] = X(f)Y + f[X, Y]$ .

The standard case of vector fields *on* a differentiable manifold corresponds to  $U = M$  and  $\Phi = \text{Id}_M$ ; we then denote  $\mathcal{X}(M, \text{Id}_M)$  by merely  $\mathcal{X}(M)$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $M$  ( $U$  is then an open interval of  $\mathbf{R}$ ).

---

**Note:** If  $M$  is parallelizable, the class `VectorFieldFreeModule` should be used instead.

---

INPUT:

- `domain` – differentiable manifold  $U$  along which the vector fields are defined
- `dest_map` – (default: None) destination map  $\Phi : U \rightarrow M$  (type: `DiffMap`); if None, it is assumed that  $U = M$  and  $\Phi$  is the identity map of  $M$  (case of vector fields *on*  $M$ )

EXAMPLES:

Module of vector fields on the 2-sphere:

```
sage: M = Manifold(2, 'M') # the 2-dimensional sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                                     intersection_name='W', restrictions1= x^2+y^2!=0,
....:                                     restrictions2= u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: XM = M.vector_field_module() ; XM
Module X(M) of vector fields on the 2-dimensional differentiable
manifold M
```

$\mathcal{X}(M)$  is a module over the algebra  $C^k(M)$ :

```
sage: XM.category()
Category of modules over Algebra of differentiable scalar fields on the
2-dimensional differentiable manifold M
sage: XM.base_ring() is M.scalar_field_algebra()
True
```

$\mathcal{X}(M)$  is not a free module:

```
sage: isinstance(XM, FiniteRankFreeModule)
False
```

because  $M = S^2$  is not parallelizable:

```
sage: M.is_manifestly_parallelizable()
False
```

On the contrary, the module of vector fields on  $U$  is a free module, since  $U$  is parallelizable (being a coordinate domain):

```
sage: XU = U.vector_field_module()
sage: isinstance(XU, FiniteRankFreeModule)
True
sage: U.is_manifestly_parallelizable()
True
```

The zero element of the module:

```
sage: z = XM.zero() ; z
Vector field zero on the 2-dimensional differentiable manifold M
sage: z.display(c_xy.frame())
zero = 0
sage: z.display(c_uv.frame())
zero = 0
```

The module  $\mathcal{X}(M)$  coerces to any module of vector fields defined on a subdomain of  $M$ , for instance  $\mathcal{X}(U)$ :

```
sage: XU.has_coerce_map_from(XM)
True
sage: XU.coerce_map_from(XM)
Coercion map:
  From: Module X(M) of vector fields on the 2-dimensional differentiable manifold M
  To:   Free module X(U) of vector fields on the Open subset U of the 2-
dimensional differentiable manifold M
```

The conversion map is actually the restriction of vector fields defined on  $M$  to  $U$ .

#### Element

alias of `VectorField`

**alternating\_form** (*degree*, *name=None*, *latex\_name=None*)

Construct an alternating form on the vector field module.

An alternating form on the vector field module is actually a differential form along the differentiable manifold  $U$  over which the vector field module is defined.

INPUT:

- *degree* – the degree of the alternating form (i.e. its tensor rank)
- *name* – (string; optional) name given to the alternating form
- *latex\_name* – (string; optional) LaTeX symbol to denote the alternating form; if none is provided, the LaTeX symbol is set to *name*

OUTPUT:

- instance of *DiffForm*

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: XM = M.vector_field_module()
sage: XM.alternating_form(2, name='a')
2-form a on the 2-dimensional differentiable manifold M
sage: XM.alternating_form(1, name='a')
1-form a on the 2-dimensional differentiable manifold M
```

**See also:**

*DiffForm* for more examples and documentation.

**ambient\_domain** ()

Return the manifold in which the vector fields of this module take their values.

If the module is  $\mathcal{X}(U, \Phi)$ , returns the codomain  $M$  of  $\Phi$ .

OUTPUT:

- instance of *DifferentiableManifold* representing the manifold in which the vector fields of this module take their values

EXAMPLES:

```
sage: M = Manifold(5, 'M')
sage: XM = M.vector_field_module()
sage: XM.ambient_domain()
5-dimensional differentiable manifold M
sage: U = Manifold(2, 'U')
sage: Phi = U.diff_map(M, name='Phi')
sage: XU = U.vector_field_module(dest_map=Phi)
sage: XU.ambient_domain()
5-dimensional differentiable manifold M
```

**automorphism** (*name=None, latex\_name=None*)

Construct an automorphism of the vector field module.

An automorphism of the vector field module is actually a field of tangent-space automorphisms along the differentiable manifold  $U$  over which the vector field module is defined.

INPUT:

- name – (string; optional) name given to the automorphism
- latex\_name – (string; optional) LaTeX symbol to denote the automorphism; if none is provided, the LaTeX symbol is set to name

OUTPUT:

- instance of *AutomorphismField*

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: XM = M.vector_field_module()
sage: XM.automorphism()
Field of tangent-space automorphisms on the 2-dimensional
differentiable manifold M
sage: XM.automorphism(name='a')
Field of tangent-space automorphisms a on the 2-dimensional
differentiable manifold M
```

**See also:**

*AutomorphismField* for more examples and documentation.

**destination\_map** ()

Return the differential map associated to this module.

The differential map associated to this module is the map

$$\Phi : U \longrightarrow M$$

such that this module is the set  $\mathcal{X}(U, \Phi)$  of all vector fields of the type

$$v : U \longrightarrow TM$$

(where  $TM$  is the tangent bundle of  $M$ ) such that

$$\forall p \in U, v(p) \in T_{\Phi(p)}M,$$

where  $T_{\Phi(p)}M$  is the tangent space to  $M$  at the point  $\Phi(p)$ .

OUTPUT:

- instance of *DiffMap* representing the differential map  $\Phi$

EXAMPLES:

```
sage: M = Manifold(5, 'M')
sage: XM = M.vector_field_module()
sage: XM.destination_map()
Identity map Id_M of the 5-dimensional differentiable manifold M
sage: U = Manifold(2, 'U')
sage: Phi = U.diff_map(M, name='Phi')
sage: XU = U.vector_field_module(dest_map=Phi)
sage: XU.destination_map()
Differentiable map Phi from the 2-dimensional differentiable
manifold U to the 5-dimensional differentiable manifold M
```

**domain()**

Return the domain of the vector fields in this module.

If the module is  $\mathcal{X}(U, \Phi)$ , returns the domain  $U$  of  $\Phi$ .

OUTPUT:

- instance of *DifferentiableManifold* representing the domain of the vector fields that belong to this module

EXAMPLES:

```
sage: M = Manifold(5, 'M')
sage: XM = M.vector_field_module()
sage: XM.domain()
5-dimensional differentiable manifold M
sage: U = Manifold(2, 'U')
sage: Phi = U.diff_map(M, name='Phi')
sage: XU = U.vector_field_module(dest_map=Phi)
sage: XU.domain()
2-dimensional differentiable manifold U
```

**dual()**

Return the dual module.

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: XM = M.vector_field_module()
sage: XM.dual()
Module /\^1(M) of 1-forms on the 2-dimensional differentiable
manifold M
```

**dual\_exterior\_power(p)**

Return the  $p$ -th exterior power of the dual of the vector field module.

If the vector field module is  $\mathcal{X}(U, \Phi)$ , the  $p$ -th exterior power of its dual is the set  $\Lambda^p(U, \Phi)$  of  $p$ -forms along  $U$  with values on  $\Phi(U)$ . It is a module over  $C^k(U)$ , the ring (algebra) of differentiable scalar fields on  $U$ .

INPUT:

- $p$  – non-negative integer

OUTPUT:

- for  $p \geq 1$ , instance of `DiffFormModule` representing the module  $\Lambda^p(U, \Phi)$ ; for  $p = 0$ , the base ring, i.e.  $C^k(U)$ , is returned instead

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: XM = M.vector_field_module()
sage: XM.dual_exterior_power(2)
Module /\^2(M) of 2-forms on the 2-dimensional differentiable
manifold M
sage: XM.dual_exterior_power(1)
Module /\^1(M) of 1-forms on the 2-dimensional differentiable
manifold M
sage: XM.dual_exterior_power(1) is XM.dual()
True
sage: XM.dual_exterior_power(0)
Algebra of differentiable scalar fields on the 2-dimensional
differentiable manifold M
sage: XM.dual_exterior_power(0) is M.scalar_field_algebra()
True
```

See also:

`DiffFormModule` for more examples and documentation.

### `general_linear_group()`

Return the general linear group of `self`.

If the vector field module is  $\mathcal{X}(U, \Phi)$ , the *general linear group* is the group  $GL(\mathcal{X}(U, \Phi))$  of automorphisms of  $\mathcal{X}(U, \Phi)$ . Note that an automorphism of  $\mathcal{X}(U, \Phi)$  can also be viewed as a *field* along  $U$  of automorphisms of the tangent spaces of  $M \supset \Phi(U)$ .

OUTPUT:

- instance of class `AutomorphismFieldGroup` representing  $GL(\mathcal{X}(U, \Phi))$

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: XM = M.vector_field_module()
sage: XM.general_linear_group()
General linear group of the Module X(M) of vector fields on the
2-dimensional differentiable manifold M
```

See also:

`AutomorphismFieldGroup` for more examples and documentation.

### `identity_map(name='Id', latex_name=None)`

Construct the identity map on the vector field module.

The identity map on the vector field module is actually a field of tangent-space identity maps along the differentiable manifold  $U$  over which the vector field module is defined.

INPUT:

- `name` – (string; default: 'Id') name given to the identity map
- `latex_name` – (string; optional) LaTeX symbol to denote the identity map; if none is provided, the LaTeX symbol is set to ' $\mathrm{Id}$ ' if `name` is 'Id' and to `name` otherwise

OUTPUT:

- instance of *AutomorphismField*

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: XM = M.vector_field_module()
sage: XM.identity_map()
Field of tangent-space identity maps on the 2-dimensional
differentiable manifold M
```

**linear\_form** (*name=None, latex\_name=None*)

Construct a linear form on the vector field module.

A linear form on the vector field module is actually a field of linear forms (i.e. a 1-form) along the differentiable manifold  $U$  over which the vector field module is defined.

INPUT:

- name – (string; optional) name given to the linear form
- latex\_name – (string; optional) LaTeX symbol to denote the linear form; if none is provided, the LaTeX symbol is set to name

OUTPUT:

- instance of *DiffForm*

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: XM = M.vector_field_module()
sage: XM.linear_form()
1-form on the 2-dimensional differentiable manifold M
sage: XM.linear_form(name='a')
1-form a on the 2-dimensional differentiable manifold M
```

**See also:**

*DiffForm* for more examples and documentation.

**metric** (*name, signature=None, latex\_name=None*)

Construct a pseudo-Riemannian metric (nondegenerate symmetric bilinear form) on the current vector field module.

A pseudo-Riemannian metric of the vector field module is actually a field of tangent-space non-degenerate symmetric bilinear forms along the manifold  $U$  on which the vector field module is defined.

INPUT:

- name – (string) name given to the metric
- signature – (integer; default: None) signature  $S$  of the metric:  $S = n_+ - n_-$ , where  $n_+$  (resp.  $n_-$ ) is the number of positive terms (resp. number of negative terms) in any diagonal writing of the metric components; if *signature* is not provided,  $S$  is set to the manifold's dimension (Riemannian signature)
- latex\_name – (string; default: None) LaTeX symbol to denote the metric; if None, it is formed from name

OUTPUT:

- instance of *PseudoRiemannianMetric* representing the defined pseudo-Riemannian metric.

## EXAMPLES:

```

sage: M = Manifold(2, 'M')
sage: XM = M.vector_field_module()
sage: XM.metric('g')
Riemannian metric g on the 2-dimensional differentiable manifold M
sage: XM.metric('g', signature=0)
Lorentzian metric g on the 2-dimensional differentiable manifold M

```

## See also:

[PseudoRiemannianMetric](#) for more documentation.

**tensor** (*tensor\_type*, *name=None*, *latex\_name=None*, *sym=None*, *antisym=None*, *specific\_type=None*)

Construct a tensor on *self*.

The tensor is actually a tensor field on the domain of the vector field module.

## INPUT:

- *tensor\_type* – pair (k,l) with k being the contravariant rank and l the covariant rank
- *name* – (string; default: None) name given to the tensor
- *latex\_name* – (string; default: None) LaTeX symbol to denote the tensor; if none is provided, the LaTeX symbol is set to *name*
- *sym* – (default: None) a symmetry or a list of symmetries among the tensor arguments: each symmetry is described by a tuple containing the positions of the involved arguments, with the convention *position=0* for the first argument; for instance:
  - *sym* = (0, 1) for a symmetry between the 1st and 2nd arguments
  - *sym* = [(0, 2), (1, 3, 4)] for a symmetry between the 1st and 3rd arguments and a symmetry between the 2nd, 4th and 5th arguments
- *antisym* – (default: None) antisymmetry or list of antisymmetries among the arguments, with the same convention as for *sym*
- *specific\_type* – (default: None) specific subclass of *TensorField* for the output

## OUTPUT:

- instance of *TensorField* representing the tensor defined on the vector field module with the provided characteristics

## EXAMPLES:

```

sage: M = Manifold(2, 'M')
sage: XM = M.vector_field_module()
sage: XM.tensor((1,2), name='t')
Tensor field t of type (1,2) on the 2-dimensional differentiable
manifold M
sage: XM.tensor((1,0), name='a')
Vector field a on the 2-dimensional differentiable manifold M
sage: XM.tensor((0,2), name='a', antisym=(0,1))
2-form a on the 2-dimensional differentiable manifold M

```

## See also:

[TensorField](#) for more examples and documentation.

**tensor\_module** (*k*, *l*)

Return the module of type- $(k, l)$  tensors on *self*.

INPUT:

- $k$  – non-negative integer; the contravariant rank, the tensor type being  $(k, l)$
- $l$  – non-negative integer; the covariant rank, the tensor type being  $(k, l)$

OUTPUT:

- instance of *TensorFieldModule* representing the module  $T^{(k,l)}(U, \Phi)$  of type- $(k, l)$  tensors on the vector field module

EXAMPLES:

A tensor field module on a 2-dimensional differentiable manifold:

```
sage: M = Manifold(2, 'M')
sage: XM = M.vector_field_module()
sage: XM.tensor_module(1,2)
Module T^(1,2)(M) of type-(1,2) tensors fields on the 2-dimensional
differentiable manifold M
```

The special case of tensor fields of type (1,0):

```
sage: XM.tensor_module(1,0)
Module X(M) of vector fields on the 2-dimensional differentiable
manifold M
```

The result is cached:

```
sage: XM.tensor_module(1,2) is XM.tensor_module(1,2)
True
sage: XM.tensor_module(1,0) is XM
True
```

See *TensorFieldModule* for more examples and documentation.

**zero()**

Return the zero of self.

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart() # makes M parallelizable
sage: XM = M.vector_field_module()
sage: XM.zero()
Vector field zero on the 2-dimensional differentiable manifold M
```

## 2.7.2 Vector Fields

Given two differentiable manifolds  $U$  and  $M$  over the same topological field  $K$  and a differentiable map

$$\Phi : U \longrightarrow M,$$

we define a *vector field along  $U$  with values on  $M$*  to be a differentiable map

$$v : U \longrightarrow TM$$

( $TM$  being the tangent bundle of  $M$ ) such that

$$\forall p \in U, v(p) \in T_{\Phi(p)}M.$$

The standard case of vector fields *on* a differentiable manifold corresponds to  $U = M$  and  $\Phi = \text{Id}_M$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $M$  ( $U$  is then an open interval of  $\mathbf{R}$ ).

Vector fields are implemented via two classes: `VectorFieldParal` and `VectorField`, depending respectively whether the manifold  $M$  is parallelizable or not, i.e. whether the bundle  $TM$  is trivial or not.

AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2013-2015) : initial version
- Marco Mancini (2015): parallelization of vector field plots
- Travis Scrimshaw (2016): review tweaks

REFERENCES:

- [KN1963]
- [Lee2013]
- [ONe1983]
- [BG1988]

```
class sage.manifolds.differentiable.vectorfield.VectorField(vector_field_module,
                                                         name=None,          la-
                                                         tex_name=None)
```

Bases: `sage.manifolds.differentiable.tensorfield.TensorField`

Vector field along a differentiable manifold.

An instance of this class is a vector field along a differentiable manifold  $U$  with values on a differentiable manifold  $M$ , via a differentiable map  $U \rightarrow M$ . More precisely, given a differentiable map

$$\Phi : U \longrightarrow M,$$

a *vector field along  $U$  with values on  $M$*  is a differentiable map

$$v : U \longrightarrow TM$$

( $TM$  being the tangent bundle of  $M$ ) such that

$$\forall p \in U, v(p) \in T_{\Phi(p)}M.$$

The standard case of vector fields *on* a differentiable manifold corresponds to  $U = M$  and  $\Phi = \text{Id}_M$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $M$  ( $U$  is then an open interval of  $\mathbf{R}$ ).

---

**Note:** If  $M$  is parallelizable, then `VectorFieldParal` must be used instead.

---

INPUT:

- `vector_field_module` – module  $\mathcal{X}(U, \Phi)$  of vector fields along  $U$  with values on  $M \supset \Phi(U)$
- `name` – (default: `None`) name given to the vector field
- `latex_name` – (default: `None`) LaTeX symbol to denote the vector field; if none is provided, the LaTeX symbol is set to `name`

EXAMPLES:

A vector field on a non-parallelizable 2-dimensional manifold:

```

sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U') ; V = M.open_subset('V')
sage: M.declare_union(U,V)   # M is the union of U and V
sage: c_xy.<x,y> = U.chart() ; c_tu.<t,u> = V.chart()
sage: transf = c_xy.transition_map(c_tu, (x+y, x-y), intersection_name='W',
....:                               restrictions1= x>0, restrictions2= t+u>0)
sage: inv = transf.inverse()
sage: W = U.intersection(V)
sage: eU = c_xy.frame() ; eV = c_tu.frame()
sage: c_tuW = c_tu.restrict(W) ; eVW = c_tuW.frame()
sage: v = M.vector_field('v') ; v
Vector field v on the 2-dimensional differentiable manifold M
sage: v.parent()
Module X(M) of vector fields on the 2-dimensional differentiable
manifold M

```

The vector field is first defined on the domain  $U$  by means of its components with respect to the frame  $eU$ :

```
sage: v[eU,:] = [-y, 1+x]
```

The components with respect to the frame  $eV$  are then deduced by continuation of the components with respect to the frame  $eVW$  on the domain  $W = U \cap V$ , expressed in terms on the coordinates covering  $V$ :

```

sage: v[eV,0] = v[eVW,0,c_tuW].expr()
sage: v[eV,1] = v[eVW,1,c_tuW].expr()

```

At this stage, the vector field is fully defined on the whole manifold:

```

sage: v.display(eU)
v = -y d/dx + (x + 1) d/dy
sage: v.display(eV)
v = (u + 1) d/dt + (-t - 1) d/du

```

The vector field acting on scalar fields:

```

sage: f = M.scalar_field({c_xy: (x+y)^2, c_tu: t^2}, name='f')
sage: s = v(f) ; s
Scalar field v(f) on the 2-dimensional differentiable manifold M
sage: s.display()
v(f): M --> R
on U: (x, y) |--> 2*x^2 - 2*y^2 + 2*x + 2*y
on V: (t, u) |--> 2*t*u + 2*t

```

Some checks:

```

sage: v(f) == f.differential()(v)
True
sage: v(f) == f.lie_der(v)
True

```

The result is defined on the intersection of the vector field's domain and the scalar field's one:

```

sage: s = v(f.restrict(U)) ; s
Scalar field v(f) on the Open subset U of the 2-dimensional
differentiable manifold M
sage: s == v(f).restrict(U)
True
sage: s = v(f.restrict(W)) ; s

```

```

Scalar field v(f) on the Open subset W of the 2-dimensional
differentiable manifold M
sage: s.display()
v(f): W --> R
      (x, y) |--> 2*x^2 - 2*y^2 + 2*x + 2*y
      (t, u) |--> 2*t*u + 2*t
sage: s = v.restrict(U)(f) ; s
Scalar field v(f) on the Open subset U of the 2-dimensional
differentiable manifold M
sage: s.display()
v(f): U --> R
      (x, y) |--> 2*x^2 - 2*y^2 + 2*x + 2*y
on W: (t, u) |--> 2*t*u + 2*t
sage: s = v.restrict(U)(f.restrict(V)) ; s
Scalar field v(f) on the Open subset W of the 2-dimensional
differentiable manifold M
sage: s.display()
v(f): W --> R
      (x, y) |--> 2*x^2 - 2*y^2 + 2*x + 2*y
      (t, u) |--> 2*t*u + 2*t

```

**bracket** (*other*)

Return the Lie bracket [self, other].

## EXAMPLES:

```

sage: M = Manifold(3, 'M', structure='smooth')
sage: X.<x, y, z> = M.chart()
sage: v = -X.frame()[0] + 2*X.frame()[1] - (x^2 - y)*X.frame()[2]
sage: w = (z + y) * X.frame()[1] - X.frame()[2]
sage: vw = v.bracket(w); vw
Vector field on the 3-dimensional differentiable manifold M
sage: vw.display()
(-x^2 + y + 2) d/dy + (-y - z) d/dz

```

**plot** (*chart=None, ambient\_coords=None, mapping=None, chart\_domain=None, fixed\_coords=None, ranges=None, number\_values=None, steps=None, parameters=None, label\_axes=True, color='blue', max\_range=8, scale=1, \*\*extra\_options*)

Plot the vector field in a Cartesian graph based on the coordinates of some ambient chart.

The vector field is drawn in terms of two (2D graphics) or three (3D graphics) coordinates of a given chart, called hereafter the *ambient chart*. The vector field's base points  $p$  (or their images  $\Phi(p)$  by some differentiable mapping  $\Phi$ ) must lie in the ambient chart's domain.

## INPUT:

- **chart** – (default: None) the ambient chart (see above); if None, the default chart of the vector field's domain is used
- **ambient\_coords** – (default: None) tuple containing the 2 or 3 coordinates of the ambient chart in terms of which the plot is performed; if None, all the coordinates of the ambient chart are considered
- **mapping** – *DiffMap* (default: None); differentiable map  $\Phi$  providing the link between the vector field's domain and the ambient chart *chart*; if None, the identity map is assumed
- **chart\_domain** – (default: None) chart on the vector field's domain to define the points at which vector arrows are to be plotted; if None, the default chart of the vector field's domain is used
- **fixed\_coords** – (default: None) dictionary with keys the coordinates of *chart\_domain* that are kept fixed and with values the value of these coordinates; if None, all the coordinates of

`chart_domain` are used

- `ranges` – (default: `None`) dictionary with keys the coordinates of `chart_domain` to be used and values tuples `(x_min, x_max)` specifying the coordinate range for the plot; if `None`, the entire coordinate range declared during the construction of `chart_domain` is considered (with `-Infinity` replaced by `-max_range` and `+Infinity` by `max_range`)
- `number_values` – (default: `None`) either an integer or a dictionary with keys the coordinates of `chart_domain` to be used and values the number of values of the coordinate for sampling the part of the vector field's domain involved in the plot; if `number_values` is a single integer, it represents the number of values for all coordinates; if `number_values` is `None`, it is set to 9 for a 2D plot and to 5 for a 3D plot
- `steps` – (default: `None`) dictionary with keys the coordinates of `chart_domain` to be used and values the step between each constant value of the coordinate; if `None`, the step is computed from the coordinate range (specified in `ranges`) and `number_values`; on the contrary, if the step is provided for some coordinate, the corresponding number of values is deduced from it and the coordinate range
- `parameters` – (default: `None`) dictionary giving the numerical values of the parameters that may appear in the coordinate expression of the vector field (see example below)
- `label_axes` – (default: `True`) boolean determining whether the labels of the coordinate axes of `chart` shall be added to the graph; can be set to `False` if the graph is 3D and must be superposed with another graph
- `color` – (default: `'blue'`) color of the arrows representing the vectors
- `max_range` – (default: 8) numerical value substituted to `+Infinity` if the latter is the upper bound of the range of a coordinate for which the plot is performed over the entire coordinate range (i.e. for which no specific plot range has been set in `ranges`); similarly `-max_range` is the numerical valued substituted for `-Infinity`
- `scale` – (default: 1) value by which the lengths of the arrows representing the vectors is multiplied
- `**extra_options` – extra options for the arrow plot, like `linestyle`, `width` or `arrowsize` (see `arrow2d()` and `arrow3d()` for details)

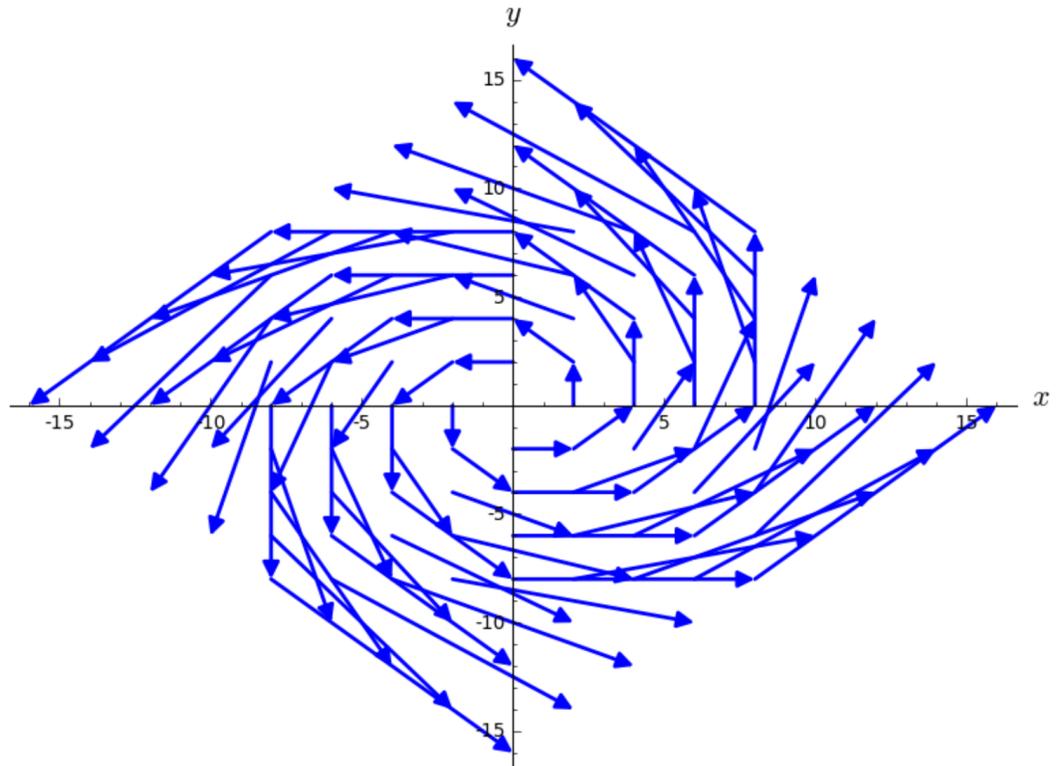
OUTPUT:

- a graphic object, either an instance of `Graphics` for a 2D plot (i.e. based on 2 coordinates of `chart`) or an instance of `Graphics3d` for a 3D plot (i.e. based on 3 coordinates of `chart`)

EXAMPLES:

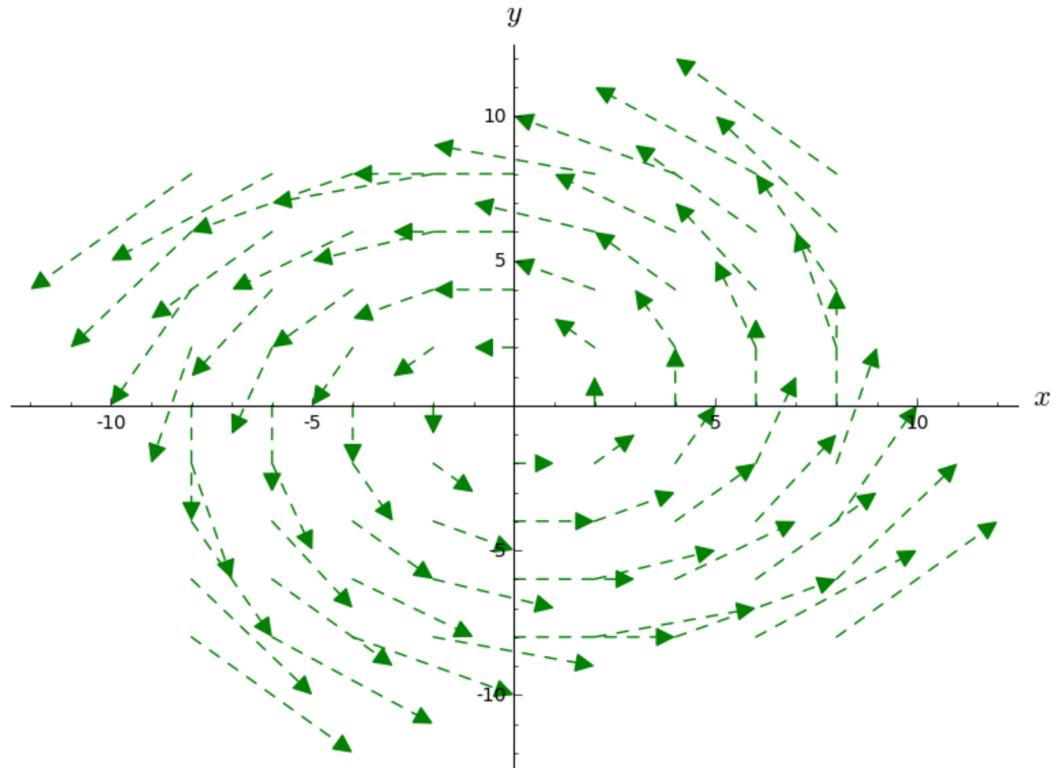
Plot of a vector field on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: v = M.vector_field(name='v')
sage: v[:] = -y, x ; v.display()
v = -y d/dx + x d/dy
sage: v.plot()
Graphics object consisting of 80 graphics primitives
```

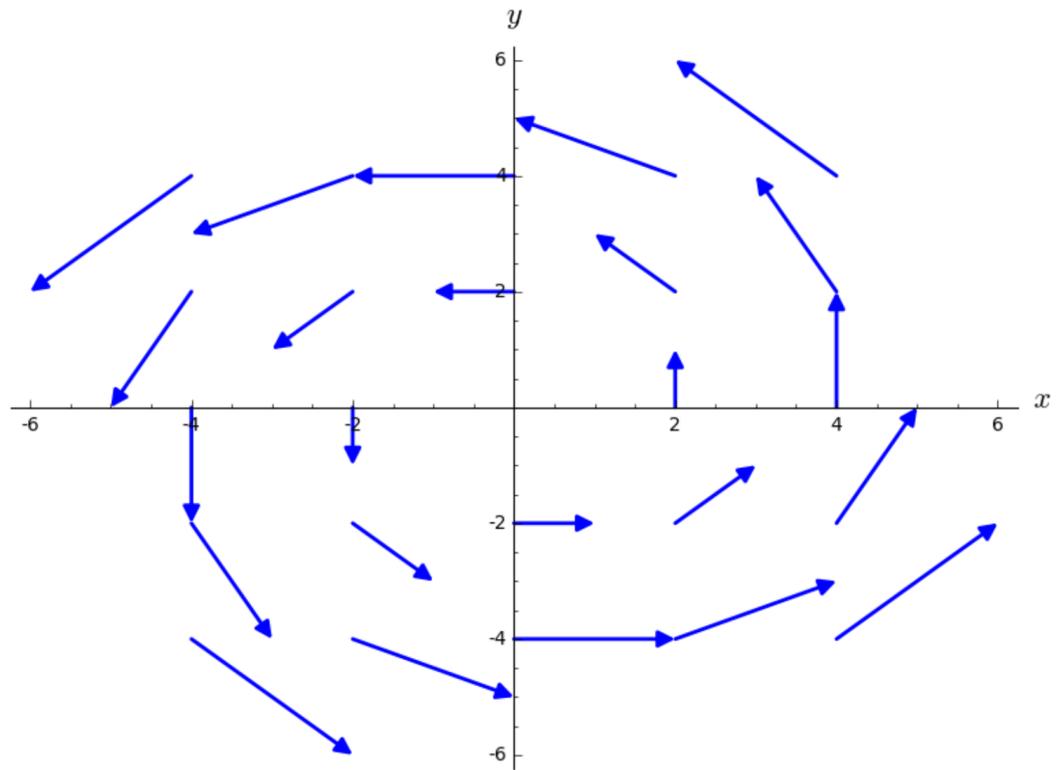


Plot with various options:

```
sage: v.plot(scale=0.5, color='green', linestyle='--', width=1,  
.....:          arrowsize=6)  
Graphics object consisting of 80 graphics primitives
```

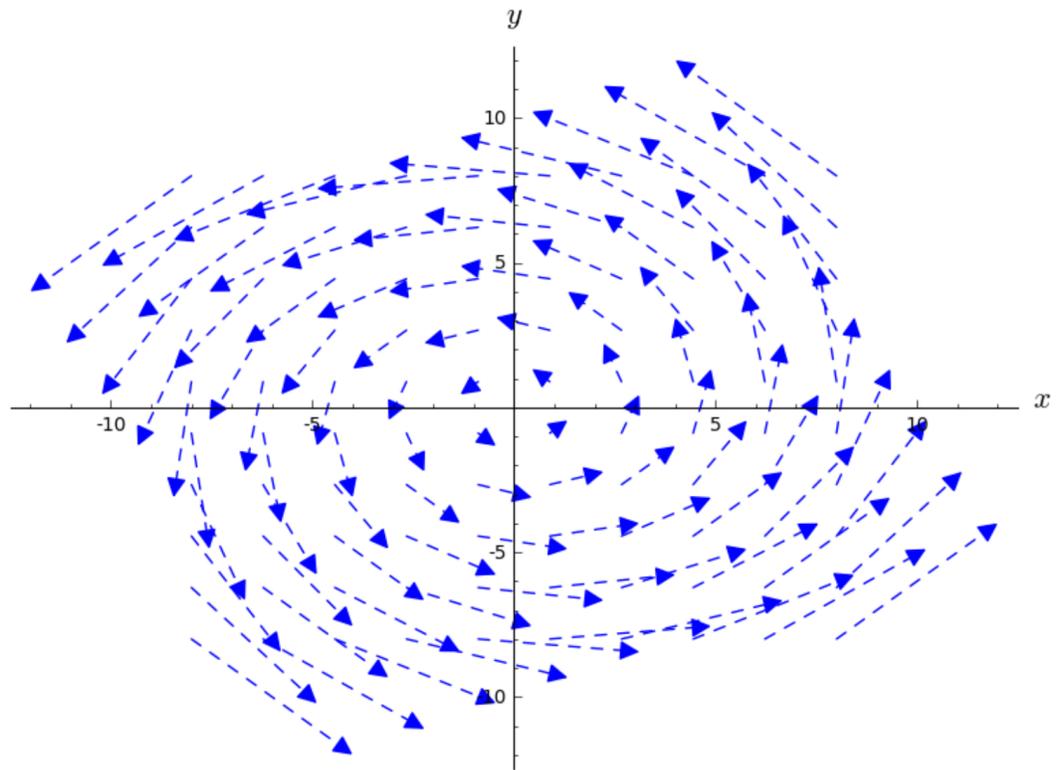


```
sage: v.plot(max_range=4, number_values=5, scale=0.5)  
Graphics object consisting of 24 graphics primitives
```



Plot using parallel computation:

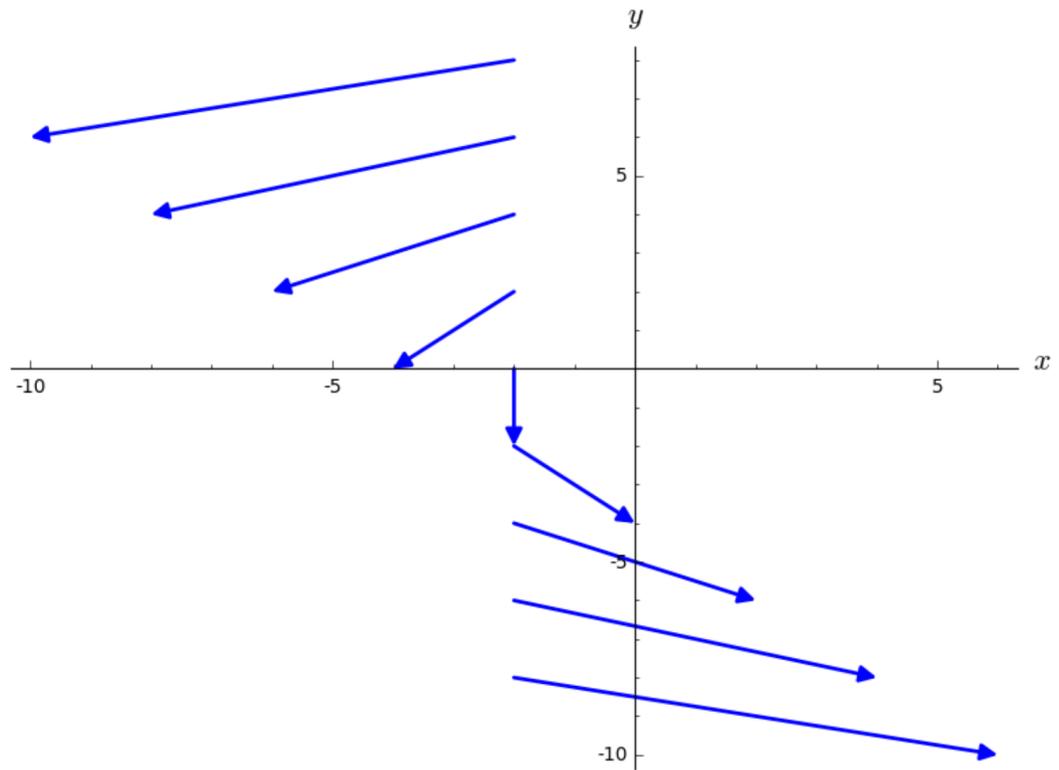
```
sage: Parallelism().set(nproc=2)
sage: v.plot(scale=0.5, number_values=100, linestyle='--', width=1,
....:         arrowsize=6)
Graphics object consisting of 100 graphics primitives
```



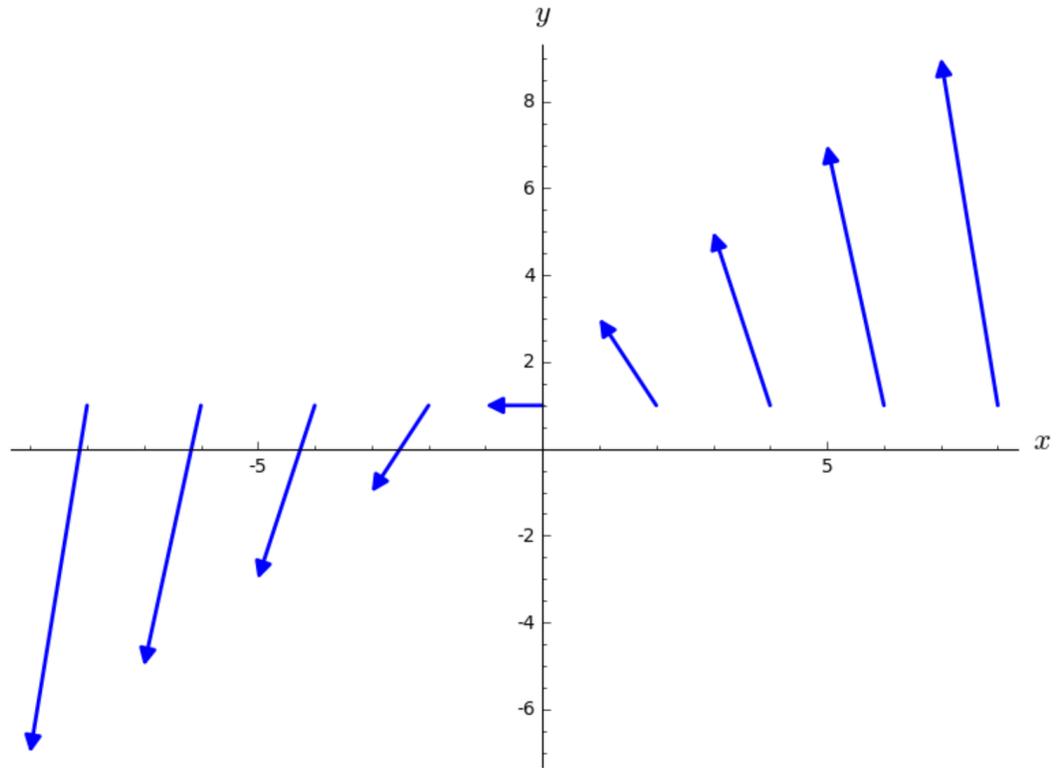
```
sage: Parallelism().set(nproc=1) # switch off parallelization
```

Plots along a line of fixed coordinate:

```
sage: v.plot(fixed_coords={x: -2})  
Graphics object consisting of 9 graphics primitives
```



```
sage: v.plot(fixed_coords={y: 1})  
Graphics object consisting of 9 graphics primitives
```



Let us now consider a vector field on a 4-dimensional manifold:

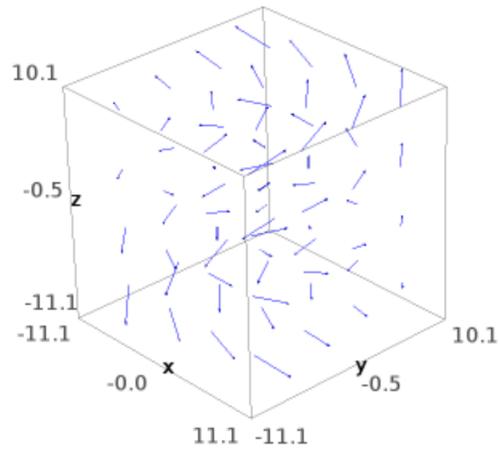
```
sage: M = Manifold(4, 'M')
sage: X.<t,x,y,z> = M.chart()
sage: v = M.vector_field(name='v')
sage: v[:] = (t/8)^2, -t*y/4, t*x/4, t*z/4 ; v.display()
v = 1/64*t^2 d/dt - 1/4*t*y d/dx + 1/4*t*x d/dy + 1/4*t*z d/dz
```

We cannot make a 4D plot directly:

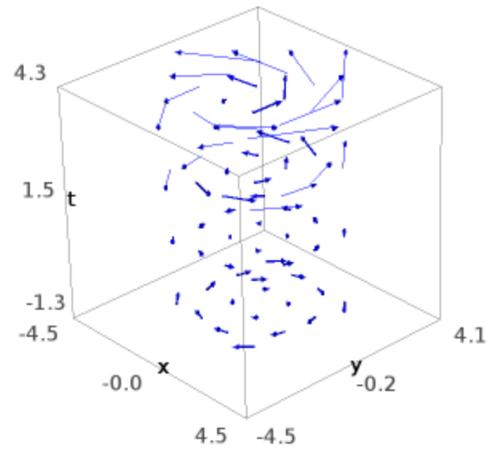
```
sage: v.plot()
Traceback (most recent call last):
...
ValueError: the number of ambient coordinates must be either 2 or 3, not 4
```

Rather, we have to select some coordinates for the plot, via the argument `ambient_coords`. For instance, for a 3D plot:

```
sage: v.plot(ambient_coords=(x, y, z), fixed_coords={t: 1}, # long time
....:         number_values=4)
Graphics3d Object
```

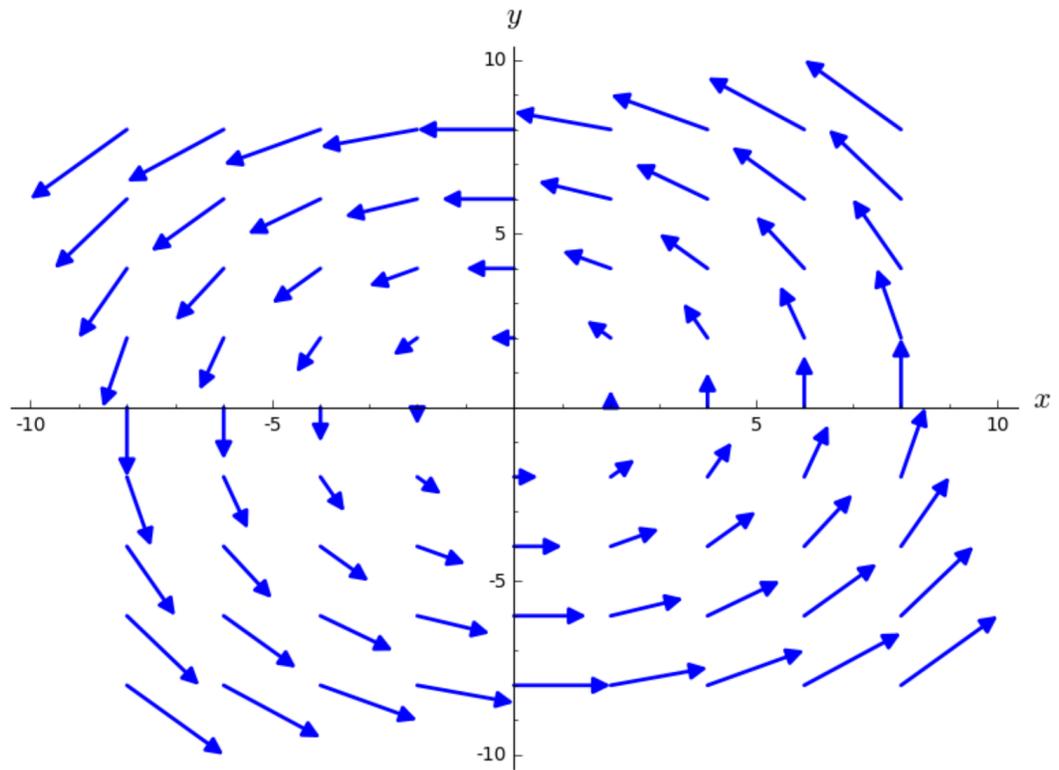


```
sage: v.plot(ambient_coords=(x, y, t), fixed_coords={z: 0}, # long time
....:         ranges={x: (-2,2), y: (-2,2), t: (-1, 4)},
....:         number_values=4)
Graphics3d Object
```

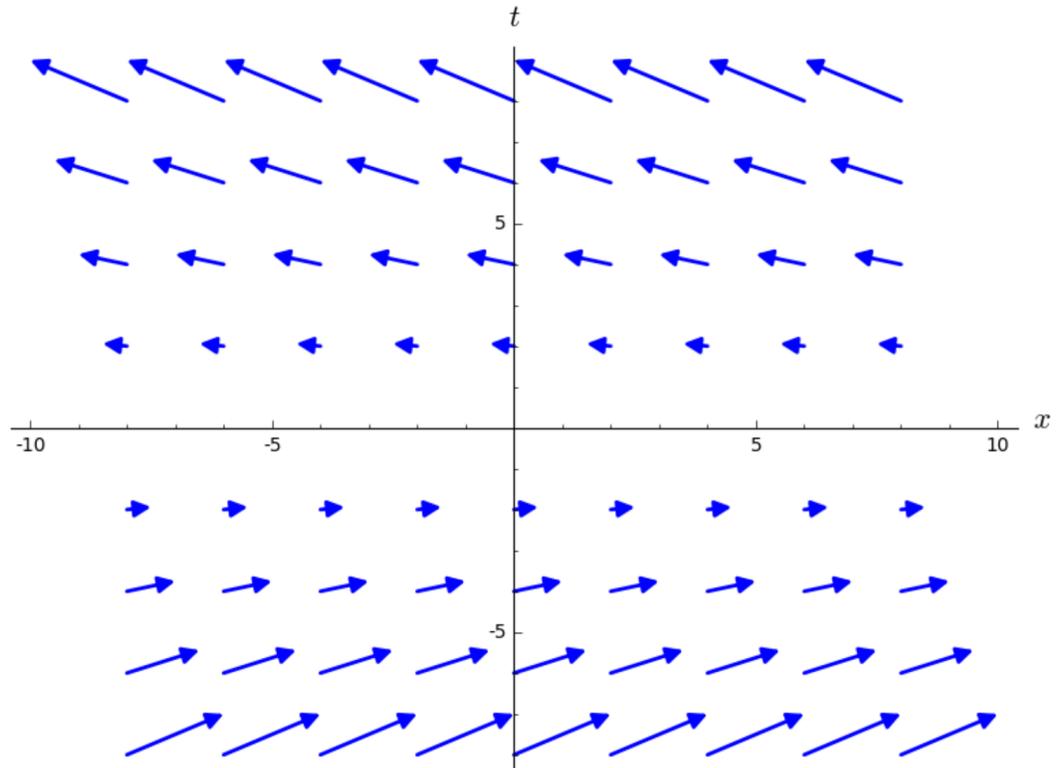


or, for a 2D plot:

```
sage: v.plot(ambient_coords=(x, y), fixed_coords={t: 1, z: 0}) # long time  
Graphics object consisting of 80 graphics primitives
```



```
sage: v.plot(ambient_coords=(x, t), fixed_coords={y: 1, z: 0}) # long time
Graphics object consisting of 72 graphics primitives
```

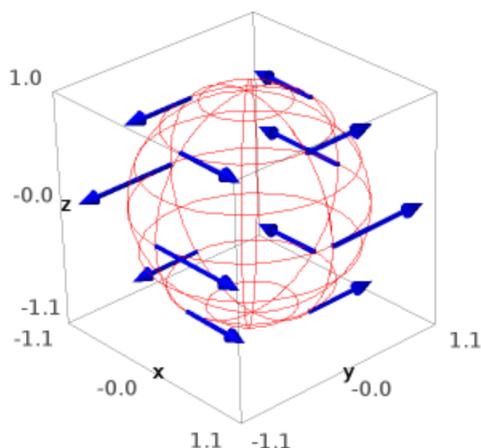


An example of plot via a differential mapping: plot of a vector field tangent to a 2-sphere viewed in  $\mathbf{R}^3$ :

```

sage: S2 = Manifold(2, 'S^2')
sage: U = S2.open_subset('U') # the open set covered by spherical coord.
sage: XS.<th,ph> = U.chart(r'th:(0,pi):\theta ph:(0,2*pi):\phi')
sage: R3 = Manifold(3, 'R^3')
sage: X3.<x,y,z> = R3.chart()
sage: F = S2.diff_map(R3, {(XS, X3): [sin(th)*cos(ph),
....:                               sin(th)*sin(ph), cos(th)]}, name='F')
sage: F.display() # the standard embedding of S^2 into R^3
F: S^2 --> R^3
on U: (th, ph) |--> (x, y, z) = (cos(ph)*sin(th), sin(ph)*sin(th), cos(th))
sage: v = XS.frame()[1] ; v # the coordinate vector d/dphi
Vector field d/dphi on the Open subset U of the 2-dimensional
differentiable manifold S^2
sage: graph_v = v.plot(chart=X3, mapping=F, label_axes=False)
sage: graph_S2 = XS.plot(chart=X3, mapping=F, number_values=9)
sage: graph_v + graph_S2
Graphics3d Object

```



Note that the default values of some arguments of the method `plot` are stored in the dictionary `plot.options`:

```
sage: v.plot.options # random (dictionary output)
{'color': 'blue', 'max_range': 8, 'scale': 1}
```

so that they can be adjusted by the user:

```
sage: v.plot.options['color'] = 'red'
```

From now on, all plots of vector fields will use red as the default color. To restore the original default options, it suffices to type:

```
sage: v.plot.reset()
```

**class** `sage.manifolds.differentiable.vectorfield.VectorFieldParal` (*vector\_field\_module*,  
*name=None, la-  
tex\_name=None*)

**Bases:** `sage.tensor.modules.free_module_tensor.FiniteRankFreeModuleElement`,  
`sage.manifolds.differentiable.tensorfield_paral.TensorFieldParal`, `sage.manifolds.differentiable.vectorfield.VectorField`

Vector field along a differentiable manifold, with values on a parallelizable manifold.

An instance of this class is a vector field along a differentiable manifold  $U$  with values on a parallelizable manifold  $M$ , via a differentiable map  $\Phi : U \rightarrow M$ . More precisely, given a differentiable map

$$\Phi : U \longrightarrow M,$$

a *vector field along  $U$  with values on  $M$*  is a differentiable map

$$v : U \longrightarrow TM$$

( $TM$  being the tangent bundle of  $M$ ) such that

$$\forall p \in U, v(p) \in T_{\Phi(p)}M.$$

The standard case of vector fields *on* a differentiable manifold corresponds to  $U = M$  and  $\Phi = \text{Id}_M$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $M$  ( $U$  is then an open interval of  $\mathbf{R}$ ).

---

**Note:** If  $M$  is not parallelizable, then *VectorField* must be used instead.

---

INPUT:

- `vector_field_module` – free module  $\mathcal{X}(U, \Phi)$  of vector fields along  $U$  with values on  $M \supset \Phi(U)$
- `name` – (default: None) name given to the vector field
- `latex_name` – (default: None) LaTeX symbol to denote the vector field; if none is provided, the LaTeX symbol is set to `name`

EXAMPLES:

A vector field on a parallelizable 3-dimensional manifold:

```
sage: M = Manifold(3, 'M')
sage: c_xyz.<x,y,z> = M.chart()
sage: v = M.vector_field('V') ; v
Vector field V on the 3-dimensional differentiable manifold M
sage: latex(v)
V
```

Vector fields are considered as elements of a module over the ring (algebra) of scalar fields on  $M$ :

```
sage: v.parent()
Free module X(M) of vector fields on the 3-dimensional differentiable
manifold M
sage: v.parent().base_ring()
Algebra of differentiable scalar fields on the 3-dimensional
differentiable manifold M
sage: v.parent() is M.vector_field_module()
True
```

A vector field is a tensor field of rank 1 and of type (1, 0):

```
sage: v.tensor_rank()
1
sage: v.tensor_type()
(1, 0)
```

Components of a vector field with respect to a given frame:

```
sage: e = M.vector_frame('e') ; M.set_default_frame(e)
sage: v[0], v[1], v[2] = (1, 4, 9) # components on M's default frame (e)
sage: v.comp()
1-index components w.r.t. Vector frame (M, (e_0,e_1,e_2))
```

The totality of the components are accessed via the operator `[ : ]`:

```
sage: v[:] = (1, 4, 9) # equivalent to v[0], v[1], v[2] = (1, 4, 9)
sage: v[:]
[1, 4, 9]
```

The components are also read on the expansion on the frame  $e$ , as provided by the method `display()`:

```
sage: v.display() # displays the expansion in the default frame
V = e_0 + 4 e_1 + 9 e_2
```

A subset of the components can be accessed by using slice notation:

```
sage: v[1:] = (-2, -3)
sage: v[:]
[1, -2, -3]
sage: v[:2]
[1, -2]
```

Components in another frame:

```
sage: f = M.vector_frame('f')
sage: for i in range(3):
....:     v.set_comp(f)[i] = (i+1)**3
....:
sage: v.comp(f)[2]
27
sage: v[f, 2] # equivalent to above
27
sage: v.display(f)
V = f_0 + 8 f_1 + 27 f_2
```

The range of the indices depends on the convention set for the manifold:

```
sage: M = Manifold(3, 'M', start_index=1)
sage: c_xyz.<x,y,z> = M.chart()
sage: e = M.vector_frame('e') ; M.set_default_frame(e)
sage: v = M.vector_field('V')
sage: (v[1], v[2], v[3]) = (1, 4, 9)
sage: v[0]
Traceback (most recent call last):
...
IndexError: index out of range: 0 not in [1, 3]
```

A vector field acts on scalar fields (derivation along the vector field):

```
sage: M = Manifold(2, 'M')
sage: c_cart.<x,y> = M.chart()
sage: f = M.scalar_field(x*y^2, name='f')
sage: v = M.vector_field('v')
sage: v[:] = (-y, x)
sage: v.display()
v = -y d/dx + x d/dy
sage: v(f)
Scalar field v(f) on the 2-dimensional differentiable manifold M
sage: v(f).expr()
2*x^2*y - y^3
sage: latex(v(f))
v\left(f\right)
```

Example of a vector field associated with a non-trivial map  $\Phi$ ; a vector field along a curve in  $M$ :

```
sage: R = Manifold(1, 'R')
sage: T.<t> = R.chart() # canonical chart on R
sage: Phi = R.diff_map(M, [cos(t), sin(t)], name='Phi') ; Phi
Differentiable map Phi from the 1-dimensional differentiable manifold R
to the 2-dimensional differentiable manifold M
sage: Phi.display()
Phi: R --> M
    t |--> (x, y) = (cos(t), sin(t))
sage: w = R.vector_field('w', dest_map=Phi) ; w
Vector field w along the 1-dimensional differentiable manifold R with
values on the 2-dimensional differentiable manifold M
sage: w.parent()
Free module X(R,Phi) of vector fields along the 1-dimensional
differentiable manifold R mapped into the 2-dimensional differentiable
manifold M
sage: w[:] = (-sin(t), cos(t))
sage: w.display()
w = -sin(t) d/dx + cos(t) d/dy
```

Value at a given point:

```
sage: p = R((0, ), name='p') ; p
Point p on the 1-dimensional differentiable manifold R
sage: w.at(p)
Tangent vector w at Point Phi(p) on the 2-dimensional differentiable
manifold M
sage: w.at(p).display()
w = d/dy
sage: w.at(p) == v.at(Phi(p))
True
```

### 2.7.3 Vector Frames

The class `VectorFrame` implements vector frames on differentiable manifolds. By *vector frame*, it is meant a field  $e$  on some differentiable manifold  $U$  endowed with a differentiable map  $\Phi : U \rightarrow M$  to a differentiable manifold  $M$  such that for each  $p \in U$ ,  $e(p)$  is a vector basis of the tangent space  $T_{\Phi(p)}M$ .

The standard case of a vector frame *on*  $U$  corresponds to  $U = M$  and  $\Phi = \text{Id}_M$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $M$  ( $U$  is then an open interval of  $\mathbf{R}$ ).

A derived class of `VectorFrame` is `CoordFrame`; it regards the vector frames associated with a chart, i.e. the so-called *coordinate bases*.

The vector frame duals, i.e. the coframes, are implemented via the class `CoFrame`. The derived class `CoordCoFrame` is devoted to coframes deriving from a chart.

AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2013-2015): initial version
- Travis Scrimshaw (2016): review tweaks

REFERENCES:

- [Lee2013]

EXAMPLES:

Setting a vector frame on a 3-dimensional manifold:

```
sage: M = Manifold(3, 'M')
sage: c_xyz.<x,y,z> = M.chart()
sage: e = M.vector_frame('e') ; e
Vector frame (M, (e_0,e_1,e_2))
sage: latex(e)
\left(M, \left(e_0,e_1,e_2\right)\right)
```

The first frame defined on a manifold is its default frame; in the present case it is the coordinate frame defined when introducing the chart `c_xyz`:

```
sage: M.default_frame()
Coordinate frame (M, (d/dx,d/dy,d/dz))
```

The default frame can be changed via the method `set_default_frame()`:

```
sage: M.set_default_frame(e)
sage: M.default_frame()
Vector frame (M, (e_0,e_1,e_2))
```

The elements of a vector frame are vector fields on the manifold:

```
sage: [e[i] for i in M.irange()]
[Vector field e_0 on the 3-dimensional differentiable manifold M,
 Vector field e_1 on the 3-dimensional differentiable manifold M,
 Vector field e_2 on the 3-dimensional differentiable manifold M]
```

Each element can be accessed by its index:

```
sage: e[0]
Vector field e_0 on the 3-dimensional differentiable manifold M
```

The index range depends on the starting index defined on the manifold:

```
sage: M = Manifold(3, 'M', start_index=1)
sage: c_xyz.<x,y,z> = M.chart()
sage: e = M.vector_frame('e')
sage: [e[i] for i in M.irange()]
[Vector field e_1 on the 3-dimensional differentiable manifold M,
 Vector field e_2 on the 3-dimensional differentiable manifold M,
 Vector field e_3 on the 3-dimensional differentiable manifold M]
sage: e[1], e[2], e[3]
(Vector field e_1 on the 3-dimensional differentiable manifold M,
 Vector field e_2 on the 3-dimensional differentiable manifold M,
 Vector field e_3 on the 3-dimensional differentiable manifold M)
```

Let us check that the vector fields `e[i]` are the frame vectors from their components with respect to the frame `e`:

```
sage: e[1].comp(e)[:]
[1, 0, 0]
sage: e[2].comp(e)[:]
[0, 1, 0]
sage: e[3].comp(e)[:]
[0, 0, 1]
```

Defining a vector frame on a manifold automatically creates the dual coframe, which bares the same name (here `e`):

```

sage: M.coframes()
[Coordinate coframe (M, (dx,dy,dz)), Coframe (M, (e^1,e^2,e^3))]
sage: f = M.coframes()[1] ; f
Coframe (M, (e^1,e^2,e^3))

```

Each element of the coframe is a 1-form:

```

sage: f[1], f[2], f[3]
(1-form e^1 on the 3-dimensional differentiable manifold M,
1-form e^2 on the 3-dimensional differentiable manifold M,
1-form e^3 on the 3-dimensional differentiable manifold M)
sage: latex(f[1]), latex(f[2]), latex(f[3])
(e^1, e^2, e^3)

```

Let us check that the coframe ( $e^i$ ) is indeed the dual of the vector frame ( $e_i$ ):

```

sage: f[1](e[1]) # the 1-form e^1 applied to the vector field e_1
Scalar field e^1(e_1) on the 3-dimensional differentiable manifold M
sage: f[1](e[1]).expr() # the explicit expression of e^1(e_1)
1
sage: f[1](e[1]).expr(), f[1](e[2]).expr(), f[1](e[3]).expr()
(1, 0, 0)
sage: f[2](e[1]).expr(), f[2](e[2]).expr(), f[2](e[3]).expr()
(0, 1, 0)
sage: f[3](e[1]).expr(), f[3](e[2]).expr(), f[3](e[3]).expr()
(0, 0, 1)

```

The coordinate frame associated to spherical coordinates of the sphere  $S^2$ :

```

sage: M = Manifold(2, 'S^2', start_index=1) # Part of S^2 covered by spherical coord.
sage: c_spher.<th,ph> = M.chart(r'th:[0,pi]:\theta ph:[0,2*pi):\phi')
sage: b = M.default_frame() ; b
Coordinate frame (S^2, (d/dth,d/dph))
sage: b[1]
Vector field d/dth on the 2-dimensional differentiable manifold S^2
sage: b[2]
Vector field d/dph on the 2-dimensional differentiable manifold S^2

```

The orthonormal frame constructed from the coordinate frame:

```

sage: change_frame = M.automorphism_field()
sage: change_frame[:] = [[1,0], [0, 1/sin(th)]]
sage: e = b.new_frame(change_frame, 'e') ; e
Vector frame (S^2, (e_1,e_2))
sage: e[1][:]
[1, 0]
sage: e[2][:]
[0, 1/sin(th)]

```

The change-of-frame automorphisms and their matrices:

```

sage: M.change_of_frame(c_spher.frame(), e)
Field of tangent-space automorphisms on the 2-dimensional
differentiable manifold S^2
sage: M.change_of_frame(c_spher.frame(), e)[:]
[ 1 0]
[ 0 1/sin(th)]
sage: M.change_of_frame(e, c_spher.frame())

```

```
Field of tangent-space automorphisms on the 2-dimensional
differentiable manifold S^2
```

```
sage: M.change_of_frame(e, c_spher.frame())[: ]
[      1      0]
[      0 sin(th)]
```

**class** sage.manifolds.differentiable.vectorframe.**CoFrame** (*frame*, *symbol*, *latex\_symbol=None*)  
 Bases: sage.tensor.modules.free\_module\_basis.FreeModuleCoBasis

Coframe on a differentiable manifold.

By *coframe*, it is meant a field  $f$  on some differentiable manifold  $U$  endowed with a differentiable map  $\Phi : U \rightarrow M$  to a differentiable manifold  $M$  such that for each  $p \in U$ ,  $f(p)$  is a basis of the vector space  $T_{\Phi(p)}^*M$  (the dual to the tangent space  $T_{\Phi(p)}M$ ).

The standard case of a coframe *on*  $U$  corresponds to  $U = M$  and  $\Phi = \text{Id}_M$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $M$  ( $U$  is then an open interval of  $\mathbf{R}$ ).

INPUT:

- *frame* – the vector frame dual to the coframe
- *symbol* – a letter (of a few letters) to denote a generic 1-form in the coframe
- *latex\_symbol* – (default: None) symbol to denote a generic 1-form in the coframe; if None, the value of *symbol* is used

EXAMPLES:

Coframe on a 3-dimensional manifold:

```
sage: M = Manifold(3, 'M', start_index=1)
sage: c_xyz.<x,y,z> = M.chart()
sage: v = M.vector_frame('v')
sage: from sage.manifolds.differentiable.vectorframe import CoFrame
sage: e = CoFrame(v, 'e') ; e
Coframe (M, (e^1,e^2,e^3))
```

Instead of importing `CoFrame` in the global namespace, the coframe can be obtained by means of the method `dual_basis()`; the symbol is then the same as that of the frame:

```
sage: a = v.dual_basis() ; a
Coframe (M, (v^1,v^2,v^3))
sage: a[1] == e[1]
True
sage: a[1] is e[1]
False
sage: e[1].display(v)
e^1 = v^1
```

The 1-forms composing the coframe are obtained via the operator `[]`:

```
sage: e[1], e[2], e[3]
(1-form e^1 on the 3-dimensional differentiable manifold M,
 1-form e^2 on the 3-dimensional differentiable manifold M,
 1-form e^3 on the 3-dimensional differentiable manifold M)
```

Checking that  $e$  is the dual of  $v$ :

```

sage: e[1](v[1]).expr(), e[1](v[2]).expr(), e[1](v[3]).expr()
(1, 0, 0)
sage: e[2](v[1]).expr(), e[2](v[2]).expr(), e[2](v[3]).expr()
(0, 1, 0)
sage: e[3](v[1]).expr(), e[3](v[2]).expr(), e[3](v[3]).expr()
(0, 0, 1)

```

**at** (*point*)

Return the value of `self` at a given point on the manifold, this value being a basis of the dual of the tangent space at the point.

INPUT:

- `point` – *ManifoldPoint*; point  $p$  in the domain  $U$  of the coframe (denoted  $f$  hereafter)

OUTPUT:

- `FreeModuleCoBasis` representing the basis  $f(p)$  of the vector space  $T_{\Phi(p)}^*M$ , dual to the tangent space  $T_{\Phi(p)}M$ , where  $\Phi : U \rightarrow M$  is the differentiable map associated with  $f$  (possibly  $\Phi = \text{Id}_U$ )

EXAMPLES:

Cobasis of a tangent space on a 2-dimensional manifold:

```

sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: p = M.point((-1,2), name='p')
sage: f = X.coframe() ; f
Coordinate coframe (M, (dx,dy))
sage: fp = f.at(p) ; fp
Dual basis (dx,dy) on the Tangent space at Point p on the
2-dimensional differentiable manifold M
sage: type(fp)
<class 'sage.tensor.modules.free_module_basis.FreeModuleCoBasis'>
sage: fp[0]
Linear form dx on the Tangent space at Point p on the 2-dimensional
differentiable manifold M
sage: fp[1]
Linear form dy on the Tangent space at Point p on the 2-dimensional
differentiable manifold M
sage: fp is X.frame().at(p).dual_basis()
True

```

**class** `sage.manifolds.differentiable.vectorframe.CoordCoFrame` (*coord\_frame*)

Bases: `sage.manifolds.differentiable.vectorframe.CoFrame`

Coordinate coframe on a differentiable manifold.

By *coordinate coframe*, it is meant the  $n$ -tuple of the differentials of the coordinates of some chart on the manifold, with  $n$  being the manifold's dimension.

INPUT:

- `coord_frame` – coordinate frame dual to the coordinate coframe

EXAMPLES:

Coordinate coframe on a 3-dimensional manifold:

```

sage: M = Manifold(3, 'M', start_index=1)
sage: c_xyz.<x,y,z> = M.chart()
sage: M.frames()

```

```
[Coordinate frame (M, (d/dx,d/dy,d/dz))]
sage: M.coframes()
[Coordinate coframe (M, (dx,dy,dz))]
sage: dX = M.coframes()[0] ; dX
Coordinate coframe (M, (dx,dy,dz))
```

The 1-forms composing the coframe are obtained via the operator []:

```
sage: dX[1]
1-form dx on the 3-dimensional differentiable manifold M
sage: dX[2]
1-form dy on the 3-dimensional differentiable manifold M
sage: dX[3]
1-form dz on the 3-dimensional differentiable manifold M
sage: dX[1][:]
[1, 0, 0]
sage: dX[2][:]
[0, 1, 0]
sage: dX[3][:]
[0, 0, 1]
```

The coframe is the dual of the coordinate frame:

```
sage: e = c_xyz.frame() ; e
Coordinate frame (M, (d/dx,d/dy,d/dz))
sage: dX[1](e[1]).expr(), dX[1](e[2]).expr(), dX[1](e[3]).expr()
(1, 0, 0)
sage: dX[2](e[1]).expr(), dX[2](e[2]).expr(), dX[2](e[3]).expr()
(0, 1, 0)
sage: dX[3](e[1]).expr(), dX[3](e[2]).expr(), dX[3](e[3]).expr()
(0, 0, 1)
```

Each 1-form of a coordinate coframe is closed:

```
sage: dX[1].exterior_derivative()
2-form ddx on the 3-dimensional differentiable manifold M
sage: dX[1].exterior_derivative() == 0
True
```

**class** sage.manifolds.differentiable.vectorframe.**CoordFrame**(*chart*)  
 Bases: *sage.manifolds.differentiable.vectorframe.VectorFrame*

Coordinate frame on a differentiable manifold.

By *coordinate frame*, it is meant a vector frame on a differentiable manifold  $M$  that is associated to a coordinate chart on  $M$ .

INPUT:

- *chart* – the chart defining the coordinates

EXAMPLES:

The coordinate frame associated to spherical coordinates of the sphere  $S^2$ :

```
sage: M = Manifold(2, 'S^2', start_index=1) # Part of S^2 covered by spherical_
↪ coord.
sage: M.chart(r'th:[0,pi]:\theta ph:[0,2*pi):\phi')
Chart (S^2, (th, ph))
sage: b = M.default_frame()
```

```

sage: b
Coordinate frame (S^2, (d/dth,d/dph))
sage: b[1]
Vector field d/dth on the 2-dimensional differentiable manifold S^2
sage: b[2]
Vector field d/dph on the 2-dimensional differentiable manifold S^2
sage: latex(b)
\left(S^2, \left(\frac{\partial}{\partial \theta}, \frac{\partial}{\partial \phi}\right)\right)

```

**chart ()**

Return the chart defining this coordinate frame.

**EXAMPLES:**

```

sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: e = X.frame()
sage: e.chart()
Chart (M, (x, y))
sage: U = M.open_subset('U', coord_def={X: x>0})
sage: e.restrict(U).chart()
Chart (U, (x, y))

```

**structure\_coeff ()**

Return the structure coefficients associated to `self`.

$n$  being the manifold's dimension, the structure coefficients of the frame  $(e_i)$  are the  $n^3$  scalar fields  $C_{ij}^k$  defined by

$$[e_i, e_j] = C_{ij}^k e_k.$$

In the present case, since  $(e_i)$  is a coordinate frame,  $C_{ij}^k = 0$ .

**OUTPUT:**

- the structure coefficients  $C_{ij}^k$ , as a vanishing instance of `CompWithSym` with 3 indices ordered as  $(k, i, j)$

**EXAMPLES:**

Structure coefficients of the coordinate frame associated to spherical coordinates in the Euclidean space  $\mathbf{R}^3$ :

```

sage: M = Manifold(3, 'R^3', r'\RR^3', start_index=1) # Part of R^3 covered_
↳by spherical coord.
sage: c_spher = M.chart(r'r:(0,+oo) th:(0,pi):\theta ph:(0,2*pi):\phi')
sage: b = M.default_frame() ; b
Coordinate frame (R^3, (d/dr,d/dth,d/dph))
sage: c = b.structure_coeff() ; c
3-indices components w.r.t. Coordinate frame
(R^3, (d/dr,d/dth,d/dph)), with antisymmetry on the index
positions (1, 2)
sage: c == 0
True

```

```

class sage.manifolds.differentiable.vectorframe.VectorFrame(vector_field_module,
                                                             symbol,                               la-
                                                             tex_symbol=None,
                                                             from_frame=None)

```

Bases: `sage.tensor.modules.free_module_basis.FreeModuleBasis`

Vector frame on a differentiable manifold.

By *vector frame*, it is meant a field  $e$  on some differentiable manifold  $U$  endowed with a differentiable map  $\Phi : U \rightarrow M$  to a differentiable manifold  $M$  such that for each  $p \in U$ ,  $e(p)$  is a vector basis of the tangent space  $T_{\Phi(p)}M$ .

The standard case of a vector frame *on*  $U$  corresponds to  $U = M$  and  $\Phi = \text{Id}_M$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $M$  ( $U$  is then an open interval of  $\mathbf{R}$ ).

For each instantiation of a vector frame, a coframe is automatically created, as an instance of the class `CoFrame`. It is returned by the method `coframe()`.

INPUT:

- `vector_field_module` – free module  $\mathcal{X}(U, \Phi)$  of vector fields along  $U$  with values on  $M \supset \Phi(U)$
- `symbol` – a letter (of a few letters) to denote a generic vector of the frame; can be set to `None` if the parameter `from_frame` is filled
- `latex_symbol` – (default: `None`) symbol to denote a generic vector of the frame; if `None`, the value of `symbol` is used
- `from_frame` – (default: `None`) vector frame  $\tilde{e}$  on the codomain  $M$  of the destination map  $\Phi$ ; the constructed frame  $e$  is then such that  $\forall p \in U, e(p) = \tilde{e}(\Phi(p))$

EXAMPLES:

Setting a vector frame on a 3-dimensional manifold:

```
sage: M = Manifold(3, 'M')
sage: c_xyz.<x,y,z> = M.chart()
sage: e = M.vector_frame('e') ; e
Vector frame (M, (e_0,e_1,e_2))
sage: latex(e)
\left(M, \left(e_0,e_1,e_2\right)\right)
```

The LaTeX symbol can be specified:

```
sage: e = M.vector_frame('E', r"\epsilon")
sage: latex(e)
\left(M, \left(\epsilon_0,\epsilon_1,\epsilon_2\right)\right)
```

Example with a non-trivial map  $\Phi$ ; a vector frame along a curve:

```
sage: U = Manifold(1, 'U') # open interval (-1,1) as a 1-dimensional manifold
sage: T.<t> = U.chart('t: (-1,1)') # canonical chart on U
sage: Phi = U.diff_map(M, [cos(t), sin(t), t], name='Phi',
....:                    latex_name=r'\Phi')
sage: Phi
Differentiable map Phi from the 1-dimensional differentiable manifold U
to the 3-dimensional differentiable manifold M
sage: f = U.vector_frame('f', dest_map=Phi) ; f
Vector frame (U, (f_0,f_1,f_2)) with values on the 3-dimensional
differentiable manifold M
sage: f.domain()
1-dimensional differentiable manifold U
sage: f.ambient_domain()
3-dimensional differentiable manifold M
```

The value of the vector frame at a given point is a basis of the corresponding tangent space:

```

sage: p = U((0,)), name='p') ; p
Point p on the 1-dimensional differentiable manifold U
sage: f.at(p)
Basis (f_0,f_1,f_2) on the Tangent space at Point Phi(p) on the
3-dimensional differentiable manifold M

```

Vector frames are bases of free modules formed by vector fields:

```

sage: e.module()
Free module X(M) of vector fields on the 3-dimensional differentiable
manifold M
sage: e.module().base_ring()
Algebra of differentiable scalar fields on the 3-dimensional
differentiable manifold M
sage: e.module() is M.vector_field_module()
True
sage: e in M.vector_field_module().bases()
True

```

```

sage: f.module()
Free module X(U,Phi) of vector fields along the 1-dimensional
differentiable manifold U mapped into the 3-dimensional differentiable
manifold M
sage: f.module().base_ring()
Algebra of differentiable scalar fields on the 1-dimensional
differentiable manifold U
sage: f.module() is U.vector_field_module(dest_map=Phi)
True
sage: f in U.vector_field_module(dest_map=Phi).bases()
True

```

### **along** (*mapping*)

Return the vector frame deduced from the current frame via a differentiable map, the codomain of which is included in the domain of of the current frame.

If  $e$  is the current vector frame,  $V$  its domain and if  $\Phi : U \rightarrow V$  is a differentiable map from some differentiable manifold  $U$  to  $V$ , the returned object is a vector frame  $\tilde{e}$  along  $U$  with values on  $V$  such that

$$\forall p \in U, \tilde{e}(p) = e(\Phi(p)).$$

INPUT:

- mapping – differentiable map  $\Phi : U \rightarrow V$

OUTPUT:

- vector frame  $\tilde{e}$  along  $U$  defined above.

EXAMPLES:

Vector frame along a curve:

```

sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: R = Manifold(1, 'R') # R as a 1-dimensional manifold
sage: T.<t> = R.chart() # canonical chart on R
sage: Phi = R.diff_map(M, {(T,X): [cos(t), t]}, name='Phi',
....:                    latex_name=r'\Phi') ; Phi
Differentiable map Phi from the 1-dimensional differentiable

```

```

manifold R to the 2-dimensional differentiable manifold M
sage: e = X.frame() ; e
Coordinate frame (M, (d/dx,d/dy))
sage: te = e.along(Phi) ; te
Vector frame (R, (d/dx,d/dy)) with values on the 2-dimensional
differentiable manifold M

```

Check of the formula  $\tilde{e}(p) = e(\Phi(p))$ :

```

sage: p = R((pi,)) ; p
Point on the 1-dimensional differentiable manifold R
sage: te[0].at(p) == e[0].at(Phi(p))
True
sage: te[1].at(p) == e[1].at(Phi(p))
True

```

The result is cached:

```

sage: te is e.along(Phi)
True

```

### **ambient\_domain()**

Return the differentiable manifold in which `self` takes its values.

The ambient domain is the codomain  $M$  of the differentiable map  $\Phi : U \rightarrow M$  associated with the frame.

OUTPUT:

- a *DifferentiableManifold* representing  $M$

EXAMPLES:

```

sage: M = Manifold(2, 'M')
sage: e = M.vector_frame('e')
sage: e.ambient_domain()
2-dimensional differentiable manifold M

```

In the present case, since  $\Phi$  is the identity map:

```

sage: e.ambient_domain() == e.domain()
True

```

An example with a non trivial map  $\Phi$ :

```

sage: U = Manifold(1, 'U')
sage: T.<t> = U.chart()
sage: X.<x,y> = M.chart()
sage: Phi = U.diff_map(M, {(T,X): [cos(t), t]}, name='Phi',
....:                    latex_name=r'\Phi') ; Phi
Differentiable map Phi from the 1-dimensional differentiable
manifold U to the 2-dimensional differentiable manifold M
sage: f = U.vector_frame('f', dest_map=Phi); f
Vector frame (U, (f_0,f_1)) with values on the 2-dimensional
differentiable manifold M
sage: f.ambient_domain()
2-dimensional differentiable manifold M
sage: f.domain()
1-dimensional differentiable manifold U

```

**at** (*point*)

Return the value of `self` at a given point, this value being a basis of the tangent vector space at the point.

INPUT:

- `point` – *ManifoldPoint*; point  $p$  in the domain  $U$  of the vector frame (denoted  $e$  hereafter)

OUTPUT:

- `FreeModuleBasis` representing the basis  $e(p)$  of the tangent vector space  $T_{\Phi(p)}M$ , where  $\Phi : U \rightarrow M$  is the differentiable map associated with  $e$  (possibly  $\Phi = \text{Id}_U$ )

EXAMPLES:

Basis of a tangent space to a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: p = M.point((-1,2), name='p')
sage: e = X.frame() ; e
Coordinate frame (M, (d/dx,d/dy))
sage: ep = e.at(p) ; ep
Basis (d/dx,d/dy) on the Tangent space at Point p on the
2-dimensional differentiable manifold M
sage: type(ep)
<class 'sage.tensor.modules.free_module_basis.FreeModuleBasis'>
sage: ep[0]
Tangent vector d/dx at Point p on the 2-dimensional differentiable
manifold M
sage: ep[1]
Tangent vector d/dy at Point p on the 2-dimensional differentiable
manifold M
```

Note that the symbols used to denote the vectors are same as those for the vector fields of the frame. At this stage, `ep` is the unique basis on the tangent space at  $p$ :

```
sage: Tp = M.tangent_space(p)
sage: Tp.bases()
[Basis (d/dx,d/dy) on the Tangent space at Point p on the
2-dimensional differentiable manifold M]
```

Let us consider a vector frame that is not a coordinate one:

```
sage: aut = M.automorphism_field()
sage: aut[:] = [[1+y^2, 0], [0, 2]]
sage: f = e.new_frame(aut, 'f') ; f
Vector frame (M, (f_0,f_1))
sage: fp = f.at(p) ; fp
Basis (f_0,f_1) on the Tangent space at Point p on the
2-dimensional differentiable manifold M
```

There are now two bases on the tangent space:

```
sage: Tp.bases()
[Basis (d/dx,d/dy) on the Tangent space at Point p on the
2-dimensional differentiable manifold M,
Basis (f_0,f_1) on the Tangent space at Point p on the
2-dimensional differentiable manifold M]
```

Moreover, the changes of bases in the tangent space have been computed from the known relation between the frames  $e$  and  $f$  (field of automorphisms `aut` defined above):

```

sage: Tp.change_of_basis(ep, fp)
Automorphism of the Tangent space at Point p on the 2-dimensional
differentiable manifold M
sage: Tp.change_of_basis(ep, fp).display()
5 d/dx*dx + 2 d/dy*dy
sage: Tp.change_of_basis(fp, ep)
Automorphism of the Tangent space at Point p on the 2-dimensional
differentiable manifold M
sage: Tp.change_of_basis(fp, ep).display()
1/5 d/dx*dx + 1/2 d/dy*dy

```

The dual bases:

```

sage: e.coframe()
Coordinate coframe (M, (dx,dy))
sage: ep.dual_basis()
Dual basis (dx,dy) on the Tangent space at Point p on the
2-dimensional differentiable manifold M
sage: ep.dual_basis() is e.coframe().at(p)
True
sage: f.coframe()
Coframe (M, (f^0,f^1))
sage: fp.dual_basis()
Dual basis (f^0,f^1) on the Tangent space at Point p on the
2-dimensional differentiable manifold M
sage: fp.dual_basis() is f.coframe().at(p)
True

```

### **coframe()**

Return the coframe of self.

EXAMPLES:

```

sage: M = Manifold(2, 'M')
sage: e = M.vector_frame('e')
sage: e.coframe()
Coframe (M, (e^0,e^1))
sage: X.<x,y> = M.chart()
sage: X.frame().coframe()
Coordinate coframe (M, (dx,dy))

```

### **destination\_map()**

Return the differential map associated to this vector frame.

Let  $e$  denote the vector frame; the differential map associated to it is the map  $\Phi : U \rightarrow M$  such that for each  $p \in U$ ,  $e(p)$  is a vector basis of the tangent space  $T_{\Phi(p)}M$ .

OUTPUT:

- a *DiffMap* representing the differential map  $\Phi$

EXAMPLES:

```

sage: M = Manifold(2, 'M')
sage: e = M.vector_frame('e')
sage: e.destination_map()
Identity map Id_M of the 2-dimensional differentiable manifold M

```

An example with a non trivial map  $\Phi$ :

```

sage: U = Manifold(1, 'U')
sage: T.<t> = U.chart()
sage: X.<x,y> = M.chart()
sage: Phi = U.diff_map(M, {(T,X): [cos(t), t]}, name='Phi',
....:                    latex_name=r'\Phi') ; Phi
Differentiable map Phi from the 1-dimensional differentiable
manifold U to the 2-dimensional differentiable manifold M
sage: f = U.vector_frame('f', dest_map=Phi); f
Vector frame (U, (f_0,f_1)) with values on the 2-dimensional
differentiable manifold M
sage: f.destination_map()
Differentiable map Phi from the 1-dimensional differentiable
manifold U to the 2-dimensional differentiable manifold M

```

**domain()**

Return the domain on which `self` is defined.

OUTPUT:

- a *DifferentiableManifold*; representing the domain of the vector frame

EXAMPLES:

```

sage: M = Manifold(2, 'M')
sage: e = M.vector_frame('e')
sage: e.domain()
2-dimensional differentiable manifold M
sage: U = M.open_subset('U')
sage: f = e.restrict(U)
sage: f.domain()
Open subset U of the 2-dimensional differentiable manifold M

```

**new\_frame** (*change\_of\_frame*, *symbol*, *latex\_symbol=None*)

Define a new vector frame from `self`.

The new vector frame is defined from a field of tangent-space automorphisms; its domain is the same as that of the current frame.

INPUT:

- *change\_of\_frame* – *AutomorphismFieldParal*; the field of tangent space automorphisms  $P$  that relates the current frame ( $e_i$ ) to the new frame ( $n_i$ ) according to  $n_i = P(e_i)$
- *symbol* – a letter (of a few letters) to denote a generic vector of the frame
- *latex\_symbol* – (default: `None`) symbol to denote a generic vector of the frame; if `None`, the value of *symbol* is used

OUTPUT:

- the new frame ( $n_i$ ), as an instance of *VectorFrame*

EXAMPLES:

Frame resulting from a  $\pi/3$ -rotation in the Euclidean plane:

```

sage: M = Manifold(2, 'R^2')
sage: c_xy.<x,y> = M.chart()
sage: e = M.vector_frame('e') ; M.set_default_frame(e)
sage: M._frame_changes
{}
sage: rot = M.automorphism_field()

```

```

sage: rot[:] = [[sqrt(3)/2, -1/2], [1/2, sqrt(3)/2]]
sage: n = e.new_frame(rot, 'n')
sage: n[0][:]
[1/2*sqrt(3), 1/2]
sage: n[1][:]
[-1/2, 1/2*sqrt(3)]
sage: a = M.change_of_frame(e,n)
sage: a[:]
[1/2*sqrt(3)      -1/2]
[      1/2 1/2*sqrt(3)]
sage: a == rot
True
sage: a is rot
False
sage: a._components # random (dictionary output)
{Vector frame (R^2, (e_0,e_1)): 2-indices components w.r.t.
 Vector frame (R^2, (e_0,e_1)),
 Vector frame (R^2, (n_0,n_1)): 2-indices components w.r.t.
 Vector frame (R^2, (n_0,n_1))}
sage: a.comp(n)[:]
[1/2*sqrt(3)      -1/2]
[      1/2 1/2*sqrt(3)]
sage: a1 = M.change_of_frame(n,e)
sage: a1[:]
[1/2*sqrt(3)      1/2]
[      -1/2 1/2*sqrt(3)]
sage: a1 == rot.inverse()
True
sage: a1 is rot.inverse()
False
sage: e[0].comp(n)[:]
[1/2*sqrt(3), -1/2]
sage: e[1].comp(n)[:]
[1/2, 1/2*sqrt(3)]

```

**restrict** (*subdomain*)

Return the restriction of `self` to some open subset of its domain.

If the restriction has not been defined yet, it is constructed here.

INPUT:

- `subdomain` – open subset  $V$  of the current frame domain  $U$

OUTPUT:

- the restriction of the current frame to  $V$  as a *VectorFrame*

EXAMPLES:

Restriction of a frame defined on  $\mathbf{R}^2$  to the unit disk:

```

sage: M = Manifold(2, 'R^2', start_index=1)
sage: c_cart.<x,y> = M.chart() # Cartesian coordinates on R^2
sage: a = M.automorphism_field()
sage: a[:] = [[1-y^2,0], [1+x^2, 2]]
sage: e = c_cart.frame().new_frame(a, 'e') ; e
Vector frame (R^2, (e_1,e_2))
sage: U = M.open_subset('U', coord_def={c_cart: x^2+y^2<1})
sage: e_U = e.restrict(U) ; e_U

```

```
Vector frame (U, (e_1,e_2))
```

The vectors of the restriction have the same symbols as those of the original frame:

```
sage: e_U[1].display()
e_1 = (-y^2 + 1) d/dx + (x^2 + 1) d/dy
sage: e_U[2].display()
e_2 = 2 d/dy
```

They are actually the restrictions of the original frame vectors:

```
sage: e_U[1] is e[1].restrict(U)
True
sage: e_U[2] is e[2].restrict(U)
True
```

### `structure_coeff()`

Evaluate the structure coefficients associated to `self`.

$n$  being the manifold's dimension, the structure coefficients of the vector frame  $(e_i)$  are the  $n^3$  scalar fields  $C_{ij}^k$  defined by

$$[e_i, e_j] = C_{ij}^k e_k$$

OUTPUT:

- the structure coefficients  $C_{ij}^k$ , as an instance of `CompWithSym` with 3 indices ordered as  $(k, i, j)$ .

EXAMPLES:

Structure coefficients of the orthonormal frame associated to spherical coordinates in the Euclidean space  $\mathbb{R}^3$ :

```
sage: M = Manifold(3, 'R^3', '\RR^3', start_index=1) # Part of R^3 covered_
↳by spherical coordinates
sage: c_spher.<r,th,ph> = M.chart(r'r:(0,+oo) th:(0,pi):\theta ph:(0,
↳2*pi):\phi')
sage: ch_frame = M.automorphism_field()
sage: ch_frame[1,1], ch_frame[2,2], ch_frame[3,3] = 1, 1/r, 1/(r*sin(th))
sage: M.frames()
[Coordinate frame (R^3, (d/dr,d/dth,d/dph))]
sage: e = c_spher.frame().new_frame(ch_frame, 'e')
sage: e[1][:] # components of e_1 in the manifold's default frame (d/dr, d/
↳dth, d/dth)
[1, 0, 0]
sage: e[2][:]
[0, 1/r, 0]
sage: e[3][:]
[0, 0, 1/(r*sin(th))]
sage: c = e.structure_coeff() ; c
3-indices components w.r.t. Vector frame (R^3, (e_1,e_2,e_3)), with
antisymmetry on the index positions (1, 2)
sage: c[:]:
[[[0, 0, 0], [0, 0, 0], [0, 0, 0]],
 [[0, -1/r, 0], [1/r, 0, 0], [0, 0, 0]],
 [[0, 0, -1/r], [0, 0, -cos(th)/(r*sin(th))], [1/r, cos(th)/(r*sin(th)), 0]]]
sage: c[2,1,2] # C^2_{12}
-1/r
sage: c[3,1,3] # C^3_{13}
```

```
-1/r
sage: c[3,2,3] # C^3_{23}
-cos(th)/(r*sin(th))
```

## 2.7.4 Group of Tangent-Space Automorphism Fields

Given a differentiable manifold  $U$  and a differentiable map  $\Phi : U \rightarrow M$  to a differentiable manifold  $M$  (possibly  $U = M$  and  $\Phi = \text{Id}_M$ ), the *group of tangent-space automorphism fields* associated with  $U$  and  $\Phi$  is the general linear group  $\text{GL}(\mathcal{X}(U, \Phi))$  of the module  $\mathcal{X}(U, \Phi)$  of vector fields along  $U$  with values on  $M \supset \Phi(U)$  (see *VectorFieldModule*). Note that  $\mathcal{X}(U, \Phi)$  is a module over  $C^k(U)$ , the algebra of differentiable scalar fields on  $U$ . Elements of  $\text{GL}(\mathcal{X}(U, \Phi))$  are fields along  $U$  of automorphisms of tangent spaces to  $M$ .

Two classes implement  $\text{GL}(\mathcal{X}(U, \Phi))$  depending whether  $M$  is parallelizable or not: *AutomorphismFieldParalGroup* and *AutomorphismFieldGroup*.

AUTHORS:

- Eric Gourgoulhon (2015): initial version
- Travis Scrimshaw (2016): review tweaks

REFERENCES:

- Chap. 15 of [God1968]

**class** sage.manifolds.differentiable.automorphismfield\_group.**AutomorphismFieldGroup** (*vector\_field\_m*

Bases: sage.structure.unique\_representation.UniqueRepresentation, sage.structure.parent.Parent

General linear group of the module of vector fields along a differentiable manifold  $U$  with values on a differentiable manifold  $M$ .

Given a differentiable manifold  $U$  and a differentiable map  $\Phi : U \rightarrow M$  to a differentiable manifold  $M$  (possibly  $U = M$  and  $\Phi = \text{Id}_M$ ), the *group of tangent-space automorphism fields* associated with  $U$  and  $\Phi$  is the general linear group  $\text{GL}(\mathcal{X}(U, \Phi))$  of the module  $\mathcal{X}(U, \Phi)$  of vector fields along  $U$  with values on  $M \supset \Phi(U)$  (see *VectorFieldModule*). Note that  $\mathcal{X}(U, \Phi)$  is a module over  $C^k(U)$ , the algebra of differentiable scalar fields on  $U$ . Elements of  $\text{GL}(\mathcal{X}(U, \Phi))$  are fields along  $U$  of automorphisms of tangent spaces to  $M$ .

---

**Note:** If  $M$  is parallelizable, then *AutomorphismFieldParalGroup* *must* be used instead.

---

INPUT:

- *vector\_field\_module* – *VectorFieldModule*; module  $\mathcal{X}(U, \Phi)$  of vector fields along  $U$  with values on  $M$

EXAMPLES:

Group of tangent-space automorphism fields of the 2-sphere:

```
sage: M = Manifold(2, 'M') # the 2-dimensional sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                               intersection_name='W',
....:                               restrictions1= x^2+y^2!=0, restrictions2= u^2+v^2!=0)
```

```

sage: uv_to_xy = xy_to_uv.inverse()
sage: G = M.automorphism_field_group() ; G
General linear group of the Module X(M) of vector fields on the
2-dimensional differentiable manifold M

```

G is the general linear group of the vector field module  $\mathcal{X}(M)$ :

```

sage: XM = M.vector_field_module() ; XM
Module X(M) of vector fields on the 2-dimensional differentiable
manifold M
sage: G is XM.general_linear_group()
True

```

G is a non-abelian group:

```

sage: G.category()
Category of groups
sage: G in Groups()
True
sage: G in CommutativeAdditiveGroups()
False

```

The elements of G are tangent-space automorphisms:

```

sage: a = G.an_element(); a
Field of tangent-space automorphisms on the 2-dimensional
differentiable manifold M
sage: a.parent() is G
True
sage: a.restrict(U).display()
2 d/dx*dx + 2 d/dy*dy
sage: a.restrict(V).display()
2 d/du*du + 2 d/dv*dv

```

The identity element of the group G:

```

sage: e = G.one() ; e
Field of tangent-space identity maps on the 2-dimensional
differentiable manifold M
sage: eU = U.default_frame() ; eU
Coordinate frame (U, (d/dx,d/dy))
sage: eV = V.default_frame(); eV
Coordinate frame (V, (d/du,d/dv))
sage: e.display(eU)
Id = d/dx*dx + d/dy*dy
sage: e.display(eV)
Id = d/du*du + d/dv*dv

```

### Element

alias of AutomorphismField

### base\_module()

Return the vector-field module of which `self` is the general linear group.

OUTPUT:

•*VectorFieldModule*

EXAMPLES:

Base module of the group of tangent-space automorphism fields of the 2-sphere:

```
sage: M = Manifold(2, 'M') # the 2-dimensional sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                                     intersection_name='W', restrictions1= x^
↪2+y^2!=0,
....:                                     restrictions2= u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: G = M.automorphism_field_group()
sage: G.base_module()
Module X(M) of vector fields on the 2-dimensional differentiable
manifold M
sage: G.base_module() is M.vector_field_module()
True
```

**one()**

Return identity element of self.

The group identity element is the field of tangent-space identity maps.

OUTPUT:

- *AutomorphismField* representing the identity element

EXAMPLES:

Identity element of the group of tangent-space automorphism fields of the 2-sphere:

```
sage: M = Manifold(2, 'M') # the 2-dimensional sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                                     intersection_name='W', restrictions1= x^
↪2+y^2!=0,
....:                                     restrictions2= u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: G = M.automorphism_field_group()
sage: G.one()
Field of tangent-space identity maps on the 2-dimensional differentiable_
↪manifold M
sage: G.one().restrict(U)[:]
[1 0]
[0 1]
sage: G.one().restrict(V)[:]
[1 0]
[0 1]
```

**class** sage.manifolds.differentiable.automorphismfield\_group.**AutomorphismFieldParalGroup** (*vector*

Bases: sage.tensor.modules.free\_module\_linear\_group.FreeModuleLinearGroup

General linear group of the module of vector fields along a differentiable manifold  $U$  with values on a parallelizable manifold  $M$ .

Given a differentiable manifold  $U$  and a differentiable map  $\Phi : U \rightarrow M$  to a parallelizable manifold  $M$  (possibly  $U = M$  and  $\Phi = \text{Id}_M$ ), the *group of tangent-space automorphism fields* associated with  $U$  and  $\Phi$  is the general linear group  $\text{GL}(\mathcal{X}(U, \Phi))$  of the module  $\mathcal{X}(U, \Phi)$  of vector fields along  $U$  with values on  $M \supset \Phi(U)$  (see [VectorFieldFreeModule](#)). Note that  $\mathcal{X}(U, \Phi)$  is a free module over  $C^k(U)$ , the algebra of differentiable scalar fields on  $U$ . Elements of  $\text{GL}(\mathcal{X}(U, \Phi))$  are fields along  $U$  of automorphisms of tangent spaces to  $M$ .

---

**Note:** If  $M$  is not parallelizable, the class [AutomorphismFieldGroup](#) must be used instead.

---

INPUT:

- `vector_field_module` – [VectorFieldFreeModule](#); free module  $\mathcal{X}(U, \Phi)$  of vector fields along  $U$  with values on  $M$

EXAMPLES:

Group of tangent-space automorphism fields of a 2-dimensional parallelizable manifold:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: XM = M.vector_field_module() ; XM
Free module X(M) of vector fields on the 2-dimensional differentiable
manifold M
sage: G = M.automorphism_field_group(); G
General linear group of the Free module X(M) of vector fields on the
2-dimensional differentiable manifold M
sage: latex(G)
\mathrm{GL}\left( \mathrm{X}\left( M\right) \right)
```

G is nothing but the general linear group of the module  $\mathcal{X}(M)$ :

```
sage: G is XM.general_linear_group()
True
```

G is a group:

```
sage: G.category()
Category of groups
sage: G in Groups()
True
```

It is not an abelian group:

```
sage: G in CommutativeAdditiveGroups()
False
```

The elements of G are tangent-space automorphisms:

```
sage: G.Element
<class 'sage.manifolds.differentiable.automorphismfield.AutomorphismFieldParal'>
sage: a = G.an_element() ; a
Field of tangent-space automorphisms on the 2-dimensional
differentiable manifold M
sage: a.parent() is G
True
```

As automorphisms of  $\mathcal{X}(M)$ , the elements of G map a vector field to a vector field:

```

sage: v = XM.an_element() ; v
Vector field on the 2-dimensional differentiable manifold M
sage: v.display()
2 d/dx + 2 d/dy
sage: a(v)
Vector field on the 2-dimensional differentiable manifold M
sage: a(v).display()
2 d/dx - 2 d/dy

```

Indeed the matrix of  $a$  with respect to the frame  $(\partial_x, \partial_y)$  is:

```

sage: a[X.frame(),:]
[ 1  0]
[ 0 -1]

```

The elements of  $G$  can also be considered as tensor fields of type  $(1, 1)$ :

```

sage: a.tensor_type()
(1, 1)
sage: a.tensor_rank()
2
sage: a.domain()
2-dimensional differentiable manifold M
sage: a.display()
d/dx*dx - d/dy*dy

```

The identity element of the group  $G$  is:

```

sage: id = G.one() ; id
Field of tangent-space identity maps on the 2-dimensional
differentiable manifold M
sage: id*a == a
True
sage: a*id == a
True
sage: a*a^(-1) == id
True
sage: a^(-1)*a == id
True

```

Construction of an element by providing its components with respect to the manifold's default frame (frame associated to the coordinates  $(x, y)$ ):

```

sage: b = G([[1+x^2, 0], [0, 1+y^2]]) ; b
Field of tangent-space automorphisms on the 2-dimensional
differentiable manifold M
sage: b.display()
(x^2 + 1) d/dx*dx + (y^2 + 1) d/dy*dy
sage: (~b).display() # the inverse automorphism
1/(x^2 + 1) d/dx*dx + 1/(y^2 + 1) d/dy*dy

```

We check the group law on these elements:

```

sage: (a*b)^(-1) == b^(-1) * a^(-1)
True

```

Invertible tensor fields of type  $(1, 1)$  can be converted to elements of  $G$ :

```

sage: t = M.tensor_field(1, 1, name='t')
sage: t[:] = [[1+exp(y), x*y], [0, 1+x^2]]
sage: t1 = G(t) ; t1
Field of tangent-space automorphisms t on the 2-dimensional
differentiable manifold M
sage: t1 in G
True
sage: t1.display()
t = (e^y + 1) d/dx*dx + x*y d/dx*dy + (x^2 + 1) d/dy*dy
sage: t1^(-1)
Field of tangent-space automorphisms t^(-1) on the 2-dimensional
differentiable manifold M
sage: (t1^(-1)).display()
t^(-1) = 1/(e^y + 1) d/dx*dx - x*y/(x^2 + (x^2 + 1)*e^y + 1) d/dx*dy
+ 1/(x^2 + 1) d/dy*dy

```

Since any automorphism field can be considered as a tensor field of type-(1,1) on M, there is a coercion map from G to the module  $T^{(1,1)}(M)$  of type-(1,1) tensor fields:

```

sage: T11 = M.tensor_field_module((1,1)) ; T11
Free module T^(1,1)(M) of type-(1,1) tensors fields on the
2-dimensional differentiable manifold M
sage: T11.has_coerce_map_from(G)
True

```

An explicit call of this coercion map is:

```

sage: tt = T11(t1) ; tt
Tensor field t of type (1,1) on the 2-dimensional differentiable
manifold M
sage: tt == t
True

```

An implicit call of the coercion map is performed to subtract an element of G from an element of  $T^{(1,1)}(M)$ :

```

sage: s = t - t1 ; s
Tensor field t-t of type (1,1) on
the 2-dimensional differentiable manifold M
sage: s.parent() is T11
True
sage: s.display()
t-t = 0

```

as well as for the reverse operation:

```

sage: s = t1 - t ; s
Tensor field t-t of type (1,1) on the 2-dimensional differentiable
manifold M
sage: s.display()
t-t = 0

```

### Element

alias of AutomorphismFieldParal

## 2.7.5 Tangent-Space Automorphism Fields

The class `AutomorphismField` implements fields of automorphisms of tangent spaces to a generic (a priori not parallelizable) differentiable manifold, while the class `AutomorphismFieldParal` is devoted to fields of automorphisms of tangent spaces to a parallelizable manifold. The latter play the important role of transitions between vector frames sharing the same domain on a differentiable manifold.

AUTHORS:

- Eric Gourgoulhon (2015): initial version
- Travis Scrimshaw (2016): review tweaks

**class** `sage.manifolds.differentiable.automorphismfield.AutomorphismField` (*vector\_field\_module*, *name=None*, *la-* *tex\_name=None*, *is\_identity=False*)

Bases: `sage.manifolds.differentiable.tensorfield.TensorField`

Field of automorphisms of tangent spaces to a generic (a priori not parallelizable) differentiable manifold.

Given a differentiable manifold  $U$  and a differentiable map  $\Phi : U \rightarrow M$  to a differentiable manifold  $M$ , a *field of tangent-space automorphisms along  $U$  with values on  $M \supset \Phi(U)$*  is a differentiable map

$$a : U \longrightarrow T^{(1,1)}M,$$

with  $T^{(1,1)}M$  being the tensor bundle of type  $(1, 1)$  over  $M$ , such that

$$\forall p \in U, a(p) \in \text{Aut}(T_{\Phi(p)}M),$$

i.e.  $a(p)$  is an automorphism of the tangent space to  $M$  at the point  $\Phi(p)$ .

The standard case of a field of tangent-space automorphisms *on* a manifold corresponds to  $U = M$  and  $\Phi = \text{Id}_M$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $M$  ( $U$  is then an open interval of  $\mathbf{R}$ ).

---

**Note:** If  $M$  is parallelizable, then `AutomorphismFieldParal` must be used instead.

---

INPUT:

- `vector_field_module` – module  $\mathcal{X}(U, \Phi)$  of vector fields along  $U$  with values on  $M$  via the map  $\Phi$
- `name` – (default: `None`) name given to the field
- `latex_name` – (default: `None`) LaTeX symbol to denote the field; if none is provided, the LaTeX symbol is set to `name`
- `is_identity` – (default: `False`) determines whether the constructed object is a field of identity automorphisms

EXAMPLES:

Field of tangent-space automorphisms on a non-parallelizable 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U') ; V = M.open_subset('V')
sage: M.declare_union(U,V)   # M is the union of U and V
sage: c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart()
sage: transf = c_xy.transition_map(c_uv, (x+y, x-y), intersection_name='W',
```

```

.....:                               restrictions1= x>0, restrictions2= u+v>0)
sage: inv = transf.inverse()
sage: a = M.automorphism_field('a') ; a
Field of tangent-space automorphisms a on the 2-dimensional
differentiable manifold M
sage: a.parent()
General linear group of the Module X(M) of vector fields on the
2-dimensional differentiable manifold M

```

We first define the components of  $a$  with respect to the coordinate frame on  $U$ :

```

sage: eU = c_xy.frame() ; eV = c_uv.frame()
sage: a[eU, :] = [[1, x], [0, 2]]

```

We then set the components with respect to the coordinate frame on  $V$  by extending the expressions of the components in the corresponding subframe on  $W = U \cap V$ :

```

sage: W = U.intersection(V)
sage: a.add_comp_by_continuation(eV, W, c_uv)

```

At this stage, the automorphisms field  $a$  is fully defined:

```

sage: a.display(eU)
a = d/dx*dx + x d/dx*dy + 2 d/dy*dy
sage: a.display(eV)
a = (1/4*u + 1/4*v + 3/2) d/du*du + (-1/4*u - 1/4*v - 1/2) d/du*dv
+ (1/4*u + 1/4*v - 1/2) d/dv*du + (-1/4*u - 1/4*v + 3/2) d/dv*dv

```

In particular, we may ask for its inverse on the whole manifold  $M$ :

```

sage: ia = a.inverse() ; ia
Field of tangent-space automorphisms a^(-1) on the 2-dimensional
differentiable manifold M
sage: ia.display(eU)
a^(-1) = d/dx*dx - 1/2*x d/dx*dy + 1/2 d/dy*dy
sage: ia.display(eV)
a^(-1) = (-1/8*u - 1/8*v + 3/4) d/du*du + (1/8*u + 1/8*v + 1/4) d/du*dv
+ (-1/8*u - 1/8*v + 1/4) d/dv*du + (1/8*u + 1/8*v + 3/4) d/dv*dv

```

Equivalently, one can use the power minus one to get the inverse:

```

sage: ia is a^(-1)
True

```

or the operator `~`:

```

sage: ia is ~a
True

```

### **inverse()**

Return the inverse automorphism of `self`.

#### EXAMPLES:

Inverse of a field of tangent-space automorphisms on a non-parallelizable 2-dimensional manifold:

```

sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U') ; V = M.open_subset('V')

```

```

sage: M.declare_union(U,V) # M is the union of U and V
sage: c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart()
sage: transf = c_xy.transition_map(c_uv, (x+y, x-y),
....:     intersection_name='W', restrictions1= x>0, restrictions2= u+v>0)
sage: inv = transf.inverse()
sage: a = M.automorphism_field('a')
sage: eU = c_xy.frame() ; eV = c_uv.frame()
sage: a[eU,:] = [[1,x], [0,2]]
sage: W = U.intersection(V)
sage: a.add_comp_by_continuation(eV, W, c_uv)
sage: ia = a.inverse() ; ia
Field of tangent-space automorphisms a(-1) on the 2-dimensional
differentiable manifold M
sage: a[eU,:], ia[eU,:]
(
[1 x] [ 1 -1/2*x]
[0 2], [ 0 1/2]
)
sage: a[eV,:], ia[eV,:]
(
[ 1/4*u + 1/4*v + 3/2 -1/4*u - 1/4*v - 1/2]
[ 1/4*u + 1/4*v - 1/2 -1/4*u - 1/4*v + 3/2],
[-1/8*u - 1/8*v + 3/4 1/8*u + 1/8*v + 1/4]
[-1/8*u - 1/8*v + 1/4 1/8*u + 1/8*v + 3/4]
)

```

Let us check that ia is indeed the inverse of a:

```

sage: s = a.contract(ia)
sage: s[eU,:], s[eV,:]
(
[1 0] [1 0]
[0 1], [0 1]
)
sage: s = ia.contract(a)
sage: s[eU,:], s[eV,:]
(
[1 0] [1 0]
[0 1], [0 1]
)

```

The result is cached:

```

sage: a.inverse() is ia
True

```

Instead of `inverse()`, one can use the power minus one to get the inverse:

```

sage: ia is a(-1)
True

```

or the operator `~`:

```

sage: ia is ~a
True

```

**restrict** (*subdomain*, *dest\_map=None*)

Return the restriction of `self` to some subdomain.

This is a redefinition of `sage.manifolds.differentiable.tensorfield.TensorField.restrict()` to take into account the identity map.

INPUT:

- `subdomain` – *DifferentiableManifold* open subset  $V$  of `self._domain`
- `dest_map` – (default: None) *DiffMap*; destination map  $\Phi : V \rightarrow N$ , where  $N$  is a subdomain of `self._codomain`; if None, the restriction of `self.base_module().destination_map()` to  $V$  is used

OUTPUT:

- a *AutomorphismField* representing the restriction

EXAMPLES:

Restrictions of an automorphism field on the 2-sphere:

```
sage: M = Manifold(2, 'S^2', start_index=1)
sage: U = M.open_subset('U') # the complement of the North pole
sage: stereoN.<x,y> = U.chart() # stereographic coordinates from the North
↳pole
sage: eN = stereoN.frame() # the associated vector frame
sage: V = M.open_subset('V') # the complement of the South pole
sage: stereoS.<u,v> = V.chart() # stereographic coordinates from the South
↳pole
sage: eS = stereoS.frame() # the associated vector frame
sage: transf = stereoN.transition_map(stereoS, (x/(x^2+y^2), y/(x^2+y^2)),
↳intersection_name='W', \
restrictions1= x^2+y^2!=0,
↳restrictions2= u^2+v^2!=0)
sage: inv = transf.inverse() # transformation from stereoS to stereoN
sage: W = U.intersection(V) # the complement of the North and South poles
sage: stereoN_W = W.atlas()[0] # restriction of stereographic coord. from
↳North pole to W
sage: stereoS_W = W.atlas()[1] # restriction of stereographic coord. from
↳South pole to W
sage: eN_W = stereoN_W.frame() ; eS_W = stereoS_W.frame()
sage: a = M.automorphism_field(name='a') ; a
Field of tangent-space automorphisms a on the 2-dimensional
differentiable manifold S^2
sage: a[eN,:] = [[1, atan(x^2+y^2)], [0,3]]
sage: a.add_comp_by_continuation(eS, W, chart=stereoS)
sage: a.restrict(U)
Field of tangent-space automorphisms a on the Open subset U of the
2-dimensional differentiable manifold S^2
sage: a.restrict(U)[eN,:]
[
1 arctan(x^2 + y^2)
0 3]
sage: a.restrict(V)
Field of tangent-space automorphisms a on the Open subset V of the
2-dimensional differentiable manifold S^2
sage: a.restrict(V)[eS,:]
[
(u^4 + 10*u^2*v^2 + v^4 + 2*(u^3*v - u*v^3)*arctan(1/(u^2 + v^2)))/(u^4 +
↳2*u^2*v^2 + v^4) - (4*u^3*v - 4*u*v^3 + (u^4 - 2*u^2*v^2 + v^4)*arctan(1/(u^
↳2 + v^2)))/(u^4 + 2*u^2*v^2 + v^4)
4*(u^2*v^2*arctan(1/(u^2 + v^2)) - u^3*v + u*v^3)/(u^4 +
↳2*u^2*v^2 + v^4) (3*u^4 - 2*u^2*v^2 + 3*v^4 - 2*(u^3*v - u*v^3)*arctan(1/(u^
↳2 + v^2)))/(u^4 + 2*u^2*v^2 + v^4)
]
sage: a.restrict(W)
```

```
Field of tangent-space automorphisms a on the Open subset W of the
2-dimensional differentiable manifold S^2
sage: a.restrict(W)[eN_W,:]
[
      1 arctan(x^2 + y^2)]
[
      0                               3]
```

Restrictions of the field of tangent-space identity maps:

```
sage: id = M.tangent_identity_field() ; id
Field of tangent-space identity maps on the 2-dimensional
differentiable manifold S^2
sage: id.restrict(U)
Field of tangent-space identity maps on the Open subset U of the
2-dimensional differentiable manifold S^2
sage: id.restrict(U)[eN,:]
[1 0]
[0 1]
sage: id.restrict(V)
Field of tangent-space identity maps on the Open subset V of the
2-dimensional differentiable manifold S^2
sage: id.restrict(V)[eS,:]
[1 0]
[0 1]
sage: id.restrict(W)[eN_W,:]
[1 0]
[0 1]
sage: id.restrict(W)[eS_W,:]
[1 0]
[0 1]
```

**class** sage.manifolds.differentiable.automorphismfield.**AutomorphismFieldParal** (*vector\_field\_module*, *name=None*, *latex\_name=None*, *tex\_name=None*, *is\_identity=False*)

Bases: sage.tensor.modules.free\_module\_automorphism.FreeModuleAutomorphism, *sage.manifolds.differentiable.tensorfield\_paral.TensorFieldParal*

Field of tangent-space automorphisms with values on a parallelizable manifold.

Given a differentiable manifold  $U$  and a differentiable map  $\Phi : U \rightarrow M$  to a parallelizable manifold  $M$ , a *field of tangent-space automorphisms along  $U$  with values on  $M \supset \Phi(U)$*  is a differentiable map

$$a : U \longrightarrow T^{(1,1)}M$$

( $T^{(1,1)}M$  being the tensor bundle of type  $(1, 1)$  over  $M$ ) such that

$$\forall p \in U, a(p) \in \text{Aut}(T_{\Phi(p)}M)$$

i.e.  $a(p)$  is an automorphism of the tangent space to  $M$  at the point  $\Phi(p)$ .

The standard case of a field of tangent-space automorphisms *on* a manifold corresponds to  $U = M$  and  $\Phi = \text{Id}_M$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $M$  ( $U$  is then an open interval of  $\mathbf{R}$ ).

---

**Note:** If  $M$  is not parallelizable, the class *AutomorphismField* must be used instead.

---

INPUT:

- `vector_field_module` – free module  $\mathcal{X}(U, \Phi)$  of vector fields along  $U$  with values on  $M$  via the map  $\Phi$
- `name` – (default: None) name given to the field
- `latex_name` – (default: None) LaTeX symbol to denote the field; if none is provided, the LaTeX symbol is set to `name`
- `is_identity` – (default: False) determines whether the constructed object is a field of identity automorphisms

## EXAMPLES:

A  $\pi/3$ -rotation in the Euclidean 2-plane:

```
sage: M = Manifold(2, 'R^2')
sage: c_xy.<x,y> = M.chart()
sage: rot = M.automorphism_field('R') ; rot
Field of tangent-space automorphisms R on the 2-dimensional
differentiable manifold R^2
sage: rot[:] = [[sqrt(3)/2, -1/2], [1/2, sqrt(3)/2]]
sage: rot.parent()
General linear group of the Free module X(R^2) of vector fields on the
2-dimensional differentiable manifold R^2
```

The inverse automorphism is obtained via the method `inverse()`:

```
sage: inv = rot.inverse() ; inv
Field of tangent-space automorphisms R^(-1) on the 2-dimensional
differentiable manifold R^2
sage: latex(inv)
R^{-1}
sage: inv[:]
[1/2*sqrt(3)      1/2]
[      -1/2 1/2*sqrt(3)]
sage: rot[:]
[1/2*sqrt(3)      -1/2]
[      1/2 1/2*sqrt(3)]
sage: inv[:] * rot[:] # check
[1 0]
[0 1]
```

Equivalently, one can use the power minus one to get the inverse:

```
sage: inv is rot^(-1)
True
```

or the operator `~`:

```
sage: inv is ~rot
True
```

**at** (*point*)

Value of `self` at a given point.

If the current field of tangent-space automorphisms is

$$a : U \longrightarrow T^{(1,1)}M$$

associated with the differentiable map

$$\Phi : U \longrightarrow M,$$

where  $U$  and  $M$  are two manifolds (possibly  $U = M$  and  $\Phi = \text{Id}_M$ ), then for any point  $p \in U$ ,  $a(p)$  is an automorphism of the tangent space  $T_{\Phi(p)}M$ .

INPUT:

- point – *ManifoldPoint*; point  $p$  in the domain of the field of automorphisms  $a$

OUTPUT:

- the automorphism  $a(p)$  of the tangent vector space  $T_{\Phi(p)}M$

EXAMPLES:

Automorphism at some point of a tangent space of a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart()
sage: a = M.automorphism_field(name='a')
sage: a[:] = [[1+exp(y), x*y], [0, 1+x^2]]
sage: a.display()
a = (e^y + 1) d/dx*dx + x*y d/dx*dy + (x^2 + 1) d/dy*dy
sage: p = M.point((-2,3), name='p') ; p
Point p on the 2-dimensional differentiable manifold M
sage: ap = a.at(p) ; ap
Automorphism a of the Tangent space at Point p on the
2-dimensional differentiable manifold M
sage: ap.display()
a = (e^3 + 1) d/dx*dx - 6 d/dx*dy + 5 d/dy*dy
sage: ap.parent()
General linear group of the Tangent space at Point p on the
2-dimensional differentiable manifold M
```

The identity map of the tangent space at point p:

```
sage: id = M.tangent_identity_field() ; id
Field of tangent-space identity maps on the 2-dimensional
differentiable manifold M
sage: idp = id.at(p) ; idp
Identity map of the Tangent space at Point p on the 2-dimensional
differentiable manifold M
sage: idp is M.tangent_space(p).identity_map()
True
sage: idp.display()
Id = d/dx*dx + d/dy*dy
sage: idp.parent()
General linear group of the Tangent space at Point p on the
2-dimensional differentiable manifold M
sage: idp * ap == ap
True
```

**inverse()**

Return the inverse automorphism of self.

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
```

```

sage: a = M.automorphism_field(name='a')
sage: a[:] = [[0, 2], [-1, 0]]
sage: b = a.inverse(); b
Field of tangent-space automorphisms a^(-1) on the 2-dimensional
differentiable manifold M
sage: b[:]
[ 0 -1]
[1/2 0]
sage: a[:]
[ 0 2]
[-1 0]

```

The result is cached:

```

sage: a.inverse() is b
True

```

Instead of `inverse()`, one can use the power minus one to get the inverse:

```

sage: b is a^(-1)
True

```

or the operator `~`:

```

sage: b is ~a
True

```

**restrict** (*subdomain*, *dest\_map=None*)

Return the restriction of `self` to some subset of its domain.

If such restriction has not been defined yet, it is constructed here.

This is a redefinition of `sage.manifolds.differentiable.tensorfield_paral.TensorFieldParal.restrict()` to take into account the identity map.

INPUT:

- *subdomain* – *DifferentiableManifold*; open subset  $V$  of `self._domain`
- *dest\_map* – (default: `None`) *DiffMap* destination map  $\Phi : V \rightarrow N$ , where  $N$  is a subset of `self._codomain`; if `None`, the restriction of `self.base_module().destination_map()` to  $V$  is used

OUTPUT:

- a *AutomorphismFieldParal* representing the restriction

EXAMPLES:

Restriction of an automorphism field defined on  $\mathbb{R}^2$  to a disk:

```

sage: M = Manifold(2, 'R^2')
sage: c_cart.<x,y> = M.chart() # Cartesian coordinates on R^2
sage: D = M.open_subset('D') # the unit open disc
sage: c_cart_D = c_cart.restrict(D, x^2+y^2<1)
sage: a = M.automorphism_field(name='a') ; a
Field of tangent-space automorphisms a on the 2-dimensional
differentiable manifold R^2
sage: a[:] = [[1, x*y], [0, 3]]
sage: a.restrict(D)
Field of tangent-space automorphisms a on the Open subset D of the

```

```

2-dimensional differentiable manifold R^2
sage: a.restrict(D)[:]
[ 1 x*y]
[ 0  3]

```

Restriction to the disk of the field of tangent-space identity maps:

```

sage: id = M.tangent_identity_field() ; id
Field of tangent-space identity maps on the 2-dimensional
differentiable manifold R^2
sage: id.restrict(D)
Field of tangent-space identity maps on the Open subset D of the
2-dimensional differentiable manifold R^2
sage: id.restrict(D)[:]
[1 0]
[0 1]
sage: id.restrict(D) == D.tangent_identity_field()
True

```

## 2.8 Tensor Fields

### 2.8.1 Tensor Field Modules

The set of tensor fields along a differentiable manifold  $U$  with values on a differentiable manifold  $M$  via a differentiable map  $\Phi : U \rightarrow M$  (possibly  $U = M$  and  $\Phi = \text{Id}_M$ ) is a module over the algebra  $C^k(U)$  of differentiable scalar fields on  $U$ . It is a free module if and only if  $M$  is parallelizable. Accordingly, two classes are devoted to tensor field modules:

- *TensorFieldModule* for tensor fields with values on a generic (in practice, not parallelizable) differentiable manifold  $M$ ,
- *TensorFieldFreeModule* for tensor fields with values on a parallelizable manifold  $M$ .

AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2014-2015): initial version
- Travis Scrimshaw (2016): review tweaks

REFERENCES:

- [KN1963]
- [Lee2013]
- [ONe1983]

**class** `sage.manifolds.differentiable.tensorfield_module.TensorFieldFreeModule` (*vector\_field\_module*,  
*ten-*  
*sor\_type*)

Bases: `sage.tensor.modules.tensor_free_module.TensorFreeModule`

Free module of tensor fields of a given type  $(k, l)$  along a differentiable manifold  $U$  with values on a parallelizable manifold  $M$ , via a differentiable map  $U \rightarrow M$ .

Given two non-negative integers  $k$  and  $l$  and a differentiable map

$$\Phi : U \longrightarrow M,$$

the *tensor field module*  $T^{(k,l)}(U, \Phi)$  is the set of all tensor fields of the type

$$t : U \longrightarrow T^{(k,l)}M$$

(where  $T^{(k,l)}M$  is the tensor bundle of type  $(k, l)$  over  $M$ ) such that

$$t(p) \in T^{(k,l)}(T_{\Phi(p)}M)$$

for all  $p \in U$ , i.e.  $t(p)$  is a tensor of type  $(k, l)$  on the tangent vector space  $T_{\Phi(p)}M$ . Since  $M$  is parallelizable, the set  $T^{(k,l)}(U, \Phi)$  is a free module over  $C^k(U)$ , the ring (algebra) of differentiable scalar fields on  $U$  (see `DiffScalarFieldAlgebra`).

The standard case of tensor fields *on* a differentiable manifold corresponds to  $U = M$  and  $\Phi = \text{Id}_M$ ; we then denote  $T^{(k,l)}(M, \text{Id}_M)$  by merely  $T^{(k,l)}(M)$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $M$  ( $U$  is then an open interval of  $\mathbf{R}$ ).

---

**Note:** If  $M$  is not parallelizable, the class `TensorFieldModule` should be used instead, for  $T^{(k,l)}(U, \Phi)$  is no longer a free module.

---

INPUT:

- `vector_field_module` – free module  $\mathcal{X}(U, \Phi)$  of vector fields along  $U$  associated with the map  $\Phi : U \rightarrow M$
- `tensor_type` – pair  $(k, l)$  with  $k$  being the contravariant rank and  $l$  the covariant rank

EXAMPLES:

Module of type-(2, 0) tensor fields on  $\mathbf{R}^3$ :

```
sage: M = Manifold(3, 'R^3')
sage: c_xyz.<x,y,z> = M.chart() # Cartesian coordinates
sage: T20 = M.tensor_field_module((2,0)) ; T20
Free module T^(2,0)(R^3) of type-(2,0) tensors fields on the
3-dimensional differentiable manifold R^3
```

$T^{(2,0)}(\mathbf{R}^3)$  is a module over the algebra  $C^k(\mathbf{R}^3)$ :

```
sage: T20.category()
Category of finite dimensional modules over Algebra of differentiable
scalar fields on the 3-dimensional differentiable manifold R^3
sage: T20.base_ring() is M.scalar_field_algebra()
True
```

$T^{(2,0)}(\mathbf{R}^3)$  is a free module:

```
sage: isinstance(T20, FiniteRankFreeModule)
True
```

because  $M = \mathbf{R}^3$  is parallelizable:

```
sage: M.is_manifestly_parallelizable()
True
```

The zero element:

```

sage: z = T20.zero() ; z
Tensor field zero of type (2,0) on the 3-dimensional differentiable
manifold R^3
sage: z[:]
[0 0 0]
[0 0 0]
[0 0 0]

```

A random element:

```

sage: t = T20.an_element() ; t
Tensor field of type (2,0) on the 3-dimensional differentiable
manifold R^3
sage: t[:]
[2 0 0]
[0 0 0]
[0 0 0]

```

The module  $T^{(2,0)}(\mathbf{R}^3)$  coerces to any module of type-(2,0) tensor fields defined on some subdomain of  $\mathbf{R}^3$ :

```

sage: U = M.open_subset('U', coord_def={c_xyz: x>0})
sage: T20U = U.tensor_field_module((2,0))
sage: T20U.has_coerce_map_from(T20)
True
sage: T20.has_coerce_map_from(T20U) # the reverse is not true
False
sage: T20U.coerce_map_from(T20)
Coercion map:
  From: Free module T^(2,0)(R^3) of type-(2,0) tensors fields on the 3-
↳dimensional differentiable manifold R^3
  To:   Free module T^(2,0)(U) of type-(2,0) tensors fields on the Open subset U
↳of the 3-dimensional differentiable manifold R^3

```

The coercion map is actually the *restriction* of tensor fields defined on  $\mathbf{R}^3$  to  $U$ .

There is also a coercion map from fields of tangent-space automorphisms to tensor fields of type (1, 1):

```

sage: T11 = M.tensor_field_module((1,1)) ; T11
Free module T^(1,1)(R^3) of type-(1,1) tensors fields on the
3-dimensional differentiable manifold R^3
sage: GL = M.automorphism_field_group() ; GL
General linear group of the Free module X(R^3) of vector fields on the
3-dimensional differentiable manifold R^3
sage: T11.has_coerce_map_from(GL)
True

```

An explicit call to this coercion map is:

```

sage: id = GL.one() ; id
Field of tangent-space identity maps on the 3-dimensional
differentiable manifold R^3
sage: tid = T11(id) ; tid
Tensor field Id of type (1,1) on the 3-dimensional differentiable
manifold R^3
sage: tid[:]
[1 0 0]
[0 1 0]
[0 0 1]

```

**Element**alias of `TensorFieldParal`

**class** `sage.manifolds.differentiable.tensorfield_module.TensorFieldModule` (*vector\_field\_module*,  
*ten-*  
*sor\_type*)

**Bases:** `sage.structure.unique_representation.UniqueRepresentation`, `sage.structure.parent.Parent`

Module of tensor fields of a given type  $(k, l)$  along a differentiable manifold  $U$  with values on a differentiable manifold  $M$ , via a differentiable map  $U \rightarrow M$ .

Given two non-negative integers  $k$  and  $l$  and a differentiable map

$$\Phi : U \longrightarrow M,$$

the *tensor field module*  $T^{(k,l)}(U, \Phi)$  is the set of all tensor fields of the type

$$t : U \longrightarrow T^{(k,l)}M$$

(where  $T^{(k,l)}M$  is the tensor bundle of type  $(k, l)$  over  $M$ ) such that

$$t(p) \in T^{(k,l)}(T_{\Phi(p)}M)$$

for all  $p \in U$ , i.e.  $t(p)$  is a tensor of type  $(k, l)$  on the tangent vector space  $T_{\Phi(p)}M$ . The set  $T^{(k,l)}(U, \Phi)$  is a module over  $C^k(U)$ , the ring (algebra) of differentiable scalar fields on  $U$  (see [DiffScalarFieldAlgebra](#)).

The standard case of tensor fields *on* a differentiable manifold corresponds to  $U = M$  and  $\Phi = \text{Id}_M$ ; we then denote  $T^{(k,l)}(M, \text{Id}_M)$  by merely  $T^{(k,l)}(M)$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $M$  ( $U$  is then an open interval of  $\mathbf{R}$ ).

---

**Note:** If  $M$  is parallelizable, the class `TensorFieldFreeModule` should be used instead.

---

**INPUT:**

- `vector_field_module` – module  $\mathcal{X}(U, \Phi)$  of vector fields along  $U$  associated with the map  $\Phi : U \rightarrow M$
- `tensor_type` – pair  $(k, l)$  with  $k$  being the contravariant rank and  $l$  the covariant rank

**EXAMPLES:**

Module of type-(2, 0) tensor fields on the 2-sphere:

```
sage: M = Manifold(2, 'M') # the 2-dimensional sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                               intersection_name='W', restrictions1= x^2+y^2!=0,
....:                               restrictions2= u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: W = U.intersection(V)
sage: T20 = M.tensor_field_module((2,0)); T20
Module T^(2,0)(M) of type-(2,0) tensors fields on the 2-dimensional
differentiable manifold M
```

$T^{(2,0)}(M)$  is a module over the algebra  $C^k(M)$ :

```
sage: T20.category()
Category of modules over Algebra of differentiable scalar fields on the
2-dimensional differentiable manifold M
sage: T20.base_ring() is M.scalar_field_algebra()
True
```

$T^{(2,0)}(M)$  is not a free module:

```
sage: isinstance(T20, FiniteRankFreeModule)
False
```

because  $M = S^2$  is not parallelizable:

```
sage: M.is_manifestly_parallelizable()
False
```

On the contrary, the module of type-(2, 0) tensor fields on  $U$  is a free module, since  $U$  is parallelizable (being a coordinate domain):

```
sage: T20U = U.tensor_field_module((2,0))
sage: isinstance(T20U, FiniteRankFreeModule)
True
sage: U.is_manifestly_parallelizable()
True
```

The zero element:

```
sage: z = T20.zero() ; z
Tensor field zero of type (2,0) on the 2-dimensional differentiable
manifold M
sage: z is T20(0)
True
sage: z[c_xy.frame(),:]
[0 0]
[0 0]
sage: z[c_uv.frame(),:]
[0 0]
[0 0]
```

The module  $T^{(2,0)}(M)$  coerces to any module of type-(2, 0) tensor fields defined on some subdomain of  $M$ , for instance  $T^{(2,0)}(U)$ :

```
sage: T20U.has_coerce_map_from(T20)
True
```

The reverse is not true:

```
sage: T20.has_coerce_map_from(T20U)
False
```

The coercion:

```
sage: T20U.coerce_map_from(T20)
Coercion map:
```

```

From: Module  $T^{(2,0)}(M)$  of type-(2,0) tensors fields on the 2-dimensional
↳differentiable manifold M
To: Free module  $T^{(2,0)}(U)$  of type-(2,0) tensors fields on the Open subset U
↳of the 2-dimensional differentiable manifold M

```

The coercion map is actually the *restriction* of tensor fields defined on  $M$  to  $U$ :

```

sage: t = M.tensor_field(2,0, name='t')
sage: eU = c_xy.frame() ; eV = c_uv.frame()
sage: t[eU,:] = [[2,0], [0,-3]]
sage: t.add_comp_by_continuation(eV, W, chart=c_uv)
sage: T20U(t) # the conversion map in action
Tensor field t of type (2,0) on the Open subset U of the 2-dimensional
differentiable manifold M
sage: T20U(t) is t.restrict(U)
True

```

There is also a coercion map from fields of tangent-space automorphisms to tensor fields of type-(1,1):

```

sage: T11 = M.tensor_field_module((1,1)) ; T11
Module  $T^{(1,1)}(M)$  of type-(1,1) tensors fields on the 2-dimensional
differentiable manifold M
sage: GL = M.automorphism_field_group() ; GL
General linear group of the Module  $X(M)$  of vector fields on the
2-dimensional differentiable manifold M
sage: T11.has_coerce_map_from(GL)
True

```

Explicit call to the coercion map:

```

sage: a = GL.one() ; a
Field of tangent-space identity maps on the 2-dimensional
differentiable manifold M
sage: a.parent()
General linear group of the Module  $X(M)$  of vector fields on the
2-dimensional differentiable manifold M
sage: ta = T11.coerce(a) ; ta
Tensor field Id of type (1,1) on the 2-dimensional differentiable
manifold M
sage: ta.parent()
Module  $T^{(1,1)}(M)$  of type-(1,1) tensors fields on the 2-dimensional
differentiable manifold M
sage: ta[eU,:] # ta on U
[1 0]
[0 1]
sage: ta[eV,:] # ta on V
[1 0]
[0 1]

```

### Element

alias of `TensorField`

### base\_module()

Return the vector field module on which `self` is constructed.

### OUTPUT:

- a `VectorFieldModule` representing the module on which `self` is defined

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: T13 = M.tensor_field_module((1,3))
sage: T13.base_module()
Module X(M) of vector fields on the 2-dimensional differentiable
manifold M
sage: T13.base_module() is M.vector_field_module()
True
sage: T13.base_module().base_ring()
Algebra of differentiable scalar fields on the 2-dimensional
differentiable manifold M
```

**tensor\_type()**

Return the tensor type of `self`.

OUTPUT:

- pair  $(k, l)$  of non-negative integers such that the tensor fields belonging to this module are of type  $(k, l)$

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: T13 = M.tensor_field_module((1,3))
sage: T13.tensor_type()
(1, 3)
sage: T20 = M.tensor_field_module((2,0))
sage: T20.tensor_type()
(2, 0)
```

**zero()**

Return the zero of `self`.

## 2.8.2 Tensor Fields

The class *TensorField* implements tensor fields on differentiable manifolds. The derived class *TensorFieldParal* is devoted to tensor fields with values on parallelizable manifolds.

Various derived classes of *TensorField* are devoted to specific tensor fields:

- *VectorField* for vector fields (rank-1 contravariant tensor fields)
- *AutomorphismField* for fields of tangent-space automorphisms
- *DiffForm* for differential forms (fully antisymmetric covariant tensor fields)

AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2013-2015) : initial version
- Travis Scrimshaw (2016): review tweaks

REFERENCES:

- [KN1963]
- [Lee2013]
- [ONe1983]

```
class sage.manifolds.differentiable.tensorfield.TensorField(vector_field_module,
                                                         tensor_type,
                                                         name=None,      la-
                                                         latex_name=None,
                                                         sym=None,      an-
                                                         tisym=None,     par-
                                                         ent=None)
```

Bases: `sage.structure.element.ModuleElement`

Tensor field along a differentiable manifold.

An instance of this class is a tensor field along a differentiable manifold  $U$  with values on a differentiable manifold  $M$ , via a differentiable map  $\Phi : U \rightarrow M$ . More precisely, given two non-negative integers  $k$  and  $l$  and a differentiable map

$$\Phi : U \longrightarrow M,$$

a tensor field of type  $(k, l)$  along  $U$  with values on  $M$  is a differentiable map

$$t : U \longrightarrow T^{(k,l)}M$$

(where  $T^{(k,l)}M$  is the tensor bundle of type  $(k, l)$  over  $M$ ) such that

$$\forall p \in U, t(p) \in T^{(k,l)}(T_qM)$$

i.e.  $t(p)$  is a tensor of type  $(k, l)$  on the tangent space  $T_qM$  at the point  $q = \Phi(p)$ , that is to say a multilinear map

$$t(p) : \underbrace{T_q^*M \times \cdots \times T_q^*M}_k \times \underbrace{T_qM \times \cdots \times T_qM}_l \longrightarrow K$$

where  $T_q^*M$  is the dual vector space to  $T_qM$  and  $K$  is the topological field over which the manifold  $M$  is defined. The integer  $k + l$  is called the *tensor rank*.

The standard case of a tensor field on a differentiable manifold corresponds to  $U = M$  and  $\Phi = \text{Id}_M$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $M$  ( $U$  is then an open interval of  $\mathbf{R}$ ).

If  $M$  is parallelizable, the class `TensorFieldParal` should be used instead.

This is a Sage *element* class, the corresponding *parent* class being `TensorFieldModule`.

INPUT:

- `vector_field_module` – module  $\mathcal{X}(U, \Phi)$  of vector fields along  $U$  associated with the map  $\Phi : U \rightarrow M$  (cf. `VectorFieldModule`)
- `tensor_type` – pair  $(k, l)$  with  $k$  being the contravariant rank and  $l$  the covariant rank
- `name` – (default: `None`) name given to the tensor field
- `latex_name` – (default: `None`) LaTeX symbol to denote the tensor field; if none is provided, the LaTeX symbol is set to `name`
- `sym` – (default: `None`) a symmetry or a list of symmetries among the tensor arguments: each symmetry is described by a tuple containing the positions of the involved arguments, with the convention `position = 0` for the first argument; for instance:
  - `sym = (0, 1)` for a symmetry between the 1st and 2nd arguments
  - `sym = [(0, 2), (1, 3, 4)]` for a symmetry between the 1st and 3rd arguments and a symmetry between the 2nd, 4th and 5th arguments.

- `antisym` – (default: None) antisymmetry or list of antisymmetries among the arguments, with the same convention as for `sym`
- `parent` – (default: None) some specific parent (e.g. exterior power for differential forms); if None, `vector_field_module.tensor_module(k, l)` is used

EXAMPLES:

Tensor field of type (0,2) on the sphere  $S^2$ :

```
sage: M = Manifold(2, 'S^2') # the 2-dimensional sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                                     intersection_name='W', restrictions1= x^2+y^2!=0,
....:                                     restrictions2= u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: W = U.intersection(V)
sage: t = M.tensor_field(0,2, name='t') ; t
Tensor field t of type (0,2) on the 2-dimensional differentiable
manifold S^2
sage: t.parent()
Module T^(0,2)(S^2) of type-(0,2) tensors fields on the 2-dimensional
differentiable manifold S^2
sage: t.parent().category()
Category of modules over Algebra of differentiable scalar fields on the
2-dimensional differentiable manifold S^2
```

The parent of  $t$  is not a free module, for the sphere  $S^2$  is not parallelizable:

```
sage: isinstance(t.parent(), FiniteRankFreeModule)
False
```

To fully define  $t$ , we have to specify its components in some vector frames defined on subsets of  $S^2$ ; let us start by the open subset  $U$ :

```
sage: eU = c_xy.frame()
sage: t[eU,:] = [[1,0], [-2,3]]
sage: t.display(eU)
t = dx*dx - 2 dy*dx + 3 dy*dy
```

To set the components of  $t$  on  $V$  consistently, we copy the expressions of the components in the common subset  $W$ :

```
sage: eV = c_uv.frame()
sage: eVW = eV.restrict(W)
sage: c_uvW = c_uv.restrict(W)
sage: t[eV,0,0] = t[eVW,0,0,c_uvW].expr() # long time
sage: t[eV,0,1] = t[eVW,0,1,c_uvW].expr() # long time
sage: t[eV,1,0] = t[eVW,1,0,c_uvW].expr() # long time
sage: t[eV,1,1] = t[eVW,1,1,c_uvW].expr() # long time
```

Actually, the above operation can be performed in a single line by means of the method `add_comp_by_continuation()`:

```
sage: t.add_comp_by_continuation(eV, W, chart=c_uv) # long time
```

At this stage,  $t$  is fully defined, having components in frames  $eU$  and  $eV$  and the union of the domains of  $eU$  and  $eV$  being whole manifold:

```
sage: t.display(eV) # long time
t = (u^4 - 4*u^3*v + 10*u^2*v^2 + 4*u*v^3 + v^4)/(u^8 + 4*u^6*v^2 + 6*u^4*v^4 +
↪4*u^2*v^6 + v^8) du*du
- 4*(u^3*v + 2*u^2*v^2 - u*v^3)/(u^8 + 4*u^6*v^2 + 6*u^4*v^4 + 4*u^2*v^6 + v^8)
↪du*dv
+ 2*(u^4 - 2*u^3*v - 2*u^2*v^2 + 2*u*v^3 + v^4)/(u^8 + 4*u^6*v^2 + 6*u^4*v^4 +
↪4*u^2*v^6 + v^8) dv*du
+ (3*u^4 + 4*u^3*v - 2*u^2*v^2 - 4*u*v^3 + 3*v^4)/(u^8 + 4*u^6*v^2 + 6*u^4*v^4 +
↪4*u^2*v^6 + v^8) dv*dv
```

Let us consider two vector fields,  $a$  and  $b$ , on  $S^2$ :

```
sage: a = M.vector_field(name='a')
sage: a[eU, :] = [-y, x]
sage: a.add_comp_by_continuation(eV, W, chart=c_uv)
sage: a.display(eV)
a = -v d/du + u d/dv
sage: b = M.vector_field(name='b')
sage: b[eU, :] = [y, -1]
sage: b.add_comp_by_continuation(eV, W, chart=c_uv)
sage: b.display(eV)
b = ((2*u + 1)*v^3 + (2*u^3 - u^2)*v)/(u^2 + v^2) d/du
- (u^4 - v^4 + 2*u*v^2)/(u^2 + v^2) d/dv
```

As a tensor field of type  $(0, 2)$ ,  $t$  acts on the pair  $(a, b)$ , resulting in a scalar field:

```
sage: f = t(a,b); f
Scalar field t(a,b) on the 2-dimensional differentiable manifold S^2
sage: f.display() # long time
t(a,b): S^2 --> R
on U: (x, y) |--> -2*x*y - y^2 - 3*x
on V: (u, v) |--> -(3*u^3 + (3*u + 1)*v^2 + 2*u*v)/(u^4 + 2*u^2*v^2 + v^4)
```

The vectors can be defined only on subsets of  $S^2$ , the domain of the result is then the common subset:

```
sage: s = t(a.restrict(U), b) ; s # long time
Scalar field t(a,b) on the Open subset U of the 2-dimensional
differentiable manifold S^2
sage: s.display() # long time
t(a,b): U --> R
(x, y) |--> -2*x*y - y^2 - 3*x
on W: (u, v) |--> -(3*u^3 + (3*u + 1)*v^2 + 2*u*v)/(u^4 + 2*u^2*v^2 + v^4)
sage: s = t(a.restrict(U), b.restrict(W)) ; s # long time
Scalar field t(a,b) on the Open subset W of the 2-dimensional
differentiable manifold S^2
sage: s.display() # long time
t(a,b): W --> R
(x, y) |--> -2*x*y - y^2 - 3*x
(u, v) |--> -(3*u^3 + (3*u + 1)*v^2 + 2*u*v)/(u^4 + 2*u^2*v^2 + v^4)
```

The tensor itself can be defined only on some open subset of  $S^2$ , yielding a result whose domain is this subset:

```

sage: s = t.restrict(V)(a,b); s # long time
Scalar field t(a,b) on the Open subset V of the 2-dimensional
differentiable manifold S^2
sage: s.display() # long time
t(a,b): V --> R
      (u, v) |--> -(3*u^3 + (3*u + 1)*v^2 + 2*u*v)/(u^4 + 2*u^2*v^2 + v^4)
on W: (x, y) |--> -2*x*y - y^2 - 3*x

```

Tests regarding the multiplication by a scalar field:

```

sage: f = M.scalar_field({c_xy: 1/(1+x^2+y^2),
....:                    c_uv: (u^2 + v^2)/(u^2 + v^2 + 1)}, name='f')
sage: t.parent().base_ring() is f.parent()
True
sage: s = f*t; s # long time
Tensor field of type (0,2) on the 2-dimensional differentiable
manifold S^2
sage: s[[0,0]] == f*t[[0,0]] # long time
True
sage: s.restrict(U) == f.restrict(U) * t.restrict(U) # long time
True
sage: s = f*t.restrict(U); s
Tensor field of type (0,2) on the Open subset U of the 2-dimensional
differentiable manifold S^2
sage: s.restrict(U) == f.restrict(U) * t.restrict(U)
True

```

**add\_comp** (*basis=None*)

Return the components of *self* in a given vector frame for assignment.

The components with respect to other frames having the same domain as the provided vector frame are kept. To delete them them, use the method `set_comp()` instead.

INPUT:

- *basis* – (default: None) vector frame in which the components are defined; if None, the components are assumed to refer to the tensor field domain’s default frame

OUTPUT:

- components in the given frame, as a Component *s*; if such components did not exist previously, they are created

EXAMPLES:

```

sage: M = Manifold(2, 'M') # the 2-dimensional sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: e_uv = c_uv.frame()
sage: t = M.tensor_field(1, 2, name='t')
sage: t.add_comp(e_uv)
3-indices components w.r.t. Coordinate frame (V, (d/du,d/dv))
sage: t.add_comp(e_uv)[1,0,1] = u+v
sage: t.display(e_uv)
t = (u + v) d/dv*du*dv

```

Setting the components in a new frame:

```

sage: e = V.vector_frame('e')
sage: t.add_comp(e)
3-indices components w.r.t. Vector frame (V, (e_0,e_1))
sage: t.add_comp(e)[0,1,1] = u*v
sage: t.display(e)
t = u*v e_0*e^1*e^1

```

The components with respect to  $e_{uv}$  are kept:

```

sage: t.display(e_uv)
t = (u + v) d/dv*du*dv

```

### `add_comp_by_continuation` (*frame, subdomain, chart=None*)

Set components with respect to a vector frame by continuation of the coordinate expression of the components in a subframe.

The continuation is performed by demanding that the components have the same coordinate expression as those on the restriction of the frame to a given subdomain.

INPUT:

- `frame` – vector frame  $e$  in which the components are to be set
- `subdomain` – open subset of  $e$ 's domain in which the components are known or can be evaluated from other components
- `chart` – (default: `None`) coordinate chart on  $e$ 's domain in which the extension of the expression of the components is to be performed; if `None`, the default's chart of  $e$ 's domain is assumed

EXAMPLES:

Components of a vector field on the sphere  $S^2$ :

```

sage: M = Manifold(2, 'S^2', start_index=1)
sage: # The two open subsets covered by stereographic coordinates (North and
↪South):
sage: U = M.open_subset('U') ; V = M.open_subset('V')
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart() # stereographic
↪coordinates
sage: transf = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                               intersection_name='W', restrictions1= x^2+y^2!=0,
....:                               restrictions2= u^2+v^2!=0)
sage: inv = transf.inverse()
sage: W = U.intersection(V) # The complement of the two poles
sage: eU = c_xy.frame() ; eV = c_uv.frame()
sage: a = M.vector_field('a')
sage: a[eU,:] = [x, 2+y]

```

At this stage, the vector field has been defined only on the open subset  $U$  (through its components in the frame  $eU$ ):

```

sage: a.display(eU)
a = x d/dx + (y + 2) d/dy

```

The components with respect to the restriction of  $eV$  to the common subdomain  $W$ , in terms of the  $(u, v)$  coordinates, are obtained by a change-of-frame formula on  $W$ :

```

sage: a.display(eV.restrict(W), c_uv.restrict(W))
a = (-4*u*v - u) d/du + (2*u^2 - 2*v^2 - v) d/dv

```

The continuation consists in extending the definition of the vector field to the whole open subset  $V$  by demanding that the components in the frame  $eV$  have the same coordinate expression as the above one:

```
sage: a.add_comp_by_continuation(eV, W, chart=c_uv)
```

We have then:

```
sage: a.display(eV)
a = (-4*u*v - u) d/du + (2*u^2 - 2*v^2 - v) d/dv
```

and  $a$  is defined on the entire manifold  $S^2$ .

### **antisymmetrize** (\*pos)

Antisymmetrization over some arguments.

INPUT:

- `pos` – (default: None) list of argument positions involved in the antisymmetrization (with the convention `position=0` for the first argument); if None, the antisymmetrization is performed over all the arguments

OUTPUT:

- the antisymmetrized tensor field (instance of *TensorField*)

EXAMPLES:

Antisymmetrization of a type-(0, 2) tensor field on a 2-dimensional non-parallelizable manifold:

```
sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U') ; V = M.open_subset('V')
sage: M.declare_union(U,V) # M is the union of U and V
sage: c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart()
sage: transf = c_xy.transition_map(c_uv, (x+y, x-y), intersection_name='W',
....:                               restrictions1= x>0, restrictions2= u+v>0)
sage: inv = transf.inverse()
sage: W = U.intersection(V)
sage: eU = c_xy.frame() ; eV = c_uv.frame()
sage: a = M.tensor_field(0,2, name='a')
sage: a[eU, :] = [[1,x], [2,y]]
sage: a.add_comp_by_continuation(eV, W, chart=c_uv)
sage: a[eV, :]
[ 1/4*u + 3/4 -1/4*u + 3/4]
[ 1/4*v - 1/4 -1/4*v - 1/4]
sage: s = a.antisymmetrize() ; s
2-form on the 2-dimensional differentiable manifold M
sage: s[eU, :]
[      0  1/2*x - 1]
[-1/2*x + 1      0]
sage: s[eV, :]
[      0 -1/8*u - 1/8*v + 1/2]
[ 1/8*u + 1/8*v - 1/2      0]
sage: s == a.antisymmetrize(0,1) # explicit positions
True
sage: s == a.antisymmetrize(1,0) # the order of positions does not matter
True
```

**See also:**

For more details and examples, see `sage.tensor.modules.free_module_tensor.FreeModuleTensor.antisymmetrize()`.

**at** (*point*)

Value of `self` at a point of its domain.

If the current tensor field is

$$t : U \longrightarrow T^{(k,l)}M$$

associated with the differentiable map

$$\Phi : U \longrightarrow M,$$

where  $U$  and  $M$  are two manifolds (possibly  $U = M$  and  $\Phi = \text{Id}_M$ ), then for any point  $p \in U$ ,  $t(p)$  is a tensor on the tangent space to  $M$  at the point  $\Phi(p)$ .

INPUT:

- `point` – *ManifoldPoint*; point  $p$  in the domain of the tensor field  $U$

OUTPUT:

- `FreeModuleTensor` representing the tensor  $t(p)$  on the tangent vector space  $T_{\Phi(p)}M$

EXAMPLES:

Tensor on a tangent space of a non-parallelizable 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U') ; V = M.open_subset('V')
sage: M.declare_union(U,V) # M is the union of U and V
sage: c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart()
sage: transf = c_xy.transition_map(c_uv, (x+y, x-y),
....:                               intersection_name='W', restrictions1= x>0,
....:                               restrictions2= u+v>0)
sage: inv = transf.inverse()
sage: W = U.intersection(V)
sage: eU = c_xy.frame() ; eV = c_uv.frame()
sage: a = M.tensor_field(1,1, name='a')
sage: a[eU,:] = [[1+y,x], [0,x+y]]
sage: a.add_comp_by_continuation(eV, W, chart=c_uv)
sage: a.display(eU)
a = (y + 1) d/dx*dx + x d/dx*dy + (x + y) d/dy*dy
sage: a.display(eV)
a = (u + 1/2) d/du*du + (-1/2*u - 1/2*v + 1/2) d/du*dv
  + 1/2 d/dv*du + (1/2*u - 1/2*v + 1/2) d/dv*dv
sage: p = M.point((2,3), chart=c_xy, name='p')
sage: ap = a.at(p) ; ap
Type-(1,1) tensor a on the Tangent space at Point p on the
2-dimensional differentiable manifold M
sage: ap.parent()
Free module of type-(1,1) tensors on the Tangent space at Point p
on the 2-dimensional differentiable manifold M
sage: ap.display(eU.at(p))
a = 4 d/dx*dx + 2 d/dx*dy + 5 d/dy*dy
sage: ap.display(eV.at(p))
a = 11/2 d/du*du - 3/2 d/du*dv + 1/2 d/dv*du + 7/2 d/dv*dv
sage: p.coord(c_uv) # to check the above expression
(5, -1)
```

**base\_module** ()

Return the vector field module on which `self` acts as a tensor.

OUTPUT:

- instance of *VectorFieldModule*

## EXAMPLES:

The module of vector fields on the 2-sphere as a “base module”:

```
sage: M = Manifold(2, 'S^2')
sage: t = M.tensor_field(0,2)
sage: t.base_module()
Module X(S^2) of vector fields on the 2-dimensional differentiable
manifold S^2
sage: t.base_module() is M.vector_field_module()
True
sage: XM = M.vector_field_module()
sage: XM.an_element().base_module() is XM
True
```

**comp** (*basis=None, from\_basis=None*)

Return the components in a given vector frame.

If the components are not known already, they are computed by the tensor change-of-basis formula from components in another vector frame.

## INPUT:

- basis* – (default: None) vector frame in which the components are required; if none is provided, the components are assumed to refer to the tensor field domain’s default frame
- from\_basis* – (default: None) vector frame from which the required components are computed, via the tensor change-of-basis formula, if they are not known already in the *basis*

## OUTPUT:

- components in the vector frame *basis*, as a *Components*

## EXAMPLES:

Components of a type-(1,1) tensor field defined on two open subsets:

```
sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U')
sage: c_xy.<x, y> = U.chart()
sage: e = U.default_frame() ; e
Coordinate frame (U, (d/dx,d/dy))
sage: V = M.open_subset('V')
sage: c_uv.<u, v> = V.chart()
sage: f = V.default_frame() ; f
Coordinate frame (V, (d/du,d/dv))
sage: M.declare_union(U,V) # M is the union of U and V
sage: t = M.tensor_field(1,1, name='t')
sage: t[e,0,0] = - x + y^3
sage: t[e,0,1] = 2+x
sage: t[f,1,1] = - u*v
sage: t.comp(e)
2-indices components w.r.t. Coordinate frame (U, (d/dx,d/dy))
sage: t.comp(e)[:]
[y^3 - x  x + 2]
[      0      0]
sage: t.comp(f)
2-indices components w.r.t. Coordinate frame (V, (d/du,d/dv))
sage: t.comp(f)[:]
```

```
[ 0 0]
[ 0 -u*v]
```

Since  $e$  is  $M$ 's default frame, the argument  $e$  can be omitted:

```
sage: e is M.default_frame()
True
sage: t.comp() is t.comp(e)
True
```

Example of computation of the components via a change of frame:

```
sage: a = V.automorphism_field()
sage: a[:] = [[1+v, -u^2], [0, 1-u]]
sage: h = f.new_frame(a, 'h')
sage: t.comp(h)
2-indices components w.r.t. Vector frame (V, (h_0,h_1))
sage: t.comp(h)[:]
[ 0 -u^3*v/(v + 1)]
[ 0 -u*v]
```

### **contract** (\*args)

Contraction of `self` with another tensor field on one or more indices.

INPUT:

- `pos1` – positions of the indices in the current tensor field involved in the contraction; `pos1` must be a sequence of integers, with 0 standing for the first index position, 1 for the second one, etc.; if `pos1` is not provided, a single contraction on the last index position of the tensor field is assumed
- `other` – the tensor field to contract with
- `pos2` – positions of the indices in `other` involved in the contraction, with the same conventions as for `pos1`; if `pos2` is not provided, a single contraction on the first index position of `other` is assumed

OUTPUT:

- tensor field resulting from the contraction at the positions `pos1` and `pos2` of the tensor field with `other`

EXAMPLES:

Contractions of a type-(1,1) tensor field with a type-(2,0) one on a 2-dimensional non-parallelizable manifold:

```
sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U') ; V = M.open_subset('V')
sage: M.declare_union(U,V) # M is the union of U and V
sage: c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart()
sage: transf = c_xy.transition_map(c_uv, (x+y, x-y), intersection_name='W',
....:                               restrictions1= x>0, restrictions2= u+v>0)
sage: inv = transf.inverse()
sage: W = U.intersection(V)
sage: eU = c_xy.frame() ; eV = c_uv.frame()
sage: a = M.tensor_field(1,1, name='a')
sage: a[eU, :] = [[1, x], [0, 2]]
sage: a.add_comp_by_continuation(eV, W, chart=c_uv)
sage: b = M.tensor_field(2,0, name='b')
sage: b[eU, :] = [[y, -1], [x+y, 2]]
```

```

sage: b.add_comp_by_continuation(eV, W, chart=c_uv)
sage: s = a.contract(b) ; s # contraction on last index of a and first one_
↳ of b
Tensor field of type (2,0) on the 2-dimensional differentiable
manifold M

```

Check 1: components with respect to the manifold's default frame (eU):

```

sage: [[bool(s[i,j] == sum(a[i,k]*b[k,j] for k in M.irange()))
....:   for j in M.irange()] for i in M.irange()]
[[True, True], [True, True]]

```

Check 2: components with respect to the frame eV:

```

sage: [[bool(s[eV,i,j] == sum(a[eV,i,k]*b[eV,k,j]
....:   for k in M.irange()))
....:   for j in M.irange()] for i in M.irange()]
[[True, True], [True, True]]

```

Instead of the explicit call to the method `contract()`, one may use the index notation with Einstein convention (summation over repeated indices); it suffices to pass the indices as a string inside square brackets:

```

sage: a['^i_k']*b['^kj'] == s
True

```

Indices not involved in the contraction may be replaced by dots:

```

sage: a['^._k']*b['^k.']== s
True

```

LaTeX notation may be used:

```

sage: a['^{i}_{k}']*b['^{kj}'] == s
True

```

Contraction on the last index of a and last index of b:

```

sage: s = a.contract(b, 1) ; s
Tensor field of type (2,0) on the 2-dimensional differentiable
manifold M
sage: a['^i_k']*b['^jk'] == s
True

```

Contraction on the first index of b and the last index of a:

```

sage: s = b.contract(0,a,1) ; s
Tensor field of type (2,0) on the 2-dimensional differentiable
manifold M
sage: b['^ki']*a['^j_k'] == s
True

```

The domain of the result is the intersection of the domains of the two tensor fields:

```

sage: aU = a.restrict(U) ; bV = b.restrict(V)
sage: s = aU.contract(b) ; s
Tensor field of type (2,0) on the Open subset U of the

```

```

2-dimensional differentiable manifold M
sage: s = a.contract(bV) ; s
Tensor field of type (2,0) on the Open subset V of the
2-dimensional differentiable manifold M
sage: s = aU.contract(bV) ; s
Tensor field of type (2,0) on the Open subset W of the
2-dimensional differentiable manifold M
sage: s0 = a.contract(b)
sage: s == s0.restrict(W)
True

```

The contraction can be performed on more than one index:  $c$  being a type-(2,2) tensor, contracting the indices in positions 2 and 3 of  $c$  with respectively those in positions 0 and 1 of  $b$  is:

```

sage: c = a*a ; c
Tensor field of type (2,2) on the 2-dimensional differentiable
manifold M
sage: s = c.contract(2,3, b, 0,1) ; s # long time
Tensor field of type (2,0) on the 2-dimensional differentiable
manifold M
sage: s == c['^.._k1']*b['^k1'] # the same double contraction in index_
↪notation; long time
True

```

The symmetries are either conserved or destroyed by the contraction:

```

sage: c = c.symmetrize(0,1).antisymmetrize(2,3)
sage: c.symmetries()
symmetry: (0, 1); antisymmetry: (2, 3)
sage: s = b.contract(0, c, 2) ; s
Tensor field of type (3,1) on the 2-dimensional differentiable
manifold M
sage: s.symmetries()
symmetry: (1, 2); no antisymmetry

```

Case of a scalar field result:

```

sage: a = M.one_form('a')
sage: a[eU,:] = [y, 1+x]
sage: a.add_comp_by_continuation(eV, W, chart=c_uv)
sage: b = M.vector_field('b')
sage: b[eU,:] = [x, y^2]
sage: b.add_comp_by_continuation(eV, W, chart=c_uv)
sage: a.display(eU)
a = y dx + (x + 1) dy
sage: b.display(eU)
b = x d/dx + y^2 d/dy
sage: s = a.contract(b) ; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M --> R
on U: (x, y) |--> (x + 1)*y^2 + x*y
on V: (u, v) |--> 1/8*u^3 - 1/8*u*v^2 + 1/8*v^3 + 1/2*u^2 - 1/8*(u^2 + 4*u)*v
sage: s == a['_i']*b['^i'] # use of index notation
True
sage: s == b.contract(a)
True

```

Case of a vanishing scalar field result:

```
sage: b = M.vector_field('b')
sage: b[eU,:] = [1+x, -y]
sage: b.add_comp_by_continuation(eV, W, chart=c_uv)
sage: s = a.contract(b) ; s
Scalar field zero on the 2-dimensional differentiable manifold M
sage: s.display()
zero: M --> R
on U: (x, y) |--> 0
on V: (u, v) |--> 0
```

**copy()**

Return an exact copy of self.

---

**Note:** The name and the derived quantities are not copied.

---

EXAMPLES:

Copy of a type-(1,1) tensor field on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U') ; V = M.open_subset('V')
sage: M.declare_union(U,V) # M is the union of U and V
sage: c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart()
sage: xy_to_uv = c_xy.transition_map(c_uv, (x+y, x-y),
....:                                     intersection_name='W', restrictions1= x>0,
....:                                     restrictions2= u+v>0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: e_xy = c_xy.frame(); e_uv = c_uv.frame()
sage: t = M.tensor_field(1, 1, name='t')
sage: t[e_xy,:] = [[x+y, 0], [2, 1-y]]
sage: t.add_comp_by_continuation(e_uv, U.intersection(V), c_uv)
sage: s = t.copy(); s
Tensor field of type (1,1) on
the 2-dimensional differentiable manifold M
sage: s.display(e_xy)
(x + y) d/dx*dx + 2 d/dy*dx + (-y + 1) d/dy*dy
sage: s == t
True
```

If the original tensor field is modified, the copy is not:

```
sage: t[e_xy,0,0] = -1
sage: t.display(e_xy)
t = -d/dx*dx + 2 d/dy*dx + (-y + 1) d/dy*dy
sage: s.display(e_xy)
(x + y) d/dx*dx + 2 d/dy*dx + (-y + 1) d/dy*dy
sage: s == t
False
```

**disp** (*basis=None, chart=None*)

Display the tensor field in terms of its expansion with respect to a given vector frame.

The output is either text-formatted (console mode) or LaTeX-formatted (notebook mode).

INPUT:

- `basis` – (default: `None`) vector frame with respect to which the tensor is expanded; if `None`, the default frame of the domain of definition of the tensor field is assumed
- `chart` – (default: `None`) chart with respect to which the components of the tensor field in the selected frame are expressed; if `None`, the default chart of the vector frame domain is assumed

EXAMPLES:

Display of a type-(1, 1) tensor field defined on two open subsets:

```
sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U')
sage: c_xy.<x, y> = U.chart()
sage: e = U.default_frame() ; e
Coordinate frame (U, (d/dx,d/dy))
sage: V = M.open_subset('V')
sage: c_uv.<u, v> = V.chart()
sage: f = V.default_frame() ; f
Coordinate frame (V, (d/du,d/dv))
sage: M.declare_union(U,V) # M is the union of U and V
sage: t = M.tensor_field(1,1, name='t')
sage: t[e,0,0] = - x + y^3
sage: t[e,0,1] = 2+x
sage: t[f,1,1] = - u*v
sage: t.display(e)
t = (y^3 - x) d/dx*dx + (x + 2) d/dx*dy
sage: t.display(f)
t = -u*v d/dv*dv
```

Since `e` is `M`'s default frame, the argument `e` can be omitted:

```
sage: e is M.default_frame()
True
sage: t.display()
t = (y^3 - x) d/dx*dx + (x + 2) d/dx*dy
```

Similarly, since `f` is `V`'s default frame, the argument `f` can be omitted when considering the restriction of `t` to `V`:

```
sage: t.restrict(V).display()
t = -u*v d/dv*dv
```

Display with respect to a frame in which `t` has not been initialized (automatic use of a change-of-frame formula):

```
sage: a = V.automorphism_field()
sage: a[:] = [[1+v, -u^2], [0, 1-u]]
sage: h = f.new_frame(a, 'h')
sage: t.display(h)
t = -u^3*v/(v + 1) h_0*h^1 - u*v h_1*h^1
```

A shortcut of `display()` is `disp()`:

```
sage: t.disp(h)
t = -u^3*v/(v + 1) h_0*h^1 - u*v h_1*h^1
```

**display** (*basis=None, chart=None*)

Display the tensor field in terms of its expansion with respect to a given vector frame.

The output is either text-formatted (console mode) or LaTeX-formatted (notebook mode).

INPUT:

- `basis` – (default: `None`) vector frame with respect to which the tensor is expanded; if `None`, the default frame of the domain of definition of the tensor field is assumed
- `chart` – (default: `None`) chart with respect to which the components of the tensor field in the selected frame are expressed; if `None`, the default chart of the vector frame domain is assumed

EXAMPLES:

Display of a type-(1, 1) tensor field defined on two open subsets:

```
sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U')
sage: c_xy.<x, y> = U.chart()
sage: e = U.default_frame() ; e
Coordinate frame (U, (d/dx,d/dy))
sage: V = M.open_subset('V')
sage: c_uv.<u, v> = V.chart()
sage: f = V.default_frame() ; f
Coordinate frame (V, (d/du,d/dv))
sage: M.declare_union(U,V) # M is the union of U and V
sage: t = M.tensor_field(1,1, name='t')
sage: t[e,0,0] = - x + y^3
sage: t[e,0,1] = 2+x
sage: t[f,1,1] = - u*v
sage: t.display(e)
t = (y^3 - x) d/dx*dx + (x + 2) d/dx*dy
sage: t.display(f)
t = -u*v d/dv*dv
```

Since `e` is `M`'s default frame, the argument `e` can be omitted:

```
sage: e is M.default_frame()
True
sage: t.display()
t = (y^3 - x) d/dx*dx + (x + 2) d/dx*dy
```

Similarly, since `f` is `V`'s default frame, the argument `f` can be omitted when considering the restriction of `t` to `V`:

```
sage: t.restrict(V).display()
t = -u*v d/dv*dv
```

Display with respect to a frame in which `t` has not been initialized (automatic use of a change-of-frame formula):

```
sage: a = V.automorphism_field()
sage: a[:] = [[1+v, -u^2], [0, 1-u]]
sage: h = f.new_frame(a, 'h')
sage: t.display(h)
t = -u^3*v/(v + 1) h_0*h^1 - u*v h_1*h^1
```

A shortcut of `display()` is `disp()`:

```
sage: t.disp(h)
t = -u^3*v/(v + 1) h_0*h^1 - u*v h_1*h^1
```

**display\_comp** (*frame=None, chart=None, coordinate\_labels=True, only\_nonzero=True, only\_nonredundant=False*)

Display the tensor components with respect to a given frame, one per line.

The output is either text-formatted (console mode) or LaTeX-formatted (notebook mode).

INPUT:

- **frame** – (default: None) vector frame with respect to which the tensor field components are defined; if None, then
  - if **chart** is not None, the coordinate frame associated to **chart** is used
  - otherwise, the default basis of the vector field module on which the tensor field is defined is used
- **chart** – (default: None) chart specifying the coordinate expression of the components; if None, the default chart of the tensor field domain is used
- **coordinate\_labels** – (default: True) boolean; if True, coordinate symbols are used by default (instead of integers) as index labels whenever **frame** is a coordinate frame
- **only\_nonzero** – (default: True) boolean; if True, only nonzero components are displayed
- **only\_nonredundant** – (default: False) boolean; if True, only nonredundant components are displayed in case of symmetries

EXAMPLES:

Display of the components of a type-(1, 1) tensor field defined on two open subsets:

```
sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U')
sage: c_xy.<x, y> = U.chart()
sage: e = U.default_frame()
sage: V = M.open_subset('V')
sage: c_uv.<u, v> = V.chart()
sage: f = V.default_frame()
sage: M.declare_union(U,V) # M is the union of U and V
sage: t = M.tensor_field(1,1, name='t')
sage: t[e,0,0] = - x + y^3
sage: t[e,0,1] = 2+x
sage: t[f,1,1] = - u*v
sage: t.display_comp(e)
t^x_x = y^3 - x
t^x_y = x + 2
sage: t.display_comp(f)
t^v_v = -u*v
```

Components in a chart frame:

```
sage: t.display_comp(chart=c_xy)
t^x_x = y^3 - x
t^x_y = x + 2
sage: t.display_comp(chart=c_uv)
t^v_v = -u*v
```

See [documentation of `sage.manifolds.differentiable.tensorfield\_paral.TensorFieldParal.display\_comp\(\)`](#) for more options.

**domain** ()

Return the manifold on which **self** is defined.

OUTPUT:

- instance of class *DifferentiableManifold*

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart()
sage: t = M.tensor_field(1,2)
sage: t.domain()
2-dimensional differentiable manifold M
sage: U = M.open_subset('U', coord_def={c_xy: x<0})
sage: h = t.restrict(U)
sage: h.domain()
Open subset U of the 2-dimensional differentiable manifold M
```

**down** (*metric*, *pos=None*)

Compute a metric dual of the tensor field by lowering some index with a given metric.

If  $T$  is the tensor field,  $(k, l)$  its type and  $p$  the position of a contravariant index (i.e.  $0 \leq p < k$ ), this method called with `pos = p` yields the tensor field  $T^b$  of type  $(k - 1, l + 1)$  whose components are

$$(T^b)^{a_1 \dots a_{k-1}}_{b_1 \dots b_{l+1}} = g_{b_1 i} T^{a_1 \dots a_p i a_{p+1} \dots a_{k-1}}_{b_2 \dots b_{l+1}},$$

$g_{ab}$  being the components of the metric tensor.

The reverse operation is *TensorField.up()*

INPUT:

- metric* – metric  $g$ , as an instance of *PseudoRiemannianMetric*
- pos* – (default: `None`) position of the index (with the convention `pos=0` for the first index); if `None`, the lowering is performed over all the contravariant indices, starting from the last one

OUTPUT:

- the tensor field  $T^b$  resulting from the index lowering operation

EXAMPLES:

Lowering the index of a vector field results in a 1-form:

```
sage: M = Manifold(2, 'M', start_index=1)
sage: c_xy.<x,y> = M.chart()
sage: g = M.metric('g')
sage: g[1,1], g[1,2], g[2,2] = 1+x, x*y, 1-y
sage: v = M.vector_field()
sage: v[:] = [-1,2]
sage: w = v.down(g) ; w
1-form on the 2-dimensional differentiable manifold M
sage: w.display()
(2*x*y - x - 1) dx + (-(x + 2)*y + 2) dy
```

Using the index notation instead of *down()*:

```
sage: w == g['_ab']*v['^b']
True
```

The reverse operation:

```
sage: v1 = w.up(g) ; v1
Vector field on the 2-dimensional differentiable manifold M
```

```
sage: v1 == v
True
```

Lowering the indices of a tensor field of type (2,0):

```
sage: t = M.tensor_field(2, 0)
sage: t[:] = [[1,2], [3,4]]
sage: td0 = t.down(g, 0) ; td0 # lowering the first index
Tensor field of type (1,1) on the 2-dimensional differentiable
manifold M
sage: td0 == g['_ac']*t['^cb'] # the same operation in index notation
True
sage: td0[:]
[ 3*x*y + x + 1 (x - 3)*y + 3]
[4*x*y + 2*x + 2 2*(x - 2)*y + 4]
sage: tdd0 = td0.down(g) ; tdd0 # the two indices have been lowered, starting
↳from the first one
Tensor field of type (0,2) on the 2-dimensional differentiable
manifold M
sage: tdd0 == g['_ac']*td0['^c_b'] # the same operation in index notation
True
sage: tdd0[:]
[ 4*x^2*y^2 + x^2 + 5*(x^2 + x)*y + 2*x + 1 2*(x^2 - 2*x)*y^2 + (x^2 +
↳2*x - 3)*y + 3*x + 3]
[(3*x^2 - 4*x)*y^2 + (x^2 + 3*x - 2)*y + 2*x + 2 (x^2 - 5*x + 4)*y^
↳2 + (5*x - 8)*y + 4]
sage: td1 = t.down(g, 1) ; td1 # lowering the second index
Tensor field of type (1,1) on the 2-dimensional differentiable
manifold M
sage: td1 == g['_ac']*t['^bc'] # the same operation in index notation
True
sage: td1[:]
[ 2*x*y + x + 1 (x - 2)*y + 2]
[4*x*y + 3*x + 3 (3*x - 4)*y + 4]
sage: tdd1 = td1.down(g) ; tdd1 # the two indices have been lowered, starting
↳from the second one
Tensor field of type (0,2) on the 2-dimensional differentiable
manifold M
sage: tdd1 == g['_ac']*td1['^c_b'] # the same operation in index notation
True
sage: tdd1[:]
[ 4*x^2*y^2 + x^2 + 5*(x^2 + x)*y + 2*x + 1 (3*x^2 - 4*x)*y^2 + (x^2 +
↳3*x - 2)*y + 2*x + 2]
[2*(x^2 - 2*x)*y^2 + (x^2 + 2*x - 3)*y + 3*x + 3 (x^2 - 5*x + 4)*y^
↳2 + (5*x - 8)*y + 4]
sage: tdd1 == tdd0 # the order of index lowering is important
False
sage: tdd = t.down(g) ; tdd # both indices are lowered, starting from the
↳last one
Tensor field of type (0,2) on the 2-dimensional differentiable
manifold M
sage: tdd[:]
[ 4*x^2*y^2 + x^2 + 5*(x^2 + x)*y + 2*x + 1 (3*x^2 - 4*x)*y^2 + (x^2 +
↳3*x - 2)*y + 2*x + 2]
[2*(x^2 - 2*x)*y^2 + (x^2 + 2*x - 3)*y + 3*x + 3 (x^2 - 5*x + 4)*y^
↳2 + (5*x - 8)*y + 4]
sage: tdd0 == tdd # to get tdd0, indices have been lowered from the first
↳one, contrary to tdd
```

```

False
sage: tdd1 == tdd # the same order for index lowering has been applied
True
sage: u0tdd = tdd.up(g, 0) ; u0tdd # the first index is raised again
Tensor field of type (1,1) on the 2-dimensional differentiable
manifold M
sage: uu0tdd = u0tdd.up(g) ; uu0tdd # the second index is then raised
Tensor field of type (2,0) on the 2-dimensional differentiable
manifold M
sage: ultdd = tdd.up(g, 1) ; ultdd # raising operation, starting from the
↪last index
Tensor field of type (1,1) on the 2-dimensional differentiable
manifold M
sage: uultdd = ultdd.up(g) ; uultdd
Tensor field of type (2,0) on the 2-dimensional differentiable
manifold M
sage: utdd = tdd.up(g) ; utdd # both indices are raised, starting from the
↪first one
Tensor field of type (2,0) on the 2-dimensional differentiable
manifold M
sage: utdd == t # should be true
True
sage: uu0tdd == t # should be true
True
sage: uultdd == t # not true, because of the order of index raising to get
↪uultdd
False

```

**lie\_der** (vector)

Lie derivative of self with respect to a vector field.

INPUT:

- vector – vector field with respect to which the Lie derivative is to be taken

OUTPUT:

- the tensor field that is the Lie derivative of the current tensor field with respect to vector

EXAMPLES:

Lie derivative of a type-(1,1) tensor field along a vector field on the 2-sphere:

```

sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U') ; V = M.open_subset('V')
sage: M.declare_union(U,V) # M is the union of U and V
sage: c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart()
sage: xy_to_uv = c_xy.transition_map(c_uv, (x+y, x-y),
....:                               intersection_name='W', restrictions1= x>0,
....:                               restrictions2= u+v>0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: e_xy = c_xy.frame(); e_uv = c_uv.frame()
sage: t = M.tensor_field(1,1, name='t')
sage: t[e_xy,:] = [[x, 1], [y, 0]]
sage: t.add_comp_by_continuation(e_uv, U.intersection(V), c_uv)
sage: w = M.vector_field(name='w')
sage: w[e_xy,:] = [-y, x]
sage: w.add_comp_by_continuation(e_uv, U.intersection(V), c_uv)
sage: lt = t.lie_derivative(w); lt
Tensor field of type (1,1) on the 2-dimensional differentiable

```

```

manifold M
sage: lt.display(e_xy)
d/dx*dx - x d/dx*dy + (-y - 1) d/dy*dy
sage: lt.display(e_uv)
-1/2*u d/du*du + (1/2*u + 1) d/du*dv + (-1/2*v + 1) d/dv*du + 1/2*v d/dv*dv

```

The result is cached:

```

sage: t.lie_derivative(w) is lt
True

```

An alias is `lie_der`:

```

sage: t.lie_der(w) is t.lie_derivative(w)
True

```

Lie derivative of a vector field:

```

sage: a = M.vector_field(name='a')
sage: a[e_xy, :] = [1-x, x-y]
sage: a.add_comp_by_continuation(e_uv, U.intersection(V), c_uv)
sage: a.lie_der(w)
Vector field on the 2-dimensional differentiable manifold M
sage: a.lie_der(w).display(e_xy)
x d/dx + (-y - 1) d/dy
sage: a.lie_der(w).display(e_uv)
(v - 1) d/du + (u + 1) d/dv

```

The Lie derivative is antisymmetric:

```

sage: a.lie_der(w) == - w.lie_der(a)
True

```

and it coincides with the commutator of the two vector fields:

```

sage: f = M.scalar_field({c_xy: 3*x-1, c_uv: 3/2*(u+v)-1})
sage: a.lie_der(w)(f) == w(a(f)) - a(w(f)) # long time
True

```

### `lie_derivative` (*vector*)

Lie derivative of `self` with respect to a vector field.

INPUT:

- `vector` – vector field with respect to which the Lie derivative is to be taken

OUTPUT:

- the tensor field that is the Lie derivative of the current tensor field with respect to `vector`

EXAMPLES:

Lie derivative of a type-(1, 1) tensor field along a vector field on the 2-sphere:

```

sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U') ; V = M.open_subset('V')
sage: M.declare_union(U,V) # M is the union of U and V
sage: c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart()
sage: xy_to_uv = c_xy.transition_map(c_uv, (x+y, x-y),
....:                               intersection_name='W', restrictions1= x>0,

```

```

....:                restrictions2= u+v>0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: e_xy = c_xy.frame(); e_uv = c_uv.frame()
sage: t = M.tensor_field(1,1, name='t')
sage: t[e_xy,:] = [[x, 1], [y, 0]]
sage: t.add_comp_by_continuation(e_uv, U.intersection(V), c_uv)
sage: w = M.vector_field(name='w')
sage: w[e_xy,:] = [-y, x]
sage: w.add_comp_by_continuation(e_uv, U.intersection(V), c_uv)
sage: lt = t.lie_derivative(w); lt
Tensor field of type (1,1) on the 2-dimensional differentiable
manifold M
sage: lt.display(e_xy)
d/dx*dx - x d/dx*dy + (-y - 1) d/dy*dy
sage: lt.display(e_uv)
-1/2*u d/du*du + (1/2*u + 1) d/du*dv + (-1/2*v + 1) d/dv*du + 1/2*v d/dv*dv

```

The result is cached:

```

sage: t.lie_derivative(w) is lt
True

```

An alias is `lie_der`:

```

sage: t.lie_der(w) is t.lie_derivative(w)
True

```

Lie derivative of a vector field:

```

sage: a = M.vector_field(name='a')
sage: a[e_xy,:] = [1-x, x-y]
sage: a.add_comp_by_continuation(e_uv, U.intersection(V), c_uv)
sage: a.lie_der(w)
Vector field on the 2-dimensional differentiable manifold M
sage: a.lie_der(w).display(e_xy)
x d/dx + (-y - 1) d/dy
sage: a.lie_der(w).display(e_uv)
(v - 1) d/du + (u + 1) d/dv

```

The Lie derivative is antisymmetric:

```

sage: a.lie_der(w) == - w.lie_der(a)
True

```

and it coincides with the commutator of the two vector fields:

```

sage: f = M.scalar_field({c_xy: 3*x-1, c_uv: 3/2*(u+v)-1})
sage: a.lie_der(w)(f) == w(a(f)) - a(w(f)) # long time
True

```

**restrict** (*subdomain*, *dest\_map=None*)

Return the restriction of `self` to some subdomain.

If the restriction has not been defined yet, it is constructed here.

INPUT:

- *subdomain* – *DifferentiableManifold*; open subset  $U$  of the tensor field domain  $S$

- `dest_map` – *DiffMap* (default: None); destination map  $\Psi : U \rightarrow V$ , where  $V$  is an open subset of the manifold  $M$  where the tensor field takes its values; if None, the restriction of  $\Phi$  to  $U$  is used,  $\Phi$  being the differentiable map  $S \rightarrow M$  associated with the tensor field

OUTPUT:

- *TensorField* representing the restriction

EXAMPLES:

Restrictions of a vector field on the 2-sphere:

```
sage: M = Manifold(2, 'S^2', start_index=1)
sage: U = M.open_subset('U') # the complement of the North pole
sage: stereoN.<x,y> = U.chart() # stereographic coordinates from the North
↳pole
sage: eN = stereoN.frame() # the associated vector frame
sage: V = M.open_subset('V') # the complement of the South pole
sage: stereoS.<u,v> = V.chart() # stereographic coordinates from the South
↳pole
sage: eS = stereoS.frame() # the associated vector frame
sage: transf = stereoN.transition_map(stereoS, (x/(x^2+y^2), y/(x^2+y^2)),
....:                               intersection_name='W', restrictions1= x^2+y^2!=0,
....:                               restrictions2= u^2+v^2!=0)
sage: inv = transf.inverse() # transformation from stereoS to stereoN
sage: W = U.intersection(V) # the complement of the North and South poles
sage: stereoN_W = W.atlas()[0] # restriction of stereographic coord. from
↳North pole to W
sage: stereoS_W = W.atlas()[1] # restriction of stereographic coord. from
↳South pole to W
sage: eN_W = stereoN_W.frame() ; eS_W = stereoS_W.frame()
sage: v = M.vector_field('v')
sage: v.set_comp(eN)[1] = 1 # given the default settings, this can be
↳abridged to v[1] = 1
sage: v.display()
v = d/dx
sage: vU = v.restrict(U) ; vU
Vector field v on the Open subset U of the 2-dimensional
differentiable manifold S^2
sage: vU.display()
v = d/dx
sage: vU == eN[1]
True
sage: vW = v.restrict(W) ; vW
Vector field v on the Open subset W of the 2-dimensional
differentiable manifold S^2
sage: vW.display()
v = d/dx
sage: vW.display(eS_W, stereoS_W)
v = (-u^2 + v^2) d/du - 2*u*v d/dv
sage: vW == eN_W[1]
True
```

At this stage, defining the restriction of  $v$  to the open subset  $V$  fully specifies  $v$ :

```
sage: v.restrict(V)[1] = vW[eS_W, 1, stereoS_W].expr() # note that eS is the
↳default frame on V
sage: v.restrict(V)[2] = vW[eS_W, 2, stereoS_W].expr()
sage: v.display(eS, stereoS)
v = (-u^2 + v^2) d/du - 2*u*v d/dv
```

```
sage: v.restrict(U).display()
v = d/dx
sage: v.restrict(V).display()
v = (-u^2 + v^2) d/du - 2*u*v d/dv
```

The restriction of the vector field to its own domain is of course itself:

```
sage: v.restrict(M) is v
True
sage: vU.restrict(U) is vU
True
```

### `set_comp` (*basis=None*)

Return the components of `self` in a given vector frame for assignment.

The components with respect to other frames having the same domain as the provided vector frame are deleted, in order to avoid any inconsistency. To keep them, use the method `add_comp()` instead.

INPUT:

- `basis` – (default: `None`) vector frame in which the components are defined; if none is provided, the components are assumed to refer to the tensor field domain’s default frame

OUTPUT:

- components in the given frame, as a `Components`; if such components did not exist previously, they are created

EXAMPLES:

```
sage: M = Manifold(2, 'M') # the 2-dimensional sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: e_uv = c_uv.frame()
sage: t = M.tensor_field(1, 2, name='t')
sage: t.set_comp(e_uv)
3-indices components w.r.t. Coordinate frame (V, (d/du,d/dv))
sage: t.set_comp(e_uv)[1,0,1] = u*v
sage: t.display(e_uv)
t = (u + v) d/dv*du*dv
```

Setting the components in a new frame (`e`):

```
sage: e = V.vector_frame('e')
sage: t.set_comp(e)
3-indices components w.r.t. Vector frame (V, (e_0,e_1))
sage: t.set_comp(e)[0,1,1] = u*v
sage: t.display(e)
t = u*v e_0*e^1*e^1
```

Since the frames `e` and `e_uv` are defined on the same domain, the components w.r.t. `e_uv` have been erased:

```
sage: t.display(c_uv.frame())
Traceback (most recent call last):
...
```

```
ValueError: no basis could be found for computing the components
in the Coordinate frame (V, (d/du,d/dv))
```

**set\_name** (*name=None, latex\_name=None*)

Set (or change) the text name and LaTeX name of self.

INPUT:

- *name* – string (default: None); name given to the tensor field
- *latex\_name* – string (default: None); LaTeX symbol to denote the tensor field; if None while name is provided, the LaTeX symbol is set to name

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: t = M.tensor_field(1, 3); t
Tensor field of type (1,3) on the 2-dimensional differentiable
manifold M
sage: t.set_name(name='t')
sage: t
Tensor field t of type (1,3) on the 2-dimensional differentiable
manifold M
sage: latex(t)
t
sage: t.set_name(latex_name=r'\tau')
sage: latex(t)
\tau
sage: t.set_name(name='a')
sage: t
Tensor field a of type (1,3) on the 2-dimensional differentiable
manifold M
sage: latex(t)
a
```

**set\_restriction** (*rst*)

Define a restriction of self to some subdomain.

INPUT:

- *rst* – *TensorField* of the same type and symmetries as the current tensor field self, defined on a subdomain of the domain of self

EXAMPLES:

```
sage: M = Manifold(2, 'M') # the 2-dimensional sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: t = M.tensor_field(1, 2, name='t')
sage: s = U.tensor_field(1, 2)
sage: s[0,0,1] = x+y
sage: t.set_restriction(s)
sage: t.display(c_xy.frame())
t = (x + y) d/dx*d/dy
sage: t.restrict(U) == s
True
```

**symmetries()**

Print the list of symmetries and antisymmetries.

EXAMPLES:

```
sage: M = Manifold(2, 'S^2')
sage: t = M.tensor_field(1,2)
sage: t.symmetries()
no symmetry; no antisymmetry
sage: t = M.tensor_field(1,2, sym=(1,2))
sage: t.symmetries()
symmetry: (1, 2); no antisymmetry
sage: t = M.tensor_field(2,2, sym=(0,1), antisym=(2,3))
sage: t.symmetries()
symmetry: (0, 1); antisymmetry: (2, 3)
sage: t = M.tensor_field(2,2, antisym=[[0,1], (2,3)])
sage: t.symmetries()
no symmetry; antisymmetries: [(0, 1), (2, 3)]
```

**symmetrize(\*pos)**

Symmetrization over some arguments.

INPUT:

- *pos* – (default: None) list of argument positions involved in the symmetrization (with the convention `position=0` for the first argument); if None, the symmetrization is performed over all the arguments

OUTPUT:

- the symmetrized tensor field (instance of *TensorField*)

EXAMPLES:

Symmetrization of a type-(0,2) tensor field on a 2-dimensional non-parallelizable manifold:

```
sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U') ; V = M.open_subset('V')
sage: M.declare_union(U,V) # M is the union of U and V
sage: c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart()
sage: transf = c_xy.transition_map(c_uv, (x+y, x-y), intersection_name='W',
....:                               restrictions1= x>0, restrictions2= u+v>0)
sage: inv = transf.inverse()
sage: W = U.intersection(V)
sage: eU = c_xy.frame() ; eV = c_uv.frame()
sage: a = M.tensor_field(0,2, name='a')
sage: a[eU,:] = [[1,x], [2,y]]
sage: a.add_comp_by_continuation(eV, W, chart=c_uv)
sage: a[eV,:]
[ 1/4*u + 3/4 -1/4*u + 3/4]
[ 1/4*v - 1/4 -1/4*v - 1/4]
sage: s = a.symmetrize() ; s
Field of symmetric bilinear forms on the 2-dimensional
differentiable manifold M
sage: s[eU,:]
[          1 1/2*x + 1]
[1/2*x + 1          y]
sage: s[eV,:]
[          1/4*u + 3/4 -1/8*u + 1/8*v + 1/4]
[-1/8*u + 1/8*v + 1/4          -1/4*v - 1/4]
```

```
sage: s == a.symmetrize(0,1) # explicit positions
True
```

**See also:**

For more details and examples, see `sage.tensor.modules.free_module_tensor.FreeModuleTensor.symmetrize()`.

**tensor\_rank()**

Return the tensor rank of `self`.

**OUTPUT:**

- integer  $k + l$ , where  $k$  is the contravariant rank and  $l$  is the covariant rank

**EXAMPLES:**

```
sage: M = Manifold(2, 'S^2')
sage: t = M.tensor_field(1,2)
sage: t.tensor_rank()
3
sage: v = M.vector_field()
sage: v.tensor_rank()
1
```

**tensor\_type()**

Return the tensor type of `self`.

**OUTPUT:**

- pair  $(k, l)$ , where  $k$  is the contravariant rank and  $l$  is the covariant rank

**EXAMPLES:**

```
sage: M = Manifold(2, 'S^2')
sage: t = M.tensor_field(1,2)
sage: t.tensor_type()
(1, 2)
sage: v = M.vector_field()
sage: v.tensor_type()
(1, 0)
```

**trace(pos1=0, pos2=1)**

Trace (contraction) on two slots of the tensor field.

**INPUT:**

- `pos1` – (default: 0) position of the first index for the contraction, with the convention `pos1=0` for the first slot
- `pos2` – (default: 1) position of the second index for the contraction, with the same convention as for `pos1`. The variance type of `pos2` must be opposite to that of `pos1`

**OUTPUT:**

- tensor field resulting from the  $(pos1, pos2)$  contraction

**EXAMPLES:**

Trace of a type- $(1, 1)$  tensor field on a 2-dimensional non-parallelizable manifold:

```

sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U') ; V = M.open_subset('V')
sage: M.declare_union(U,V) # M is the union of U and V
sage: c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart()
sage: xy_to_uv = c_xy.transition_map(c_uv, (x+y, x-y),
....:                               intersection_name='W', restrictions1= x>0,
....:                               restrictions2= u+v>0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: e_xy = c_xy.frame(); e_uv = c_uv.frame()
sage: W = U.intersection(V)
sage: a = M.tensor_field(1,1, name='a')
sage: a[e_xy,:] = [[1,x], [2,y]]
sage: a.add_comp_by_continuation(e_uv, W, chart=c_uv)
sage: s = a.trace() ; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M --> R
on U: (x, y) |--> y + 1
on V: (u, v) |--> 1/2*u - 1/2*v + 1
sage: s == a.trace(0,1) # explicit mention of the positions
True

```

Instead of the explicit call to the method `trace()`, one may use the index notation with Einstein convention (summation over repeated indices); it suffices to pass the indices as a string inside square brackets:

```

sage: a['^i_i']
Scalar field on the 2-dimensional differentiable manifold M
sage: a['^i_i'] == s
True

```

Any letter can be used to denote the repeated index:

```

sage: a['^b_b'] == s
True

```

Trace of a type-(1,2) tensor field:

```

sage: b = M.tensor_field(1,2, name='b') ; b
Tensor field b of type (1,2) on the 2-dimensional differentiable
manifold M
sage: b[e_xy,:] = [[[0,x+y], [y,0]], [[0,2], [3*x,-2]]]
sage: b.add_comp_by_continuation(e_uv, W, chart=c_uv) # long time
sage: s = b.trace(0,1) ; s # contraction on first and second slots
1-form on the 2-dimensional differentiable manifold M
sage: s.display(e_xy)
3*x dx + (x + y - 2) dy
sage: s.display(e_uv) # long time
(5/4*u + 3/4*v - 1) du + (1/4*u + 3/4*v + 1) dv

```

Use of the index notation:

```

sage: b['^k_ki']
1-form on the 2-dimensional differentiable manifold M
sage: b['^k_ki'] == s # long time
True

```

Indices not involved in the contraction may be replaced by dots:

```
sage: b['^k_k.'] == s # long time
True
```

The symbol ^ may be omitted:

```
sage: b['k_k.'] == s # long time
True
```

LaTeX notations are allowed:

```
sage: b['^{k}_{ki}'] == s # long time
True
```

Contraction on first and third slots:

```
sage: s = b.trace(0,2) ; s
1-form on the 2-dimensional differentiable manifold M
sage: s.display(e_xy)
2 dx + (y - 2) dy
sage: s.display(e_uv) # long time
(1/4*u - 1/4*v) du + (-1/4*u + 1/4*v + 2) dv
```

Use of index notation:

```
sage: b['^k_.k'] == s # long time
True
```

**up** (*metric*, *pos=None*)

Compute a metric dual of the tensor field by raising some index with a given metric.

If  $T$  is the tensor field,  $(k, l)$  its type and  $p$  the position of a covariant index (i.e.  $k \leq p < k + l$ ), this method called with `pos = p` yields the tensor field  $T^\sharp$  of type  $(k + 1, l - 1)$  whose components are

$$(T^\sharp)^{a_1 \dots a_{k+1}}_{b_1 \dots b_{l-1}} = g^{a_{k+1} i} T^{a_1 \dots a_k}_{b_1 \dots b_{p-k} i b_{p-k+1} \dots b_{l-1}}$$

$g^{ab}$  being the components of the inverse metric.

The reverse operation is `TensorField.down()`

INPUT:

- `metric` – metric  $g$ , as an instance of `PseudoRiemannianMetric`
- `pos` – (default: `None`) position of the index (with the convention `pos=0` for the first index); if `None`, the raising is performed over all the covariant indices, starting from the first one

OUTPUT:

- the tensor field  $T^\sharp$  resulting from the index raising operation

EXAMPLES:

Raising the index of a 1-form results in a vector field:

```
sage: M = Manifold(2, 'M', start_index=1)
sage: c_xy.<x,y> = M.chart()
sage: g = M.metric('g')
sage: g[1,1], g[1,2], g[2,2] = 1+x, x*y, 1-y
sage: w = M.one_form()
sage: w[:] = [-1, 2]
sage: v = w.up(g) ; v
```

```

Vector field on the 2-dimensional differentiable manifold M
sage: v.display()
((2*x - 1)*y + 1)/(x^2*y^2 + (x + 1)*y - x - 1) d/dx
 - (x*y + 2*x + 2)/(x^2*y^2 + (x + 1)*y - x - 1) d/dy
sage: ig = g.inverse(); ig[:]
[ (y - 1)/(x^2*y^2 + (x + 1)*y - x - 1)      x*y/(x^2*y^2 + (x + 1)*y - x - 1)
 ↪1]
[      x*y/(x^2*y^2 + (x + 1)*y - x - 1) - (x + 1)/(x^2*y^2 + (x + 1)*y - x - 1)
 ↪1]

```

Using the index notation instead of `up()`:

```

sage: v == ig['^ab']*w['_b']
True

```

The reverse operation:

```

sage: w1 = v.down(g) ; w1
1-form on the 2-dimensional differentiable manifold M
sage: w1.display()
-dx + 2 dy
sage: w1 == w
True

```

The reverse operation in index notation:

```

sage: g['_ab']*v['^b'] == w
True

```

Raising the indices of a tensor field of type (0,2):

```

sage: t = M.tensor_field(0, 2)
sage: t[:] = [[1,2], [3,4]]
sage: tu0 = t.up(g, 0) ; tu0 # raising the first index
Tensor field of type (1,1) on the 2-dimensional differentiable
manifold M
sage: tu0[:]
[ ((3*x + 1)*y - 1)/(x^2*y^2 + (x + 1)*y - x - 1) 2*((2*x + 1)*y - 1)/(x^2*y^
 ↪2 + (x + 1)*y - x - 1)]
[      (x*y - 3*x - 3)/(x^2*y^2 + (x + 1)*y - x - 1)      2*(x*y - 2*x - 2)/(x^2*y^
 ↪2 + (x + 1)*y - x - 1)]
sage: tu0 == ig['^ac']*t['_cb'] # the same operation in index notation
True
sage: tuu0 = tu0.up(g) ; tuu0 # the two indices have been raised, starting
 ↪from the first one
Tensor field of type (2,0) on the 2-dimensional differentiable
manifold M
sage: tuu0 == tu0['^a_c']*ig['^cb'] # the same operation in index notation
True
sage: tu1 = t.up(g, 1) ; tu1 # raising the second index
Tensor field of type (1,1) on the 2-dimensional differentiable
manifold M
sage: tu1 == ig['^ac']*t['_bc'] # the same operation in index notation
True
sage: tu1[:]
[ ((2*x + 1)*y - 1)/(x^2*y^2 + (x + 1)*y - x - 1) ((4*x + 3)*y - 3)/(x^2*y^2 +
 ↪(x + 1)*y - x - 1)]
[      (x*y - 2*x - 2)/(x^2*y^2 + (x + 1)*y - x - 1) (3*x*y - 4*x - 4)/(x^2*y^2 +
 ↪(x + 1)*y - x - 1)]

```

```

sage: tuu1 = tu1.up(g) ; tuu1 # the two indices have been raised, starting
↳from the second one
Tensor field of type (2,0) on the 2-dimensional differentiable
manifold M
sage: tuu1 == tu1['^a_c']*ig['^cb'] # the same operation in index notation
True
sage: tuu0 == tuu1 # the order of index raising is important
False
sage: tuu = t.up(g) ; tuu # both indices are raised, starting from the first
↳one
Tensor field of type (2,0) on the 2-dimensional differentiable
manifold M
sage: tuu0 == tuu # the same order for index raising has been applied
True
sage: tuu1 == tuu # to get tuu1, indices have been raised from the last one,
↳contrary to tuu
False
sage: d0tuu = tuu.down(g, 0) ; d0tuu # the first index is lowered again
Tensor field of type (1,1) on the 2-dimensional differentiable
manifold M
sage: dd0tuu = d0tuu.down(g) ; dd0tuu # the second index is then lowered
Tensor field of type (0,2) on the 2-dimensional differentiable
manifold M
sage: d1tuu = tuu.down(g, 1) ; d1tuu # lowering operation, starting from the
↳last index
Tensor field of type (1,1) on the 2-dimensional differentiable
manifold M
sage: dd1tuu = d1tuu.down(g) ; dd1tuu
Tensor field of type (0,2) on the 2-dimensional differentiable
manifold M
sage: ddtuu = tuu.down(g) ; ddtuu # both indices are lowered, starting from
↳the last one
Tensor field of type (0,2) on the 2-dimensional differentiable
manifold M
sage: ddtuu == t # should be true
True
sage: dd0tuu == t # not true, because of the order of index lowering to get
↳dd0tuu
False
sage: dd1tuu == t # should be true
True

```

### 2.8.3 Tensor Fields with Values on a Parallelizable Manifold

The class *TensorFieldParal* implements tensor fields along a differentiable manifolds with values on a parallelizable differentiable manifold. For non-parallelizable manifolds, see the class *TensorField*.

Various derived classes of *TensorFieldParal* are devoted to specific tensor fields:

- *VectorFieldParal* for vector fields (rank-1 contravariant tensor fields)
- *AutomorphismFieldParal* for fields of tangent-space automorphisms
- *DiffFormParal* for differential forms (fully antisymmetric covariant tensor fields)

AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2013-2015) : initial version

- Travis Scrimshaw (2016): review tweaks

## REFERENCES:

- [KN1963]
- [Lee2013]
- [ONe1983]

## EXAMPLES:

A tensor field of type  $(1, 1)$  on a 2-dimensional differentiable manifold:

```
sage: M = Manifold(2, 'M', start_index=1)
sage: c_xy.<x,y> = M.chart()
sage: t = M.tensor_field(1, 1, 'T') ; t
Tensor field T of type (1,1) on the 2-dimensional differentiable manifold M
sage: t.tensor_type()
(1, 1)
sage: t.tensor_rank()
2
```

Components with respect to the manifold's default frame are created by providing the relevant indices inside square brackets:

```
sage: t[1,1] = x^2
```

Unset components are initialized to zero:

```
sage: t[:] # list of components w.r.t. the manifold's default vector frame
[x^2  0]
[ 0  0]
```

The full set of components with respect to a given vector frame is returned by the method `comp()`:

```
sage: t.comp(c_xy.frame())
2-indices components w.r.t. Coordinate frame (M, (d/dx,d/dy))
```

If no vector frame is mentioned in the argument of `comp()`, it is assumed to be the manifold's default frame:

```
sage: M.default_frame()
Coordinate frame (M, (d/dx,d/dy))
sage: t.comp() is t.comp(c_xy.frame())
True
```

Individual components with respect to the manifold's default frame are accessed by listing their indices inside double square brackets. They are *scalar fields* on the manifold:

```
sage: t[[1,1]]
Scalar field on the 2-dimensional differentiable manifold M
sage: t[[1,1]].display()
M --> R
(x, y) |--> x^2
sage: t[[1,2]]
Scalar field zero on the 2-dimensional differentiable manifold M
sage: t[[1,2]].display()
zero: M --> R
(x, y) |--> 0
```

A direct access to the coordinate expression of some component is obtained via the single square brackets:

```
sage: t[1,1]
x^2
sage: t[1,1] is t[[1,1]].coord_function() # the coordinate function
True
sage: t[1,1] is t[[1,1]].coord_function(c_xy)
True
sage: t[1,1].expr() is t[[1,1]].expr() # the symbolic expression
True
```

Expressions in a chart different from the manifold's default one are obtained by specifying the chart as the last argument inside the single square brackets:

```
sage: c_uv.<u,v> = M.chart()
sage: xy_to_uv = c_xy.transition_map(c_uv, [x+y, x-y])
sage: uv_to_xy = xy_to_uv.inverse()
sage: t[1,1, c_uv]
1/4*u^2 + 1/2*u*v + 1/4*v^2
```

Note that  $t[1,1, c_{uv}]$  is the component of the tensor  $t$  with respect to the coordinate frame associated to the chart  $(x,y)$  expressed in terms of the coordinates  $(u,v)$ . Indeed,  $t[1,1, c_{uv}]$  is a shortcut for  $t.comp(c_{xy}.frame())[[1,1]].coord_function(c_{uv})$ :

```
sage: t[1,1, c_uv] is t.comp(c_xy.frame())[[1,1]].coord_function(c_uv)
True
```

Similarly,  $t[1,1]$  is a shortcut for  $t.comp(c_{xy}.frame())[[1,1]].coord_function(c_{xy})$ :

```
sage: t[1,1] is t.comp(c_xy.frame())[[1,1]].coord_function(c_xy)
True
sage: t[1,1] is t.comp()[[1,1]].coord_function() # since c_xy.frame() and c_xy are
↳the manifold's default values
True
```

All the components can be set at once via `[ : ]`:

```
sage: t[:] = [[1, -x], [x*y, 2]]
sage: t[:]
[ 1 -x]
[x*y 2]
```

To set the components in a vector frame different from the manifold's default one, the method `set_comp()` can be employed:

```
sage: e = M.vector_frame('e')
sage: t.set_comp(e)[1,1] = x+y
sage: t.set_comp(e)[2,1], t.set_comp(e)[2,2] = y, -3*x
```

but, as a shortcut, one may simply specify the frame as the first argument of the square brackets:

```
sage: t[e,1,1] = x+y
sage: t[e,2,1], t[e,2,2] = y, -3*x
sage: t.comp(e)
2-indices components w.r.t. Vector frame (M, (e_1,e_2))
sage: t.comp(e)[:]
[x + y 0]
[ y -3*x]
```

```
sage: t[e,:] # a shortcut of the above
[x + y  0]
[   y -3*x]
```

All the components in some frame can be set at once, via the operator [:]:

```
sage: t[e,:] = [[x+y, 0], [y, -3*x]]
sage: t[e,:] # same as above:
[x + y  0]
[   y -3*x]
```

To avoid any inconsistency between the various components, the method `set_comp()` clears the components in other frames. To keep the other components, one must use the method `add_comp()`:

```
sage: t = M.tensor_field(1, 1, 'T') # Let us restart
sage: t[:] = [[1, -x], [x*y, 2]] # by first setting the components in the frame c_xy.
↪ frame()
sage: # We now set the components in the frame e with add_comp:
sage: t.add_comp(e)[:]= [[x+y, 0], [y, -3*x]]
```

The expansion of the tensor field in a given frame is obtained via the method `display()` (the symbol `*` stands for tensor product):

```
sage: t.display() # expansion in the manifold's default frame
T = d/dx*dx - x d/dx*dy + x*y d/dy*dx + 2 d/dy*dy
sage: t.display(e)
T = (x + y) e_1*e^1 + y e_2*e^1 - 3*x e_2*e^2
```

By definition, a tensor field acts as a multilinear map on 1-forms and vector fields; in the present case, `T` being of type  $(1,1)$ , it acts on pairs (1-form, vector field):

```
sage: a = M.one_form('a')
sage: a[:] = (1, x)
sage: v = M.vector_field('V')
sage: v[:] = (y, 2)
sage: t(a,v)
Scalar field T(a,V) on the 2-dimensional differentiable manifold M
sage: t(a,v).display()
T(a,V): M --> R
(x, y) |--> x^2*y^2 + 2*x + y
(u, v) |--> 1/16*u^4 - 1/8*u^2*v^2 + 1/16*v^4 + 3/2*u + 1/2*v
sage: latex(t(a,v))
T\left(a,V\right)
```

Check by means of the component expression of `t(a, v)`:

```
sage: t(a,v).expr() - t[1,1]*a[1]*v[1] - t[1,2]*a[1]*v[2] \
.....: - t[2,1]*a[2]*v[1] - t[2,2]*a[2]*v[2]
0
```

A scalar field (rank-0 tensor field):

```
sage: f = M.scalar_field(x*y + 2, name='f') ; f
Scalar field f on the 2-dimensional differentiable manifold M
sage: f.tensor_type()
(0, 0)
```

A scalar field acts on points on the manifold:

```
sage: p = M.point((1,2))
sage: f(p)
4
```

See *DiffScalarField* for more details on scalar fields.

A vector field (rank-1 contravariant tensor field):

```
sage: v = M.vector_field('v') ; v
Vector field v on the 2-dimensional differentiable manifold M
sage: v.tensor_type()
(1, 0)
sage: v[1], v[2] = -x, y
sage: v.display()
v = -x d/dx + y d/dy
```

A field of symmetric bilinear forms:

```
sage: q = M.sym_bilin_form_field('Q') ; q
Field of symmetric bilinear forms Q on the 2-dimensional differentiable
manifold M
sage: q.tensor_type()
(0, 2)
```

The components of a symmetric bilinear form are dealt by the subclass *CompFullySym* of the class *Components*, which takes into account the symmetry between the two indices:

```
sage: q[1,1], q[1,2], q[2,2] = (0, -x, y) # no need to set the component (2,1)
sage: type(q.comp())
<class 'sage.tensor.modules.comp.CompFullySym'>
sage: q[:] # note that the component (2,1) is equal to the component (1,2)
[ 0 -x]
[-x y]
sage: q.display()
Q = -x dx*dy - x dy*dx + y dy*dy
```

More generally, tensor symmetries or antisymmetries can be specified via the keywords *sym* and *antisym*. For instance a rank-4 covariant tensor symmetric with respect to its first two arguments (no. 0 and no. 1) and antisymmetric with respect to its last two ones (no. 2 and no. 3) is declared as follows:

```
sage: t = M.tensor_field(0, 4, 'T', sym=(0,1), antisym=(2,3))
sage: t[1,2,1,2] = 3
sage: t[2,1,1,2] # check of the symmetry with respect to the first 2 indices
3
sage: t[1,2,2,1] # check of the antisymmetry with respect to the last 2 indices
-3
```

```
class sage.manifolds.differentiable.tensorfield_paral.TensorFieldParal(vector_field_module,
                                                                    ten-
                                                                    sor_type,
                                                                    name=None,
                                                                    la-
                                                                    tex_name=None,
                                                                    sym=None,
                                                                    anti-
                                                                    sym=None)
    Bases: sage.tensor.modules.free_module_tensor.FreeModuleTensor, sage.
```

`manifolds.differentiable.tensorfield.TensorField`

Tensor field along a differentiable manifold, with values on a parallelizable manifold.

An instance of this class is a tensor field along a differentiable manifold  $U$  with values on a parallelizable manifold  $M$ , via a differentiable map  $\Phi : U \rightarrow M$ . More precisely, given two non-negative integers  $k$  and  $l$  and a differentiable map

$$\Phi : U \longrightarrow M,$$

a tensor field of type  $(k, l)$  along  $U$  with values on  $M$  is a differentiable map

$$t : U \longrightarrow T^{(k,l)}M$$

(where  $T^{(k,l)}M$  is the tensor bundle of type  $(k, l)$  over  $M$ ) such that

$$t(p) \in T^{(k,l)}(T_qM)$$

for all  $p \in U$ , i.e.  $t(p)$  is a tensor of type  $(k, l)$  on the tangent space  $T_qM$  at the point  $q = \Phi(p)$ . That is to say a multilinear map

$$t(p) : \underbrace{T_q^*M \times \cdots \times T_q^*M}_k \times \underbrace{T_qM \times \cdots \times T_qM}_l \longrightarrow K,$$

where  $T_q^*M$  is the dual vector space to  $T_qM$  and  $K$  is the topological field over which the manifold  $M$  is defined. The integer  $k + l$  is called the *tensor rank*.

The standard case of a tensor field on a differentiable manifold corresponds to  $U = M$  and  $\Phi = \text{Id}_M$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $M$  ( $U$  is then an open interval of  $\mathbf{R}$ ).

---

**Note:** If  $M$  is not parallelizable, the class `TensorField` should be used instead.

---

INPUT:

- `vector_field_module` – free module  $\mathcal{X}(U, \Phi)$  of vector fields along  $U$  associated with the map  $\Phi : U \rightarrow M$  (cf. `VectorFieldFreeModule`)
- `tensor_type` – pair  $(k, l)$  with  $k$  being the contravariant rank and  $l$  the covariant rank
- `name` – (default: `None`) name given to the tensor field
- `latex_name` – (default: `None`) LaTeX symbol to denote the tensor field; if none is provided, the LaTeX symbol is set to `name`
- `sym` – (default: `None`) a symmetry or a list of symmetries among the tensor arguments: each symmetry is described by a tuple containing the positions of the involved arguments, with the convention `position=0` for the first argument; for instance:
  - `sym=(0, 1)` for a symmetry between the 1st and 2nd arguments
  - `sym=[(0, 2), (1, 3, 4)]` for a symmetry between the 1st and 3rd arguments and a symmetry between the 2nd, 4th and 5th arguments
- `antisym` – (default: `None`) antisymmetry or list of antisymmetries among the arguments, with the same convention as for `sym`

EXAMPLES:

A tensor field of type  $(2, 0)$  on a 3-dimensional parallelizable manifold:

```

sage: M = Manifold(3, 'M')
sage: c_xyz.<x,y,z> = M.chart() # makes M parallelizable
sage: t = M.tensor_field(2, 0, 'T') ; t
Tensor field T of type (2,0) on the 3-dimensional differentiable
manifold M

```

Tensor fields are considered as elements of a module over the ring  $C^k(M)$  of scalar fields on  $M$ :

```

sage: t.parent()
Free module T^(2,0)(M) of type-(2,0) tensors fields on the
3-dimensional differentiable manifold M
sage: t.parent().base_ring()
Algebra of differentiable scalar fields on the 3-dimensional
differentiable manifold M

```

The components with respect to the manifold's default frame are set or read by means of square brackets:

```

sage: e = M.vector_frame('e') ; M.set_default_frame(e)
sage: for i in M.irange():
....:     for j in M.irange():
....:         t[i,j] = (i+1)**(j+1)
....:
sage: [[ t[i,j] for j in M.irange()] for i in M.irange()]
[[1, 1, 1], [2, 4, 8], [3, 9, 27]]

```

A shortcut for the above is using [:]:

```

sage: t[:]
[ 1  1  1]
[ 2  4  8]
[ 3  9 27]

```

The components with respect to another frame are set via the method `set_comp()` and read via the method `comp()`; both return an instance of `Components`:

```

sage: f = M.vector_frame('f') # a new frame defined on M, in addition to e
sage: t.set_comp(f)[0,0] = -3
sage: t.comp(f)
2-indices components w.r.t. Vector frame (M, (f_0,f_1,f_2))
sage: t.comp(f)[0,0]
-3
sage: t.comp(f)[:] # the full list of components
[-3  0  0]
[ 0  0  0]
[ 0  0  0]

```

To avoid any inconsistency between the various components, the method `set_comp()` deletes the components in other frames. Accordingly, the components in the frame  $e$  have been deleted:

```

sage: t._components
{Vector frame (M, (f_0,f_1,f_2)): 2-indices components w.r.t. Vector
frame (M, (f_0,f_1,f_2))}

```

To keep the other components, one must use the method `add_comp()`:

```

sage: t = M.tensor_field(2, 0, 'T') # Let us restart
sage: t[0,0] = 2 # sets the components in the frame e
sage: # We now set the components in the frame f with add_comp:

```

```

sage: t.add_comp(f)[0,0] = -3
sage: # The components w.r.t. frame e have been kept:
sage: t._components # random (dictionary output)
{Vector frame (M, (e_0,e_1,e_2)): 2-indices components w.r.t. Vector frame (M, (e_
↪0,e_1,e_2)),
 Vector frame (M, (f_0,f_1,f_2)): 2-indices components w.r.t. Vector frame (M, (f_
↪0,f_1,f_2))}

```

The basic properties of a tensor field are:

```

sage: t.domain()
3-dimensional differentiable manifold M
sage: t.tensor_type()
(2, 0)

```

Symmetries and antisymmetries are declared via the keywords `sym` and `antisym`. For instance, a rank-6 covariant tensor that is symmetric with respect to its 1st and 3rd arguments and antisymmetric with respect to the 2nd, 5th and 6th arguments is set up as follows:

```

sage: a = M.tensor_field(0, 6, 'T', sym=(0,2), antisym=(1,4,5))
sage: a[0,0,1,0,1,2] = 3
sage: a[1,0,0,0,1,2] # check of the symmetry
3
sage: a[0,1,1,0,0,2], a[0,1,1,0,2,0] # check of the antisymmetry
(-3, 3)

```

Multiple symmetries or antisymmetries are allowed; they must then be declared as a list. For instance, a rank-4 covariant tensor that is antisymmetric with respect to its 1st and 2nd arguments and with respect to its 3rd and 4th argument must be declared as:

```

sage: r = M.tensor_field(0, 4, 'T', antisym=[[0,1], (2,3)])
sage: r[0,1,2,0] = 3
sage: r[1,0,2,0] # first antisymmetry
-3
sage: r[0,1,0,2] # second antisymmetry
-3
sage: r[1,0,0,2] # both antisymmetries acting
3

```

Tensor fields of the same type can be added and subtracted:

```

sage: a = M.tensor_field(2, 0)
sage: a[0,0], a[0,1], a[0,2] = (1,2,3)
sage: b = M.tensor_field(2, 0)
sage: b[0,0], b[1,1], b[2,2], b[0,2] = (4,5,6,7)
sage: s = a + 2*b ; s
Tensor field of type (2,0) on the 3-dimensional differentiable
manifold M
sage: a[:], (2*b)[:], s[:]
(
 [1 2 3] [ 8 0 14] [ 9 2 17]
 [0 0 0] [ 0 10 0] [ 0 10 0]
 [0 0 0], [ 0 0 12], [ 0 0 12]
)
sage: s = a - b ; s
Tensor field of type (2,0) on the 3-dimensional differentiable
manifold M
sage: a[:], b[:], s[:]

```

```
(
[1 2 3] [4 0 7] [-3 2 -4]
[0 0 0] [0 5 0] [ 0 -5 0]
[0 0 0], [0 0 6], [ 0 0 -6]
)
```

Symmetries are preserved by the addition whenever it is possible:

```
sage: a = M.tensor_field(2, 0, sym=(0,1))
sage: a[0,0], a[0,1], a[0,2] = (1,2,3)
sage: s = a + b
sage: a[:], b[:], s[:]
(
[1 2 3] [4 0 7] [ 5 2 10]
[2 0 0] [0 5 0] [ 2 5 0]
[3 0 0], [0 0 6], [ 3 0 6]
)
sage: a.symmetries()
symmetry: (0, 1); no antisymmetry
sage: b.symmetries()
no symmetry; no antisymmetry
sage: s.symmetries()
no symmetry; no antisymmetry
sage: # let us now make b symmetric:
sage: b = M.tensor_field(2, 0, sym=(0,1))
sage: b[0,0], b[1,1], b[2,2], b[0,2] = (4,5,6,7)
sage: s = a + b
sage: a[:], b[:], s[:]
(
[1 2 3] [4 0 7] [ 5 2 10]
[2 0 0] [0 5 0] [ 2 5 0]
[3 0 0], [7 0 6], [10 0 6]
)
sage: s.symmetries() # s is symmetric because both a and b are
symmetry: (0, 1); no antisymmetry
```

The tensor product is taken with the operator `*`:

```
sage: c = a*b ; c
Tensor field of type (4,0) on the 3-dimensional differentiable
manifold M
sage: c.symmetries() # since a and b are both symmetric, a*b has two symmetries:
symmetries: [(0, 1), (2, 3)]; no antisymmetry
```

The tensor product of two fully contravariant tensors is not symmetric in general:

```
sage: a*b == b*a
False
```

The tensor product of a fully contravariant tensor by a fully covariant one is symmetric:

```
sage: d = M.diff_form(2) # a fully covariant tensor field
sage: d[0,1], d[0,2], d[1,2] = (3, 2, 1)
sage: s = a*d ; s
Tensor field of type (2,2) on the 3-dimensional differentiable
manifold M
sage: s.symmetries()
symmetry: (0, 1); antisymmetry: (2, 3)
```

```

sage: s1 = d*a ; s1
Tensor field of type (2,2) on the 3-dimensional differentiable
manifold M
sage: s1.symmetries()
symmetry: (0, 1); antisymmetry: (2, 3)
sage: d*a == a*d
True

```

Example of tensor field associated with a non-trivial differentiable map  $\Phi$ : tensor field along a curve in  $M$ :

```

sage: R = Manifold(1, 'R') # R as a 1-dimensional manifold
sage: T.<t> = R.chart() # canonical chart on R
sage: Phi = R.diff_map(M, [cos(t), sin(t), t], name='Phi') ; Phi
Differentiable map Phi from the 1-dimensional differentiable manifold R
to the 3-dimensional differentiable manifold M
sage: h = R.tensor_field(2, 0, name='h', dest_map=Phi) ; h
Tensor field h of type (2,0) along the 1-dimensional differentiable
manifold R with values on the 3-dimensional differentiable manifold M
sage: h.parent()
Free module T^(2,0)(R,Phi) of type-(2,0) tensors fields along the
1-dimensional differentiable manifold R mapped into the 3-dimensional
differentiable manifold M
sage: h[0,0], h[0,1], h[2,0] = 1+t, t^2, sin(t)
sage: h.display()
h = (t + 1) d/dx*d/dx + t^2 d/dx*d/dy + sin(t) d/dz*d/dx

```

**add\_comp** (*basis=None*)

Return the components of the tensor field in a given vector frame for assignment.

The components with respect to other frames on the same domain are kept. To delete them them, use the method `set_comp()` instead.

INPUT:

- `basis` – (default: None) vector frame in which the components are defined; if none is provided, the components are assumed to refer to the tensor field domain’s default frame

OUTPUT:

- components in the given frame, as an instance of the class `Components`; if such components did not exist previously, they are created

EXAMPLES:

```

sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: e_xy = X.frame()
sage: t = M.tensor_field(1,1, name='t')
sage: t.add_comp(e_xy)
2-indices components w.r.t. Coordinate frame (M, (d/dx,d/dy))
sage: t.add_comp(e_xy)[1,0] = 2
sage: t.display(e_xy)
t = 2 d/dy*d/dx

```

Adding components with respect to a new frame (e):

```

sage: e = M.vector_frame('e')
sage: t.add_comp(e)
2-indices components w.r.t. Vector frame (M, (e_0,e_1))
sage: t.add_comp(e)[0,1] = x

```

```
sage: t.display(e)
t = x e_0*e^1
```

The components with respect to the frame  $e_{xy}$  are kept:

```
sage: t.display(e_xy)
t = 2 d/dy*dx
```

Adding components in a frame defined on a subdomain:

```
sage: U = M.open_subset('U', coord_def={X: x>0})
sage: f = U.vector_frame('f')
sage: t.add_comp(f)
2-indices components w.r.t. Vector frame (U, (f_0, f_1))
sage: t.add_comp(f)[0,1] = 1+y
sage: t.display(f)
t = (y + 1) f_0*f^1
```

The components previously defined are kept:

```
sage: t.display(e_xy)
t = 2 d/dy*dx
sage: t.display(e)
t = x e_0*e^1
```

#### **at** (*point*)

Value of `self` at a point of its domain.

If the current tensor field is

$$t : U \longrightarrow T^{(k,l)} M$$

associated with the differentiable map

$$\Phi : U \longrightarrow M,$$

where  $U$  and  $M$  are two manifolds (possibly  $U = M$  and  $\Phi = \text{Id}_M$ ), then for any point  $p \in U$ ,  $t(p)$  is a tensor on the tangent space to  $M$  at the point  $\Phi(p)$ .

INPUT:

- `point` – *ManifoldPoint* point  $p$  in the domain of the tensor field  $U$

OUTPUT:

- `FreeModuleTensor` representing the tensor  $t(p)$  on the tangent vector space  $T_{\Phi(p)} M$

EXAMPLES:

Vector in a tangent space of a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart()
sage: p = M.point((-2,3), name='p')
sage: v = M.vector_field('v')
sage: v[:] = [y, x^2] ; v.display()
v = y d/dx + x^2 d/dy
sage: vp = v.at(p) ; vp
Tangent vector v at Point p on the 2-dimensional differentiable
manifold M
```

```

sage: vp.parent()
Tangent space at Point p on the 2-dimensional differentiable
manifold M
sage: vp.display()
v = 3 d/dx + 4 d/dy

```

A 1-form gives birth to a linear form in the tangent space:

```

sage: w = M.one_form('w')
sage: w[:] = [-x, 1+y] ; w.display()
w = -x dx + (y + 1) dy
sage: wp = w.at(p) ; wp
Linear form w on the Tangent space at Point p on the 2-dimensional
differentiable manifold M
sage: wp.parent()
Dual of the Tangent space at Point p on the 2-dimensional
differentiable manifold M
sage: wp.display()
w = 2 dx + 4 dy

```

A tensor field of type (1, 1) yields a tensor of type (1, 1) in the tangent space:

```

sage: t = M.tensor_field(1, 1, name='t')
sage: t[0,0], t[0,1], t[1,1] = 1+x, x*y, 1-y
sage: t.display()
t = (x + 1) d/dx*dx + x*y d/dx*dy + (-y + 1) d/dy*dy
sage: tp = t.at(p) ; tp
Type-(1,1) tensor t on the Tangent space at Point p on the
2-dimensional differentiable manifold M
sage: tp.parent()
Free module of type-(1,1) tensors on the Tangent space at Point p
on the 2-dimensional differentiable manifold M
sage: tp.display()
t = -d/dx*dx - 6 d/dx*dy - 2 d/dy*dy

```

A 2-form yields an alternating form of degree 2 in the tangent space:

```

sage: a = M.diff_form(2, name='a')
sage: a[0,1] = x*y
sage: a.display()
a = x*y dx/\dy
sage: ap = a.at(p) ; ap
Alternating form a of degree 2 on the Tangent space at Point p on
the 2-dimensional differentiable manifold M
sage: ap.parent()
2nd exterior power of the dual of the Tangent space at Point p on
the 2-dimensional differentiable manifold M
sage: ap.display()
a = -6 dx/\dy

```

Example with a non trivial map  $\Phi$ :

```

sage: U = Manifold(1, 'U') # (0,2*pi) as a 1-dimensional manifold
sage: T.<t> = U.chart(r't:(0,2*pi)') # canonical chart on U
sage: Phi = U.diff_map(M, [cos(t), sin(t)], name='Phi',
.....:                  latex_name=r'\Phi')
sage: v = U.vector_field(name='v', dest_map=Phi) ; v
Vector field v along the 1-dimensional differentiable manifold U

```

```

with values on the 2-dimensional differentiable manifold M
sage: v[:] = [1+t, t^2]
sage: v.display()
v = (t + 1) d/dx + t^2 d/dy
sage: p = U((pi/6,))
sage: vp = v.at(p) ; vp
Tangent vector v at Point on the 2-dimensional differentiable
manifold M
sage: vp.parent() is M.tangent_space(Phi(p))
True
sage: vp.display()
v = (1/6*pi + 1) d/dx + 1/36*pi^2 d/dy

```

**comp** (*basis=None, from\_basis=None*)

Return the components in a given vector frame.

If the components are not known already, they are computed by the tensor change-of-basis formula from components in another vector frame.

INPUT:

- *basis* – (default: None) vector frame in which the components are required; if none is provided, the components are assumed to refer to the tensor field domain’s default frame
- *from\_basis* – (default: None) vector frame from which the required components are computed, via the tensor change-of-basis formula, if they are not known already in the *basis*

OUTPUT:

- components in the vector frame *basis*, as an instance of the class `Components`

EXAMPLES:

```

sage: M = Manifold(2, 'M', start_index=1)
sage: X.<x,y> = M.chart()
sage: t = M.tensor_field(1,2, name='t')
sage: t[1,2,1] = x*y
sage: t.comp(X.frame())
3-indices components w.r.t. Coordinate frame (M, (d/dx,d/dy))
sage: t.comp() # the default frame is X.frame()
3-indices components w.r.t. Coordinate frame (M, (d/dx,d/dy))
sage: t.comp()[:]:
[[[0, 0], [x*y, 0]], [[0, 0], [0, 0]]]
sage: e = M.vector_frame('e')
sage: t[e, 2,1,1] = x-3
sage: t.comp(e)
3-indices components w.r.t. Vector frame (M, (e_1,e_2))
sage: t.comp(e)[:]:
[[[0, 0], [0, 0]], [[x - 3, 0], [0, 0]]]

```

**contract** (*\*args*)

Contraction with another tensor field, on one or more indices.

INPUT:

- *pos1* – positions of the indices in *self* involved in the contraction; *pos1* must be a sequence of integers, with 0 standing for the first index position, 1 for the second one, etc. If *pos1* is not provided, a single contraction on the last index position of *self* is assumed
- *other* – the tensor field to contract with

- `pos2` – positions of the indices in `other` involved in the contraction, with the same conventions as for `pos1`. If `pos2` is not provided, a single contraction on the first index position of `other` is assumed

OUTPUT:

- tensor field resulting from the contraction at the positions `pos1` and `pos2` of `self` with `other`

EXAMPLES:

Contraction of a tensor field of type (2,0) with a tensor field of type (1,1):

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: a = M.tensor_field(2,0, name='a')
sage: a[:] = [[1+x, 2], [y, -x^2]]
sage: b = M.tensor_field(1,1, name='b')
sage: b[:] = [[-y, 1], [x, x+y]]
sage: s = a.contract(0, b, 1); s
Tensor field of type (2,0) on the 2-dimensional differentiable manifold M
sage: s.display()
-x*y d/dx*d/dx + (x^2 + x*y + y^2 + x) d/dx*d/dy
+ (-x^2 - 2*y) d/dy*d/dx + (-x^3 - x^2*y + 2*x) d/dy*d/dy
```

Check:

```
sage: all([s[ind] == sum(a[k, ind[0]]*b[ind[1], k] for k in [0..1])
....:      for ind in M.index_generator(2)])
True
```

The same contraction with repeated index notation:

```
sage: s == a['^ki']*b['^j_k']
True
```

Contraction on the second index of a:

```
sage: s = a.contract(1, b, 1); s
Tensor field of type (2,0) on the 2-dimensional differentiable manifold M
sage: s.display()
-(x + 1)*y + 2) d/dx*d/dx + (x^2 + 3*x + 2*y) d/dx*d/dy
+ (-x^2 - y^2) d/dy*d/dx + (-x^3 - (x^2 - x)*y) d/dy*d/dy
```

Check:

```
sage: all([s[ind] == sum(a[ind[0], k]*b[ind[1], k] for k in [0..1])
....:      for ind in M.index_generator(2)])
True
```

The same contraction with repeated index notation:

```
sage: s == a['^ik']*b['^j_k']
True
```

See also:

*sage.manifolds.differentiable.tensorfield.TensorField.contract()* for more examples.

**display\_comp** (*frame=None, chart=None, coordinate\_labels=True, only\_nonzero=True, only\_nonredundant=False*)

Display the tensor components with respect to a given frame, one per line.

The output is either text-formatted (console mode) or LaTeX-formatted (notebook mode).

INPUT:

- *frame* – (default: *None*) vector frame with respect to which the tensor field components are defined; if *None*, then
  - if *chart* is not *None*, the coordinate frame associated to *chart* is used
  - otherwise, the default basis of the vector field module on which the tensor field is defined is used
- *chart* – (default: *None*) chart specifying the coordinate expression of the components; if *None*, the default chart of the tensor field domain is used
- *coordinate\_labels* – (default: *True*) boolean; if *True*, coordinate symbols are used by default (instead of integers) as index labels whenever *frame* is a coordinate frame
- *only\_nonzero* – (default: *True*) boolean; if *True*, only nonzero components are displayed
- *only\_nonredundant* – (default: *False*) boolean; if *True*, only nonredundant components are displayed in case of symmetries

EXAMPLES:

Display of the components of a type-(2, 1) tensor field on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: t = M.tensor_field(2, 1, name='t', sym=(0,1))
sage: t[0,0,0], t[0,1,0], t[1,1,1] = x+y, x*y, -3
sage: t.display_comp()
t^xx_x = x + y
t^xy_x = x*y
t^yx_x = x*y
t^yy_y = -3
```

By default, only the non-vanishing components are displayed; to see all the components, the argument *only\_nonzero* must be set to *False*:

```
sage: t.display_comp(only_nonzero=False)
t^xx_x = x + y
t^xx_y = 0
t^xy_x = x*y
t^xy_y = 0
t^yx_x = x*y
t^yx_y = 0
t^yy_x = 0
t^yy_y = -3
```

*t* being symmetric with respect to its first two indices, one may ask to skip the components that can be deduced by symmetry:

```
sage: t.display_comp(only_nonredundant=True)
t^xx_x = x + y
t^xy_x = x*y
t^yy_y = -3
```

Instead of coordinate labels, one may ask for integers:

```
sage: t.display_comp(coordinate_labels=False)
t^00_0 = x + y
t^01_0 = x*y
t^10_0 = x*y
t^11_1 = -3
```

Display in a frame different from the default one (note that since  $f$  is not a coordinate frame, integer are used to label the indices):

```
sage: a = M.automorphism_field()
sage: a[:] = [[1+y^2, 0], [0, 2+x^2]]
sage: f = X.frame().new_frame(a, 'f')
sage: t.display_comp(frame=f)
t^00_0 = (x + y)/(y^2 + 1)
t^01_0 = x*y/(x^2 + 2)
t^10_0 = x*y/(x^2 + 2)
t^11_1 = -3/(x^2 + 2)
```

Display with respect to a chart different from the default one:

```
sage: Y.<u,v> = M.chart()
sage: X_to_Y = X.transition_map(Y, [x+y, x-y])
sage: Y_to_X = X_to_Y.inverse()
sage: t.display_comp(chart=Y)
t^uu_u = 1/4*u^2 - 1/4*v^2 + 1/2*u - 3/2
t^uu_v = 1/4*u^2 - 1/4*v^2 + 1/2*u + 3/2
t^uv_u = 1/2*u + 3/2
t^uv_v = 1/2*u - 3/2
t^vu_u = 1/2*u + 3/2
t^vu_v = 1/2*u - 3/2
t^vv_u = -1/4*u^2 + 1/4*v^2 + 1/2*u - 3/2
t^vv_v = -1/4*u^2 + 1/4*v^2 + 1/2*u + 3/2
```

Note that the frame defining the components is the coordinate frame associated with chart  $Y$ , i.e. we have:

```
sage: str(t.display_comp(chart=Y)) == str(t.display_comp(frame=Y.frame(),
↳chart=Y))
True
```

Display of the components with respect to a specific frame, expressed in terms of a specific chart:

```
sage: t.display_comp(frame=f, chart=Y)
t^00_0 = 4*u/(u^2 - 2*u*v + v^2 + 4)
t^01_0 = (u^2 - v^2)/(u^2 + 2*u*v + v^2 + 8)
t^10_0 = (u^2 - v^2)/(u^2 + 2*u*v + v^2 + 8)
t^11_1 = -12/(u^2 + 2*u*v + v^2 + 8)
```

### **lie\_der** (vector)

Compute the Lie derivative with respect to a vector field.

INPUT:

- vector – vector field with respect to which the Lie derivative is to be taken

OUTPUT:

- the tensor field that is the Lie derivative of `self` with respect to `vector`

EXAMPLES:

Lie derivative of a vector:

```
sage: M = Manifold(2, 'M', start_index=1)
sage: c_xy.<x,y> = M.chart()
sage: v = M.vector_field('v')
sage: v[:] = (-y, x)
sage: w = M.vector_field()
sage: w[:] = (2*x+y, x*y)
sage: w.lie_derivative(v)
Vector field on the 2-dimensional differentiable manifold M
sage: w.lie_derivative(v).display()
((x - 2)*y + x) d/dx + (x^2 - y^2 - 2*x - y) d/dy
```

The result is cached:

```
sage: w.lie_derivative(v) is w.lie_derivative(v)
True
```

An alias is lie\_der:

```
sage: w.lie_der(v) is w.lie_derivative(v)
True
```

The Lie derivative is antisymmetric:

```
sage: w.lie_der(v) == -v.lie_der(w)
True
```

For vectors, it coincides with the commutator:

```
sage: f = M.scalar_field(x^3 + x*y^2)
sage: w.lie_der(v)(f).display()
M --> R
(x, y) |--> -(x + 2)*y^3 + 3*x^3 - x*y^2 + 5*(x^3 - 2*x^2)*y
sage: w.lie_der(v)(f) == v(w(f)) - w(v(f)) # rhs = commutator [v,w] acting_
↪ on f
True
```

Lie derivative of a 1-form:

```
sage: om = M.one_form()
sage: om[:] = (y^2*sin(x), x^3*cos(y))
sage: om.lie_der(v)
1-form on the 2-dimensional differentiable manifold M
sage: om.lie_der(v).display()
(-y^3*cos(x) + x^3*cos(y) + 2*x*y*sin(x)) dx
+ (-x^4*sin(y) - 3*x^2*y*cos(y) - y^2*sin(x)) dy
```

Parallel computation:

```
sage: Parallelism().set('tensor', nproc=2)
sage: Parallelism().get('tensor')
2
sage: om.lie_der(v)
1-form on the 2-dimensional differentiable manifold M
sage: om.lie_der(v).display()
(-y^3*cos(x) + x^3*cos(y) + 2*x*y*sin(x)) dx
+ (-x^4*sin(y) - 3*x^2*y*cos(y) - y^2*sin(x)) dy
```

```
sage: Parallelism().set('tensor', nproc=1) # switch off parallelization
```

Check of Cartan identity:

```
sage: om.lie_der(v) == (v.contract(0, om.exterior_derivative(), 0)
....:                  + om(v).exterior_derivative())
True
```

### lie\_derivative (vector)

Compute the Lie derivative with respect to a vector field.

INPUT:

- vector – vector field with respect to which the Lie derivative is to be taken

OUTPUT:

- the tensor field that is the Lie derivative of self with respect to vector

EXAMPLES:

Lie derivative of a vector:

```
sage: M = Manifold(2, 'M', start_index=1)
sage: c_xy.<x,y> = M.chart()
sage: v = M.vector_field('v')
sage: v[:] = (-y, x)
sage: w = M.vector_field()
sage: w[:] = (2*x+y, x*y)
sage: w.lie_derivative(v)
Vector field on the 2-dimensional differentiable manifold M
sage: w.lie_derivative(v).display()
((x - 2)*y + x) d/dx + (x^2 - y^2 - 2*x - y) d/dy
```

The result is cached:

```
sage: w.lie_derivative(v) is w.lie_derivative(v)
True
```

An alias is lie\_der:

```
sage: w.lie_der(v) is w.lie_derivative(v)
True
```

The Lie derivative is antisymmetric:

```
sage: w.lie_der(v) == -v.lie_der(w)
True
```

For vectors, it coincides with the commutator:

```
sage: f = M.scalar_field(x^3 + x*y^2)
sage: w.lie_der(v)(f).display()
M --> R
(x, y) |--> -(x + 2)*y^3 + 3*x^3 - x*y^2 + 5*(x^3 - 2*x^2)*y
sage: w.lie_der(v)(f) == v(w(f)) - w(v(f)) # rhs = commutator [v,w] acting_
-> on f
True
```

Lie derivative of a 1-form:

```
sage: om = M.one_form()
sage: om[:] = (y^2*sin(x), x^3*cos(y))
sage: om.lie_der(v)
1-form on the 2-dimensional differentiable manifold M
sage: om.lie_der(v).display()
(-y^3*cos(x) + x^3*cos(y) + 2*x*y*sin(x)) dx
+ (-x^4*sin(y) - 3*x^2*y*cos(y) - y^2*sin(x)) dy
```

Parallel computation:

```
sage: Parallelism().set('tensor', nproc=2)
sage: Parallelism().get('tensor')
2
sage: om.lie_der(v)
1-form on the 2-dimensional differentiable manifold M
sage: om.lie_der(v).display()
(-y^3*cos(x) + x^3*cos(y) + 2*x*y*sin(x)) dx
+ (-x^4*sin(y) - 3*x^2*y*cos(y) - y^2*sin(x)) dy

sage: Parallelism().set('tensor', nproc=1) # switch off parallelization
```

Check of Cartan identity:

```
sage: om.lie_der(v) == (v.contract(0, om.exterior_derivative(), 0)
....:                  + om(v).exterior_derivative())
True
```

**restrict** (*subdomain*, *dest\_map=None*)

Return the restriction of *self* to some subdomain.

If the restriction has not been defined yet, it is constructed here.

INPUT:

- *subdomain* – *DifferentiableManifold*; open subset  $U$  of the tensor field domain  $S$
- *dest\_map* – *DiffMap* (default: `None`); destination map  $\Psi : U \rightarrow V$ , where  $V$  is an open subset of the manifold  $M$  where the tensor field takes its values; if `None`, the restriction of  $\Phi$  to  $U$  is used,  $\Phi$  being the differentiable map  $S \rightarrow M$  associated with the tensor field

OUTPUT:

- instance of *TensorFieldParal* representing the restriction

EXAMPLES:

Restriction of a vector field defined on  $\mathbf{R}^2$  to a disk:

```
sage: M = Manifold(2, 'R^2')
sage: c_cart.<x,y> = M.chart() # Cartesian coordinates on R^2
sage: v = M.vector_field('v')
sage: v[:] = [x+y, -1+x^2]
sage: D = M.open_subset('D') # the unit open disc
sage: c_cart_D = c_cart.restrict(D, x^2+y^2<1)
sage: v_D = v.restrict(D) ; v_D
Vector field v on the Open subset D of the 2-dimensional
differentiable manifold R^2
sage: v_D.display()
v = (x + y) d/dx + (x^2 - 1) d/dy
```

The symbolic expressions of the components with respect to Cartesian coordinates are equal:

```
sage: bool( v_D[1].expr() == v[1].expr() )
True
```

but neither the chart functions representing the components (they are defined on different charts):

```
sage: v_D[1] == v[1]
False
```

nor the scalar fields representing the components (they are defined on different open subsets):

```
sage: v_D[[1]] == v[[1]]
False
```

The restriction of the vector field to its own domain is of course itself:

```
sage: v.restrict(M) is v
True
```

### `set_comp` (*basis=None*)

Return the components of the tensor field in a given vector frame for assignment.

The components with respect to other frames on the same domain are deleted, in order to avoid any inconsistency. To keep them, use the method `add_comp()` instead.

INPUT:

- `basis` – (default: `None`) vector frame in which the components are defined; if none is provided, the components are assumed to refer to the tensor field domain's default frame

OUTPUT:

- components in the given frame, as an instance of the class `Components`; if such components did not exist previously, they are created

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: e_xy = X.frame()
sage: t = M.tensor_field(1,1, name='t')
sage: t.set_comp(e_xy)
2-indices components w.r.t. Coordinate frame (M, (d/dx,d/dy))
sage: t.set_comp(e_xy)[1,0] = 2
sage: t.display(e_xy)
t = 2 d/dy*dx
```

Setting components in a new frame (`e`):

```
sage: e = M.vector_frame('e')
sage: t.set_comp(e)
2-indices components w.r.t. Vector frame (M, (e_0,e_1))
sage: t.set_comp(e)[0,1] = x
sage: t.display(e)
t = x e_0*e^1
```

The components with respect to the frame `e_xy` have been erased:

```
sage: t.display(e_xy)
Traceback (most recent call last):
...
ValueError: no basis could be found for computing the components
in the Coordinate frame (M, (d/dx,d/dy))
```

Setting components in a frame defined on a subdomain deletes previously defined components as well:

```
sage: U = M.open_subset('U', coord_def={X: x>0})
sage: f = U.vector_frame('f')
sage: t.set_comp(f)
2-indices components w.r.t. Vector frame (U, (f_0,f_1))
sage: t.set_comp(f)[0,1] = 1+y
sage: t.display(f)
t = (y + 1) f_0*f^1
sage: t.display(e)
Traceback (most recent call last):
...
ValueError: no basis could be found for computing the components
in the Vector frame (M, (e_0,e_1))
```

## 2.9 Differential Forms

### 2.9.1 Differential Form Modules

The set  $\Lambda^p(U, \Phi)$  of  $p$ -forms along a differentiable manifold  $U$  with values on a differentiable manifold  $M$  via a differentiable map  $\Phi : U \rightarrow M$  (possibly  $U = M$  and  $\Phi = \text{Id}_M$ ) is a module over the algebra  $C^k(U)$  of differentiable scalar fields on  $U$ . It is a free module if and only if  $M$  is parallelizable. Accordingly, two classes implement  $\Lambda^p(U, \Phi)$ :

- *DiffFormModule* for differential forms with values on a generic (in practice, not parallelizable) differentiable manifold  $M$
- *DiffFormFreeModule* for differential forms with values on a parallelizable manifold  $M$

AUTHORS:

- Eric Gourgoulhon (2015): initial version
- Travis Scrimshaw (2016): review tweaks

REFERENCES:

- [KN1963]
- [Lee2013]

**class** sage.manifolds.differentiable.diff\_form\_module.**DiffFormFreeModule** (*vector\_field\_module*,  
*de-  
gree*)

Bases: sage.tensor.modules.ext\_pow\_free\_module.ExtPowerFreeModule

Free module of differential forms of a given degree  $p$  ( $p$ -forms) along a differentiable manifold  $U$  with values on a parallelizable manifold  $M$ .

Given a differentiable manifold  $U$  and a differentiable map  $\Phi : U \rightarrow M$  to a parallelizable manifold  $M$ , the set  $\Lambda^p(U, \Phi)$  of  $p$ -forms along  $U$  with values on  $M$  is a free module over  $C^k(U)$ , the commutative algebra of differentiable scalar fields on  $U$  (see *DiffScalarFieldAlgebra*). The standard case of  $p$ -forms on

a differentiable manifold  $M$  corresponds to  $U = M$  and  $\Phi = \text{Id}_M$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $M$  ( $U$  is then an open interval of  $\mathbf{R}$ ).

This class implements  $\Lambda^p(U, \Phi)$  in the case where  $M$  is parallelizable;  $\Lambda^p(U, \Phi)$  is then a *free* module. If  $M$  is not parallelizable, the class `DiffFormModule` must be used instead.

This is a Sage *parent* class, whose *element* class is `DiffFormParal`.

INPUT:

- `vector_field_module` – free module  $\mathcal{X}(U, \Phi)$  of vector fields along  $U$  associated with the map  $\Phi : U \rightarrow V$
- `degree` – positive integer; the degree  $p$  of the differential forms

EXAMPLES:

Free module of 2-forms on a parallelizable 3-dimensional manifold:

```
sage: M = Manifold(3, 'M')
sage: X.<x,y,z> = M.chart()
sage: XM = M.vector_field_module() ; XM
Free module X(M) of vector fields on the 3-dimensional differentiable
manifold M
sage: A = M.diff_form_module(2) ; A
Free module /\^2(M) of 2-forms on the 3-dimensional differentiable
manifold M
sage: latex(A)
\Lambda^{2}\left(M\right)
```

$A$  is a module over the algebra  $C^k(M)$  of (differentiable) scalar fields on  $M$ :

```
sage: A.category()
Category of finite dimensional modules over Algebra of differentiable
scalar fields on the 3-dimensional differentiable manifold M
sage: CM = M.scalar_field_algebra() ; CM
Algebra of differentiable scalar fields on the 3-dimensional
differentiable manifold M
sage: A in Modules(CM)
True
sage: A.base_ring()
Algebra of differentiable scalar fields on
the 3-dimensional differentiable manifold M
sage: A.base_module()
Free module X(M) of vector fields on
the 3-dimensional differentiable manifold M
sage: A.base_module() is XM
True
sage: A.rank()
3
```

Elements can be constructed from  $A$ . In particular,  $0$  yields the zero element of  $A$ :

```
sage: A(0)
2-form zero on the 3-dimensional differentiable manifold M
sage: A(0) is A.zero()
True
```

while non-zero elements are constructed by providing their components in a given vector frame:

```

sage: comp = [[0, 3*x, -z], [-3*x, 0, 4], [z, -4, 0]]
sage: a = A(comp, frame=X.frame(), name='a') ; a
2-form a on the 3-dimensional differentiable manifold M
sage: a.display()
a = 3*x dx/\dy - z dx/\dz + 4 dy/\dz

```

An alternative is to construct the 2-form from an empty list of components and to set the nonzero nonredundant components afterwards:

```

sage: a = A([], name='a')
sage: a[0,1] = 3*x # component in the manifold's default frame
sage: a[0,2] = -z
sage: a[1,2] = 4
sage: a.display()
a = 3*x dx/\dy - z dx/\dz + 4 dy/\dz

```

The module  $\Lambda^1(M)$  is nothing but the dual of  $\mathcal{X}(M)$  (the free module of vector fields on  $M$ ):

```

sage: L1 = M.diff_form_module(1) ; L1
Free module /\^1(M) of 1-forms on the 3-dimensional differentiable
manifold M
sage: L1 is XM.dual()
True

```

Since any tensor field of type  $(0, 1)$  is a 1-form, there is a coercion map from the set  $T^{(0,1)}(M)$  of such tensors to  $\Lambda^1(M)$ :

```

sage: T01 = M.tensor_field_module((0,1)) ; T01
Free module T^(0,1)(M) of type-(0,1) tensors fields on the
3-dimensional differentiable manifold M
sage: L1.has_coerce_map_from(T01)
True

```

There is also a coercion map in the reverse direction:

```

sage: T01.has_coerce_map_from(L1)
True

```

For a degree  $p \geq 2$ , the coercion holds only in the direction  $\Lambda^p(M) \rightarrow T^{(0,p)}(M)$ :

```

sage: T02 = M.tensor_field_module((0,2)); T02
Free module T^(0,2)(M) of type-(0,2) tensors fields on the
3-dimensional differentiable manifold M
sage: T02.has_coerce_map_from(A)
True
sage: A.has_coerce_map_from(T02)
False

```

The coercion map  $T^{(0,1)}(M) \rightarrow \Lambda^1(M)$  in action:

```

sage: b = T01([-x, 2, 3*y], name='b'); b
Tensor field b of type (0,1) on the 3-dimensional differentiable
manifold M
sage: b.display()
b = -x dx + 2 dy + 3*y dz
sage: lb = L1(b) ; lb
1-form b on the 3-dimensional differentiable manifold M

```

```
sage: lb.display()
b = -x dx + 2 dy + 3*y dz
```

The coercion map  $\Lambda^1(M) \rightarrow T^{(0,1)}(M)$  in action:

```
sage: tlb = T01(lb); tlb
Tensor field b of type (0,1) on
the 3-dimensional differentiable manifold M
sage: tlb == b
True
```

The coercion map  $\Lambda^2(M) \rightarrow T^{(0,2)}(M)$  in action:

```
sage: T02 = M.tensor_field_module((0,2)) ; T02
Free module T^(0,2)(M) of type-(0,2) tensors fields on the
3-dimensional differentiable manifold M
sage: ta = T02(a) ; ta
Tensor field a of type (0,2) on the 3-dimensional differentiable
manifold M
sage: ta.display()
a = 3*x dx*dy - z dx*dz - 3*x dy*dx + 4 dy*dz + z dz*dx - 4 dz*dy
sage: a.display()
a = 3*x dx/\dy - z dx/\dz + 4 dy/\dz
sage: ta.symmetries() # the antisymmetry is preserved
no symmetry; antisymmetry: (0, 1)
```

There is also coercion to subdomains, which is nothing but the restriction of the differential form to some subset of its domain:

```
sage: U = M.open_subset('U', coord_def={X: x^2+y^2<1})
sage: B = U.diff_form_module(2) ; B
Free module /\^2(U) of 2-forms on the Open subset U of the
3-dimensional differentiable manifold M
sage: B.has_coerce_map_from(A)
True
sage: a_U = B(a) ; a_U
2-form a on the Open subset U of the 3-dimensional differentiable
manifold M
sage: a_U.display()
a = 3*x dx/\dy - z dx/\dz + 4 dy/\dz
```

### Element

alias of `DiffFormParal`

**class** `sage.manifolds.differentiable.diff_form_module.DiffFormModule` (*vector\_field\_module*, *degree*)

Bases: `sage.structure.unique_representation.UniqueRepresentation`, `sage.structure.parent.Parent`

Module of differential forms of a given degree  $p$  ( $p$ -forms) along a differentiable manifold  $U$  with values on a differentiable manifold  $M$ .

Given a differentiable manifold  $U$  and a differentiable map  $\Phi : U \rightarrow M$  to a differentiable manifold  $M$ , the set  $\Lambda^p(U, \Phi)$  of  $p$ -forms along  $U$  with values on  $M$  is a module over  $C^k(U)$ , the commutative algebra of differentiable scalar fields on  $U$  (see *DiffScalarFieldAlgebra*). The standard case of  $p$ -forms on a differentiable manifold  $M$  corresponds to  $U = M$  and  $\Phi = \text{Id}_M$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $M$  ( $U$  is then an open interval of  $\mathbf{R}$ ).

**Note:** This class implements  $\Lambda^p(U, \Phi)$  in the case where  $M$  is not assumed to be parallelizable; the module  $\Lambda^p(U, \Phi)$  is then not necessarily free. If  $M$  is parallelizable, the class `DiffFormFreeModule` must be used instead.

INPUT:

- `vector_field_module` – module  $\mathcal{X}(U, \Phi)$  of vector fields along  $U$  with values on  $M$  via the map  $\Phi : U \rightarrow M$
- `degree` – positive integer; the degree  $p$  of the differential forms

EXAMPLES:

Module of 2-forms on a non-parallelizable 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U') ; V = M.open_subset('V')
sage: M.declare_union(U,V) # M is the union of U and V
sage: c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart()
sage: transf = c_xy.transition_map(c_uv, (x+y, x-y),
....: intersection_name='W', restrictions1= x>0, restrictions2= u+v>0)
sage: inv = transf.inverse()
sage: W = U.intersection(V)
sage: eU = c_xy.frame() ; eV = c_uv.frame()
sage: XM = M.vector_field_module() ; XM
Module X(M) of vector fields on the 2-dimensional differentiable
manifold M
sage: A = M.diff_form_module(2) ; A
Module /\^2(M) of 2-forms on the 2-dimensional differentiable manifold M
sage: latex(A)
\Lambda^{2}\left(M\right)
```

Modules of differential forms are unique:

```
sage: A is M.diff_form_module(2)
True
```

$\Lambda^2(M)$  is a module over the algebra  $C^k(M)$  of (differentiable) scalar fields on  $M$ :

```
sage: A.category()
Category of modules over Algebra of differentiable scalar fields on
the 2-dimensional differentiable manifold M
sage: CM = M.scalar_field_algebra() ; CM
Algebra of differentiable scalar fields on the 2-dimensional
differentiable manifold M
sage: A in Modules(CM)
True
sage: A.base_ring() is CM
True
sage: A.base_module()
Module X(M) of vector fields on the 2-dimensional differentiable
manifold M
sage: A.base_module() is XM
True
```

Elements can be constructed from `A()`. In particular, `0` yields the zero element of `A`:

```

sage: z = A(0) ; z
2-form zero on the 2-dimensional differentiable manifold M
sage: z.display(eU)
zero = 0
sage: z.display(eV)
zero = 0
sage: z is A.zero()
True

```

while non-zero elements are constructed by providing their components in a given vector frame:

```

sage: a = A([[0,3*x],[ -3*x,0]], frame=eU, name='a') ; a
2-form a on the 2-dimensional differentiable manifold M
sage: a.add_comp_by_continuation(eV, W, c_uv) # finishes the initialization of a
sage: a.display(eU)
a = 3*x dx/\dy
sage: a.display(eV)
a = (-3/4*u - 3/4*v) du/\dv

```

An alternative is to construct the 2-form from an empty list of components and to set the nonzero nonredundant components afterwards:

```

sage: a = A([], name='a')
sage: a[eU,0,1] = 3*x
sage: a.add_comp_by_continuation(eV, W, c_uv)
sage: a.display(eU)
a = 3*x dx/\dy
sage: a.display(eV)
a = (-3/4*u - 3/4*v) du/\dv

```

The module  $\Lambda^1(M)$  is nothing but the dual of  $\mathcal{X}(M)$  (the module of vector fields on  $M$ ):

```

sage: L1 = M.diff_form_module(1) ; L1
Module /\^1(M) of 1-forms on the 2-dimensional differentiable manifold M
sage: L1 is XM.dual()
True

```

Since any tensor field of type  $(0, 1)$  is a 1-form, there is a coercion map from the set  $T^{(0,1)}(M)$  of such tensors to  $\Lambda^1(M)$ :

```

sage: T01 = M.tensor_field_module((0,1)) ; T01
Module T^(0,1)(M) of type-(0,1) tensors fields on the 2-dimensional
differentiable manifold M
sage: L1.has_coerce_map_from(T01)
True

```

There is also a coercion map in the reverse direction:

```

sage: T01.has_coerce_map_from(L1)
True

```

For a degree  $p \geq 2$ , the coercion holds only in the direction  $\Lambda^p(M) \rightarrow T^{(0,p)}(M)$ :

```

sage: T02 = M.tensor_field_module((0,2)) ; T02
Module T^(0,2)(M) of type-(0,2) tensors fields on the 2-dimensional
differentiable manifold M
sage: T02.has_coerce_map_from(A)
True

```

```
sage: A.has_coerce_map_from(T02)
False
```

The coercion map  $T^{(0,1)}(M) \rightarrow \Lambda^1(M)$  in action:

```
sage: b = T01([y,x], frame=eU, name='b') ; b
Tensor field b of type (0,1) on the 2-dimensional differentiable
manifold M
sage: b.add_comp_by_continuation(eV, W, c_uv)
sage: b.display(eU)
b = y dx + x dy
sage: b.display(eV)
b = 1/2*u du - 1/2*v dv
sage: lb = L1(b) ; lb
1-form b on the 2-dimensional differentiable manifold M
sage: lb.display(eU)
b = y dx + x dy
sage: lb.display(eV)
b = 1/2*u du - 1/2*v dv
```

The coercion map  $\Lambda^1(M) \rightarrow T^{(0,1)}(M)$  in action:

```
sage: tlb = T01(lb) ; tlb
Tensor field b of type (0,1) on the 2-dimensional differentiable
manifold M
sage: tlb.display(eU)
b = y dx + x dy
sage: tlb.display(eV)
b = 1/2*u du - 1/2*v dv
sage: tlb == b
True
```

The coercion map  $\Lambda^2(M) \rightarrow T^{(0,2)}(M)$  in action:

```
sage: ta = T02(a) ; ta
Tensor field a of type (0,2) on the 2-dimensional differentiable
manifold M
sage: ta.display(eU)
a = 3*x dx*dy - 3*x dy*dx
sage: a.display(eU)
a = 3*x dx/\dy
sage: ta.display(eV)
a = (-3/4*u - 3/4*v) du*dv + (3/4*u + 3/4*v) dv*du
sage: a.display(eV)
a = (-3/4*u - 3/4*v) du/\dv
```

There is also coercion to subdomains, which is nothing but the restriction of the differential form to some subset of its domain:

```
sage: L2U = U.diff_form_module(2) ; L2U
Free module /\^2(U) of 2-forms on the Open subset U of the
2-dimensional differentiable manifold M
sage: L2U.has_coerce_map_from(A)
True
sage: a_U = L2U(a) ; a_U
2-form a on the Open subset U of the 2-dimensional differentiable
manifold M
```

```
sage: a_U.display(eU)
a = 3*x dx/\dy
```

**Element**

alias of `DiffForm`

**base\_module()**

Return the vector field module on which the differential form module is constructed.

**OUTPUT:**

- a *VectorFieldModule* representing the module on which the differential form module is defined

**EXAMPLES:**

```
sage: M = Manifold(3, 'M')
sage: A2 = M.diff_form_module(2) ; A2
Module /\^2(M) of 2-forms on the 3-dimensional differentiable
manifold M
sage: A2.base_module()
Module X(M) of vector fields on the 3-dimensional differentiable
manifold M
sage: A2.base_module() is M.vector_field_module()
True
sage: U = M.open_subset('U')
sage: A2U = U.diff_form_module(2) ; A2U
Module /\^2(U) of 2-forms on the Open subset U of the 3-dimensional
differentiable manifold M
sage: A2U.base_module()
Module X(U) of vector fields on the Open subset U of the
3-dimensional differentiable manifold M
```

**degree()**

Return the degree of the differential forms in the module.

**OUTPUT:**

- integer  $p$  such that the module is a set of  $p$ -forms

**EXAMPLES:**

```
sage: M = Manifold(3, 'M')
sage: M.diff_form_module(1).degree()
1
sage: M.diff_form_module(2).degree()
2
sage: M.diff_form_module(3).degree()
3
```

**zero()**

Return the zero of self.

**EXAMPLES:**

```
sage: M = Manifold(3, 'M')
sage: A2 = M.diff_form_module(2)
sage: A2.zero()
2-form zero on the 3-dimensional differentiable manifold M
```

## 2.9.2 Differential Forms

Let  $U$  and  $M$  be two differentiable manifolds. Given a positive integer  $p$  and a differentiable map  $\Phi : U \rightarrow M$ , a *differential form of degree  $p$ , or  $p$ -form, along  $U$  with values on  $M$*  is a field along  $U$  of alternating multilinear forms of degree  $p$  in the tangent spaces to  $M$ . The standard case of a differential form on a differentiable manifold corresponds to  $U = M$  and  $\Phi = \text{Id}_M$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $M$  ( $U$  is then an open interval of  $\mathbf{R}$ ).

Two classes implement differential forms, depending whether the manifold  $M$  is parallelizable:

- `DiffFormParal` when  $M$  is parallelizable
- `DiffForm` when  $M$  is not assumed parallelizable.

---

**Note:** A difference with the class `DifferentialForm` is that the present classes lie at the tensor field level. Accordingly, an instance of `DiffForm` or `DiffFormParal` can have various sets of components, each in a different coordinate system (or more generally in a different coframe), while the class `DifferentialForm` considers differential forms at the component level in a fixed chart. In this respect, the class `DifferentialForm` is closer to the class `CompFullyAntiSym` than to `DiffForm`.

---

### AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2013, 2014): initial version
- Joris Vankerschaver (2010): developed a previous class, `DifferentialForm` (cf. the above note), which inspired the storage of the non-zero components as a dictionary whose keys are the indices.
- Travis Scrimshaw (2016): review tweaks

### REFERENCES:

- [KN1963]
- [Lee2013]

**class** `sage.manifolds.differentiable.diff_form.DiffForm`(*vector\_field\_module*, *degree*, *name=None*, *latex\_name=None*)

Bases: `sage.manifolds.differentiable.tensorfield.TensorField`

Differential form with values on a generic (i.e. a priori not parallelizable) differentiable manifold.

Given a differentiable manifold  $U$ , a differentiable map  $\Phi : U \rightarrow M$  to a differentiable manifold  $M$  and a positive integer  $p$ , a *differential form of degree  $p$  (or  $p$ -form) along  $U$  with values on  $M \supset \Phi(U)$*  is a differentiable map

$$a : U \longrightarrow T^{(0,p)}M$$

( $T^{(0,p)}M$  being the tensor bundle of type  $(0, p)$  over  $M$ ) such that

$$a(x) \in \Lambda^p(T_{\Phi(x)}M)$$

for all  $x \in U$ , i.e.  $a(x)$  is an alternating multilinear form of degree  $p$  of the tangent space to  $M$  at the point  $\Phi(x)$ .

The standard case of a differential form on a manifold  $M$  corresponds to  $U = M$  and  $\Phi = \text{Id}_M$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $M$  ( $U$  is then an open interval of  $\mathbf{R}$ ).

---

**Note:** If  $M$  is parallelizable, the class `DiffFormParal` must be used instead.

---

## INPUT:

- `vector_field_module` – module  $\mathcal{X}(U, \Phi)$  of vector fields along  $U$  with values on  $M$  via the map  $\Phi$
- `degree` – the degree of the differential form (i.e. its tensor rank)
- `name` – (default: None) name given to the differential form
- `latex_name` – (default: None) LaTeX symbol to denote the differential form; if none is provided, the LaTeX symbol is set to `name`

## EXAMPLES:

Differential form of degree 2 on a non-parallelizable 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U') ; V = M.open_subset('V')
sage: M.declare_union(U,V) # M is the union of U and V
sage: c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart()
sage: xy_to_uv = c_xy.transition_map(c_uv, (x+y, x-y), intersection_name='W',
....:                               restrictions1= x>0, restrictions2= u+v>0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: W = U.intersection(V)
sage: eU = c_xy.frame() ; eV = c_uv.frame()
sage: a = M.diff_form(2, name='a') ; a
2-form a on the 2-dimensional differentiable manifold M
sage: a.parent()
Module /\^2(M) of 2-forms on the 2-dimensional differentiable
manifold M
sage: a.degree()
2
```

Setting the components of `a`:

```
sage: a[eU, 0, 1] = x*y^2 + 2*x
sage: a.add_comp_by_continuation(eV, W, c_uv)
sage: a.display(eU)
a = (x*y^2 + 2*x) dx/\dy
sage: a.display(eV)
a = (-1/16*u^3 + 1/16*u*v^2 - 1/16*v^3
+ 1/16*(u^2 - 8)*v - 1/2*u) du/\dv
```

A 1-form on  $M$ :

```
sage: a = M.one_form('a') ; a
1-form a on the 2-dimensional differentiable manifold M
sage: a.parent()
Module /\^1(M) of 1-forms on the 2-dimensional differentiable
manifold M
sage: a.degree()
1
```

Setting the components of the 1-form in a consistent way:

```
sage: a[eU, :] = [-y, x]
sage: a.add_comp_by_continuation(eV, W, c_uv)
sage: a.display(eU)
a = -y dx + x dy
sage: a.display(eV)
a = 1/2*v du - 1/2*u dv
```

The exterior derivative of the 1-form is a 2-form:

```
sage: da = a.exterior_derivative() ; da
2-form da on the 2-dimensional differentiable manifold M
sage: da.display(eU)
da = 2 dx/\dy
sage: da.display(eV)
da = -du/\dv
```

Another 1-form:

```
sage: b = M.one_form('b')
sage: b[eU, :] = [1+x*y, x^2]
sage: b.add_comp_by_continuation(eV, W, c_uv)
```

Adding two 1-forms results in another 1-form:

```
sage: s = a + b ; s
1-form a+b on the 2-dimensional differentiable manifold M
sage: s.display(eU)
a+b = ((x - 1)*y + 1) dx + (x^2 + x) dy
sage: s.display(eV)
a+b = (1/4*u^2 + 1/4*(u + 2)*v + 1/2) du
+ (-1/4*u*v - 1/4*v^2 - 1/2*u + 1/2) dv
```

The exterior product of two 1-forms is a 2-form:

```
sage: s = a.wedge(b) ; s
2-form a/\b on the 2-dimensional differentiable manifold M
sage: s.display(eU)
a/\b = (-2*x^2*y - x) dx/\dy
sage: s.display(eV)
a/\b = (1/8*u^3 - 1/8*u*v^2 - 1/8*v^3 + 1/8*(u^2 + 2)*v + 1/4*u) du/\dv
```

Multiplying a 1-form by a scalar field results in another 1-form:

```
sage: f = M.scalar_field({c_xy: (x+y)^2, c_uv: u^2}, name='f')
sage: s = f*a ; s
1-form on the 2-dimensional differentiable manifold M
sage: s.display(eU)
(-x^2*y - 2*x*y^2 - y^3) dx + (x^3 + 2*x^2*y + x*y^2) dy
sage: s.display(eV)
1/2*u^2*v du - 1/2*u^3 dv
```

**degree()**

Return the degree of self.

OUTPUT:

- integer  $p$  such that the differential form is a  $p$ -form

EXAMPLES:

```
sage: M = Manifold(3, 'M')
sage: a = M.diff_form(2); a
2-form on the 3-dimensional differentiable manifold M
sage: a.degree()
2
sage: b = M.diff_form(1); b
```

```
1-form on the 3-dimensional differentiable manifold M
sage: b.degree()
1
```

**exterior\_derivative()**

Compute the exterior derivative of self.

OUTPUT:

- instance of *DiffForm* representing the exterior derivative of the differential form

EXAMPLES:

Exterior derivative of a 1-form on the 2-sphere:

```
sage: M = Manifold(2, 'M') # the 2-dimensional sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                               intersection_name='W', restrictions1= x^2+y^2!=0,
....:                               restrictions2= u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: e_xy = c_xy.frame(); e_uv = c_uv.frame()
```

The 1-form:

```
sage: a = M.diff_form(1, name='a')
sage: a[e_xy,:] = -y^2, x^2
sage: a.add_comp_by_continuation(e_uv, U.intersection(V), c_uv)
sage: a.display(e_xy)
a = -y^2 dx + x^2 dy
sage: a.display(e_uv)
a = -(2*u^3*v - u^2*v^2 + v^4)/(u^8 + 4*u^6*v^2 + 6*u^4*v^4 + 4*u^2*v^6 + v^
↪8) du
+ (u^4 - u^2*v^2 + 2*u*v^3)/(u^8 + 4*u^6*v^2 + 6*u^4*v^4 + 4*u^2*v^6 + v^8) ↪
↪dv
```

Its exterior derivative:

```
sage: da = a.exterior_derivative(); da
2-form da on the 2-dimensional differentiable manifold M
sage: da.display(e_xy)
da = (2*x + 2*y) dx/\dy
sage: da.display(e_uv)
da = -2*(u + v)/(u^6 + 3*u^4*v^2 + 3*u^2*v^4 + v^6) du/\dv
```

The result is cached, i.e. is not recomputed unless a is changed:

```
sage: a.exterior_derivative() is da
True
```

Instead of invoking the method `exterior_derivative()`, one may use the global function `exterior_derivative()` or its alias `xder()`:

```
sage: from sage.manifolds.utilities import xder
sage: xder(a) is a.exterior_derivative()
True
```

Let us check Cartan's identity:

```
sage: v = M.vector_field(name='v')
sage: v[e_xy, :] = -y, x
sage: v.add_comp_by_continuation(e_uv, U.intersection(V), c_uv)
sage: a.lie_der(v) == v.contract(xder(a)) + xder(a(v)) # long time
True
```

### `hodge_dual` (metric)

Compute the Hodge dual of the differential form with respect to some metric.

If the differential form is a  $p$ -form  $A$ , its *Hodge dual* with respect to a pseudo-Riemannian metric  $g$  is the  $(n - p)$ -form  $*A$  defined by

$$*A_{i_1 \dots i_{n-p}} = \frac{1}{p!} A_{k_1 \dots k_p} \epsilon^{k_1 \dots k_p}_{i_1 \dots i_{n-p}}$$

where  $n$  is the manifold's dimension,  $\epsilon$  is the volume  $n$ -form associated with  $g$  (see `volume_form()`) and the indices  $k_1, \dots, k_p$  are raised with  $g$ .

INPUT:

- `metric`: a pseudo-Riemannian metric defined on the same manifold as the current differential form; must be an instance of `PseudoRiemannianMetric`

OUTPUT:

- the  $(n - p)$ -form  $*A$

EXAMPLES:

Hodge dual of a 1-form on the 2-sphere equipped with the standard metric: we first construct  $\mathbb{S}^2$  and its metric  $g$ :

```
sage: M = Manifold(2, 'S^2', start_index=1)
sage: U = M.open_subset('U') ; V = M.open_subset('V')
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart() # stereographic coord.
↔ (North and South)
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....: intersection_name='W', restrictions1= x^2+y^2!=0,
....: restrictions2= u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: W = U.intersection(V) # The complement of the two poles
sage: eU = c_xy.frame() ; eV = c_uv.frame()
sage: g = M.metric('g')
sage: g[eU,1,1], g[eU,2,2] = 4/(1+x^2+y^2)^2, 4/(1+x^2+y^2)^2
sage: g[eV,1,1], g[eV,2,2] = 4/(1+u^2+v^2)^2, 4/(1+u^2+v^2)^2
```

Then we construct the 1-form and take its Hodge dual w.r.t.  $g$ :

```
sage: a = M.one_form(name='a')
sage: a[eU, :] = -y, x
sage: a.add_comp_by_continuation(eV, W, c_uv)
sage: a.display(eU)
a = -y dx + x dy
```

```

sage: a.display(eV)
a = -v/(u^4 + 2*u^2*v^2 + v^4) du + u/(u^4 + 2*u^2*v^2 + v^4) dv
sage: sa = a.hodge_dual(g); sa
1-form *a on the 2-dimensional differentiable manifold S^2
sage: sa.display(eU)
*a = -x dx - y dy
sage: sa.display(eV)
*a = -u/(u^4 + 2*u^2*v^2 + v^4) du - v/(u^4 + 2*u^2*v^2 + v^4) dv

```

Instead of calling the method `hodge_dual()` on the differential form, one can invoke the method `hodge_star()` of the metric:

```

sage: a.hodge_dual(g) == g.hodge_star(a)
True

```

For a 1-form and a Riemannian metric in dimension 2, the Hodge dual applied twice is minus the identity:

```

sage: ssa = sa.hodge_dual(g); ssa
1-form **a on the 2-dimensional differentiable manifold S^2
sage: ssa == -a
True

```

The Hodge dual of the metric volume 2-form is the constant scalar field 1 (considered as a 0-form):

```

sage: eps = g.volume_form(); eps
2-form eps_g on the 2-dimensional differentiable manifold S^2
sage: eps.display(eU)
eps_g = 4/(x^4 + y^4 + 2*(x^2 + 1)*y^2 + 2*x^2 + 1) dx/\dy
sage: eps.display(eV)
eps_g = 4/(u^4 + v^4 + 2*(u^2 + 1)*v^2 + 2*u^2 + 1) du/\dv
sage: seps = eps.hodge_dual(g); seps
Scalar field *eps_g on the 2-dimensional differentiable manifold S^2
sage: seps.display()
*eps_g: S^2 --> R
on U: (x, y) |--> 1
on V: (u, v) |--> 1

```

### wedge (*other*)

Exterior product with another differential form.

INPUT:

- `other` – another differential form (on the same manifold)

OUTPUT:

- a `DiffForm` of the exterior product `self/\other`

EXAMPLES:

Exterior product of two 1-forms on the 2-sphere:

```

sage: M = Manifold(2, 'S^2', start_index=1) # the 2-dimensional sphere S^2
sage: U = M.open_subset('U') ; V = M.open_subset('V')
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart() # stereographic coord.
↔ (North and South)
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                                     intersection_name='W', restrictions1= x^2+y^2!=0,
....:                                     restrictions2= u^2+v^2!=0)

```

```

sage: uv_to_xy = xy_to_uv.inverse()
sage: W = U.intersection(V) # The complement of the two poles
sage: e_xy = c_xy.frame() ; e_uv = c_uv.frame()
sage: a = M.diff_form(1, name='a')
sage: a[e_xy,:] = y, x
sage: a.add_comp_by_continuation(e_uv, W, c_uv)
sage: b = M.diff_form(1, name='b')
sage: b[e_xy,:] = x^2 + y^2, y
sage: b.add_comp_by_continuation(e_uv, W, c_uv)
sage: c = a.wedge(b); c
2-form a/\b on the 2-dimensional differentiable manifold S^2
sage: c.display(e_xy)
a/\b = (-x^3 - (x - 1)*y^2) dx/\dy
sage: c.display(e_uv)
a/\b = -(v^2 - u)/(u^8 + 4*u^6*v^2 + 6*u^4*v^4 + 4*u^2*v^6 + v^8) du/\dv

```

**class** sage.manifolds.differentiable.diff\_form.**DiffFormParal**(vector\_field\_module, degree, name=None, latex\_name=None)

Bases: sage.tensor.modules.free\_module\_alt\_form.FreeModuleAltForm, sage.manifolds.differentiable.tensorfield\_paral.TensorFieldParal

Differential form with values on a parallelizable manifold.

Given a differentiable manifold  $U$ , a differentiable map  $\Phi : U \rightarrow M$  to a parallelizable manifold  $M$  and a positive integer  $p$ , a *differential form of degree  $p$*  (or  $p$ -form) along  $U$  with values on  $M \supset \Phi(U)$  is a differentiable map

$$a : U \longrightarrow T^{(0,p)}M$$

( $T^{(0,p)}M$  being the tensor bundle of type  $(0, p)$  over  $M$ ) such that

$$a(x) \in \Lambda^p(T_{\Phi(x)}M)$$

for all  $x \in U$ , i.e.  $a(x)$  is an alternating multilinear form of degree  $p$  of the tangent space to  $M$  at the point  $\Phi(x)$ .

The standard case of a differential form on a manifold  $M$  corresponds to  $U = M$  and  $\Phi = \text{Id}_M$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $M$  ( $U$  is then an open interval of  $\mathbf{R}$ ).

---

**Note:** If  $M$  is not parallelizable, the class *DiffForm* must be used instead.

---

INPUT:

- vector\_field\_module – free module  $\mathcal{X}(U, \Phi)$  of vector fields along  $U$  with values on  $M$  via the map  $\Phi$
- degree – the degree of the differential form (i.e. its tensor rank)
- name – (default: None) name given to the differential form
- latex\_name – (default: None) LaTeX symbol to denote the differential form; if none is provided, the LaTeX symbol is set to name

EXAMPLES:

A 2-form on a 4-dimensional manifold:

```

sage: M = Manifold(4, 'M')
sage: c_txyz.<t,x,y,z> = M.chart()
sage: a = M.diff_form(2, 'a') ; a
2-form a on the 4-dimensional differentiable manifold M
sage: a.parent()
Free module /\^2(M) of 2-forms on the 4-dimensional differentiable
manifold M

```

A differential form is a tensor field of purely covariant type:

```

sage: a.tensor_type()
(0, 2)

```

It is antisymmetric, its components being `CompFullyAntiSym`:

```

sage: a.symmetries()
no symmetry; antisymmetry: (0, 1)
sage: a[0,1] = 2
sage: a[1,0]
-2
sage: a.comp()
Fully antisymmetric 2-indices components w.r.t. Coordinate frame (M, (d/dt,d/dx,d/
↪dy,d/dz))
sage: type(a.comp())
<class 'sage.tensor.modules.comp.CompFullyAntiSym'>

```

Setting a component with repeated indices to a non-zero value results in an error:

```

sage: a[1,1] = 3
Traceback (most recent call last):
...
ValueError: by antisymmetry, the component cannot have a nonzero value
for the indices (1, 1)
sage: a[1,1] = 0 # OK, albeit useless
sage: a[1,2] = 3 # OK

```

The expansion of a differential form with respect to a given coframe is displayed via the method `display()`:

```

sage: a.display() # expansion with respect to the default coframe (dt, dx, dy, dz)
a = 2 dt/\dx + 3 dx/\dy
sage: latex(a.display()) # output for the notebook
a = 2 \mathrm{d} t\wedge \mathrm{d} x
+ 3 \mathrm{d} x\wedge \mathrm{d} y

```

Differential forms can be added or subtracted:

```

sage: b = M.diff_form(2)
sage: b[0,1], b[0,2], b[0,3] = (1,2,3)
sage: s = a + b ; s
2-form on the 4-dimensional differentiable manifold M
sage: a[:], b[:], s[:]
(
[ 0 2 0 0] [ 0 1 2 3] [ 0 3 2 3]
[-2 0 3 0] [-1 0 0 0] [-3 0 3 0]
[ 0 -3 0 0] [-2 0 0 0] [-2 -3 0 0]
[ 0 0 0 0], [-3 0 0 0], [-3 0 0 0]
)
sage: s = a - b ; s

```

```

2-form on the 4-dimensional differentiable manifold M
sage: s[:]
[ 0  1 -2 -3]
[-1  0  3  0]
[ 2 -3  0  0]
[ 3  0  0  0]

```

An example of 3-form is the volume element on  $\mathbf{R}^3$  in Cartesian coordinates:

```

sage: M = Manifold(3, 'R3', '\RR^3', start_index=1)
sage: c_cart.<x,y,z> = M.chart()
sage: eps = M.diff_form(3, 'epsilon', r'\epsilon')
sage: eps[1,2,3] = 1 # the only independent component
sage: eps[:] # all the components are set from the previous line:
[[[0, 0, 0], [0, 0, 1], [0, -1, 0]], [[0, 0, -1], [0, 0, 0], [1, 0, 0]],
 [[0, 1, 0], [-1, 0, 0], [0, 0, 0]]]
sage: eps.display()
epsilon = dx/\dy/\dz

```

Spherical components of the volume element from the tensorial change-of-frame formula:

```

sage: c_spher.<r,th,ph> = M.chart(r'r:[0,+oo] th:[0,pi]:\theta ph:[0,2*pi]:\phi')
sage: spher_to_cart = c_spher.transition_map(c_cart,
.....: [r*sin(th)*cos(ph), r*sin(th)*sin(ph), r*cos(th)])
sage: cart_to_spher = spher_to_cart.set_inverse(sqrt(x^2+y^2+z^2),
.....: atan2(sqrt(x^2+y^2),z), atan2(y, x))
sage: eps.comp(c_spher.frame()) # computation of the components in the spherical_
↪frame
Fully antisymmetric 3-indices components w.r.t. Coordinate frame
(R3, (d/dr,d/dth,d/dph))
sage: eps.comp(c_spher.frame())[1,2,3, c_spher]
r^2*sin(th)
sage: eps.display(c_spher.frame())
epsilon = sqrt(x^2 + y^2 + z^2)*sqrt(x^2 + y^2) dr/\dth/\dph
sage: eps.display(c_spher.frame(), c_spher)
epsilon = r^2*sin(th) dr/\dth/\dph

```

The exterior product of two differential forms is performed via the method `wedge()`:

```

sage: a = M.one_form('A')
sage: a[:] = (x*y*z, -z*x, y*z)
sage: b = M.one_form('B')
sage: b[:] = (cos(z), sin(x), cos(y))
sage: ab = a.wedge(b) ; ab
2-form A/\B on the 3-dimensional differentiable manifold R3
sage: ab[:]
[
0 x*y*z*sin(x) + x*z*cos(z) x*y*z*cos(y) - y*z*cos(z)
[-x*y*z*sin(x) - x*z*cos(z) 0 -(x*cos(y) + y*sin(x))*z]
[-x*y*z*cos(y) + y*z*cos(z) (x*cos(y) + y*sin(x))*z 0]
sage: ab.display()
A/\B = (x*y*z*sin(x) + x*z*cos(z)) dx/\dy + (x*y*z*cos(y) - y*z*cos(z)) dx/\dz
- (x*cos(y) + y*sin(x))*z dy/\dz

```

The tensor product of a 1-form and a 2-form is not a 3-form but a tensor field of type  $(0, 3)$  with less symmetries:

```

sage: c = a*ab ; c
Tensor field A*(A/\B) of type (0,3) on the 3-dimensional differentiable
manifold R3

```

```

sage: c.symmetries() # the antisymmetry is only w.r.t. the last two arguments:
no symmetry; antisymmetry: (1, 2)
sage: d = ab*a ; d
Tensor field (A/\B)*A of type (0,3) on the 3-dimensional differentiable
manifold R3
sage: d.symmetries() # the antisymmetry is only w.r.t. the first two arguments:
no symmetry; antisymmetry: (0, 1)

```

The exterior derivative of a differential form is obtained by means of the `exterior_derivative()`:

```

sage: da = a.exterior_derivative() ; da
2-form dA on the 3-dimensional differentiable manifold R3
sage: da.display()
dA = -(x + 1)*z dx/\dy - x*y dx/\dz + (x + z) dy/\dz
sage: db = b.exterior_derivative() ; db
2-form dB on the 3-dimensional differentiable manifold R3
sage: db.display()
dB = cos(x) dx/\dy + sin(z) dx/\dz - sin(y) dy/\dz
sage: dab = ab.exterior_derivative() ; dab
3-form d(A/\B) on the 3-dimensional differentiable manifold R3

```

As a 3-form over a 3-dimensional manifold,  $d(A/\B)$  is necessarily proportional to the volume 3-form:

```

sage: dab == dab[[1,2,3]]/eps[[1,2,3]]*eps
True

```

We may also check that the classical anti-derivation formula is fulfilled:

```

sage: dab == da.wedge(b) - a.wedge(db)
True

```

The Lie derivative of a 2-form is a 2-form:

```

sage: v = M.vector_field('v')
sage: v[:] = (y*z, -x*z, x*y)
sage: ab.lie_der(v) # long time
2-form on the 3-dimensional differentiable manifold R3

```

Let us check Cartan formula, which expresses the Lie derivative in terms of exterior derivatives:

```

sage: ab.lie_der(v) == (v.contract(ab.exterior_derivative()) # long time
....:                  + v.contract(ab).exterior_derivative())
True

```

A 1-form on a  $\mathbf{R}^3$ :

```

sage: om = M.one_form('omega', r'\omega') ; om
1-form omega on the 3-dimensional differentiable manifold R3

```

A 1-form is of course a differential form:

```

sage: isinstance(om, sage.manifolds.differentiable.diff_form.DiffFormParal)
True
sage: om.parent()
Free module /\^1(R3) of 1-forms on the 3-dimensional differentiable
manifold R3
sage: om.tensor_type()
(0, 1)

```

Setting the components with respect to the manifold's default frame:

```
sage: om[:] = (2*z, x, x-y)
sage: om[:]
[2*z, x, x - y]
sage: om.display()
omega = 2*z dx + x dy + (x - y) dz
```

A 1-form acts on vector fields:

```
sage: v = M.vector_field('V')
sage: v[:] = (x, 2*y, 3*z)
sage: om(v)
Scalar field omega(V) on the 3-dimensional differentiable manifold R3
sage: om(v).display()
omega(V): R3 --> R
(x, y, z) |--> 2*x*y + (5*x - 3*y)*z
(r, th, ph) |--> 2*r^2*cos(ph)*sin(ph)*sin(th)^2 + r^2*(5*cos(ph)
- 3*sin(ph))*cos(th)*sin(th)
sage: latex(om(v))
\omega\left(V\right)
```

The tensor product of two 1-forms is a tensor field of type (0,2):

```
sage: a = M.one_form('A')
sage: a[:] = (1, 2, 3)
sage: b = M.one_form('B')
sage: b[:] = (6, 5, 4)
sage: c = a*b ; c
Tensor field A*B of type (0,2) on the 3-dimensional differentiable
manifold R3
sage: c[:]
[ 6  5  4]
[12 10  8]
[18 15 12]
sage: c.symmetries() # c has no symmetries:
no symmetry; no antisymmetry
```

The exterior product of two 1-forms is a 2-form:

```
sage: d = a.wedge(b) ; d
2-form A/\B on the 3-dimensional differentiable manifold R3
sage: d[:]
[ 0 -7 -14]
[ 7  0 -7]
[14  7  0]
sage: d.symmetries()
no symmetry; antisymmetry: (0, 1)
```

We can check the standard formula relating the exterior product to the tensor product:

```
sage: a.wedge(b) == a*b - b*a
True
```

**exterior\_derivative()**

Compute the exterior derivative of self.

OUTPUT:

- a `DiffFormParal` representing the exterior derivative of the differential form

EXAMPLES:

Exterior derivative of a 1-form on a 4-dimensional manifold:

```
sage: M = Manifold(4, 'M')
sage: c_txyz.<t,x,y,z> = M.chart()
sage: a = M.one_form('A')
sage: a[:] = (t*x*y*z, z*y**2, x*z**2, x**2 + y**2)
sage: da = a.exterior_derivative() ; da
2-form dA on the 4-dimensional differentiable manifold M
sage: da.display()
dA = -t*y*z dt/\dx - t*x*z dt/\dy - t*x*y dt/\dz
      + (-2*y*z + z^2) dx/\dy + (-y^2 + 2*x) dx/\dz
      + (-2*x*z + 2*y) dy/\dz
sage: latex(da)
\mathrm{d}A
```

The result is cached, i.e. is not recomputed unless `a` is changed:

```
sage: a.exterior_derivative() is da
True
```

Instead of invoking the method `exterior_derivative()`, one may use the global function `exterior_derivative()` or its alias `xder()`:

```
sage: from sage.manifolds.utilities import xder
sage: xder(a) is a.exterior_derivative()
True
```

The exterior derivative is nilpotent:

```
sage: dda = da.exterior_derivative() ; dda
3-form ddA on the 4-dimensional differentiable manifold M
sage: dda.display()
ddA = 0
sage: dda == 0
True
```

Let us check Cartan's identity:

```
sage: v = M.vector_field(name='v')
sage: v[:] = -y, x, t, z
sage: a.lie_der(v) == v.contract(xder(a)) + xder(a(v)) # long time
True
```

### `hodge_dual` (metric)

Compute the Hodge dual of the differential form with respect to some metric.

If the differential form is a  $p$ -form  $A$ , its *Hodge dual* with respect to a pseudo-Riemannian metric  $g$  is the  $(n - p)$ -form  $*A$  defined by

$$*A_{i_1 \dots i_{n-p}} = \frac{1}{p!} A_{k_1 \dots k_p} \epsilon^{k_1 \dots k_p}_{i_1 \dots i_{n-p}}$$

where  $n$  is the manifold's dimension,  $\epsilon$  is the volume  $n$ -form associated with  $g$  (see `volume_form()`) and the indices  $k_1, \dots, k_p$  are raised with  $g$ .

INPUT:

- `metric`: a pseudo-Riemannian metric defined on the same manifold as the current differential form; must be an instance of `PseudoRiemannianMetric`

OUTPUT:

- the  $(n - p)$ -form  $*A$

EXAMPLES:

Hodge dual of a 1-form in the Euclidean space  $R^3$ :

```
sage: M = Manifold(3, 'M', start_index=1)
sage: X.<x,y,z> = M.chart()
sage: g = M.metric('g') # the Euclidean metric
sage: g[1,1], g[2,2], g[3,3] = 1, 1, 1
sage: a = M.one_form('A')
sage: var('Ax Ay Az')
(Ax, Ay, Az)
sage: a[:] = (Ax, Ay, Az)
sage: sa = a.hodge_dual(g) ; sa
2-form *A on the 3-dimensional differentiable manifold M
sage: sa.display()
*A = Az dx/\dy - Ay dx/\dz + Ax dy/\dz
sage: ssa = sa.hodge_dual(g) ; ssa
1-form **A on the 3-dimensional differentiable manifold M
sage: ssa.display()
**A = Ax dx + Ay dy + Az dz
sage: ssa == a # must hold for a Riemannian metric in dimension 3
True
```

Instead of calling the method `hodge_dual()` on the differential form, one can invoke the method `hodge_star()` of the metric:

```
sage: a.hodge_dual(g) == g.hodge_star(a)
True
```

See the documentation of `hodge_star()` for more examples.

**wedge** (*other*)

Exterior product of `self` with another differential form.

INPUT:

- `other` – another differential form

OUTPUT:

- instance of `DiffFormParal` representing the exterior product `self/\other`

EXAMPLES:

Exterior product of a 1-form and a 2-form on a 3-dimensional manifold:

```
sage: M = Manifold(3, 'M', start_index=1)
sage: X.<x,y,z> = M.chart()
sage: a = M.one_form(name='a')
sage: a[:] = [2, 1+x, y*z]
sage: b = M.diff_form(2, name='b')
sage: b[1,2], b[1,3], b[2,3] = y^2, z+x, z^2
sage: a.display()
a = 2 dx + (x + 1) dy + y*z dz
sage: b.display()
```

```

b = y^2 dx/\dy + (x + z) dx/\dz + z^2 dy/\dz
sage: s = a.wedge(b); s
3-form a/\b on the 3-dimensional differentiable manifold M
sage: s.display()
a/\b = (-x^2 + (y^3 - x - 1)*z + 2*z^2 - x) dx/\dy/\dz

```

Check:

```

sage: s[1,2,3] == a[1]*b[2,3] + a[2]*b[3,1] + a[3]*b[1,2]
True

```

## 2.10 Affine Connections

The class *AffineConnection* implements affine connections on smooth manifolds.

AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2013-2015) : initial version
- Marco Mancini (2015) : parallelization of some computations

REFERENCES:

- [Lee1997]
- [KN1963]
- [ONe1983]

**class** sage.manifolds.differentiable.affine\_connection.**AffineConnection** (*domain*,  
*name*,  
*la-*  
*tex\_name=None*)

Bases: sage.structure.sage\_object.SageObject

Affine connection on a smooth manifold.

Let  $M$  be a differentiable manifold of class  $C^\infty$  (smooth manifold) over a non-discrete topological field  $K$  (in most applications  $K = \mathbf{R}$  or  $K = \mathbf{C}$ ), let  $C^\infty(M)$  be the algebra of smooth functions  $M \rightarrow K$  (cf. *DiffScalarFieldAlgebra*) and let  $\mathcal{X}(M)$  be the  $C^\infty(M)$ -module of vector fields on  $M$  (cf. *VectorFieldModule*). An *affine connection* on  $M$  is an operator

$$\begin{aligned} \nabla : \mathcal{X}(M) \times \mathcal{X}(M) &\longrightarrow \mathcal{X}(M) \\ (u, v) &\longmapsto \nabla_u v \end{aligned}$$

that

- is  $K$ -bilinear, i.e. is bilinear when considering  $\mathcal{X}(M)$  as a vector space over  $K$
- is  $C^\infty(M)$ -linear w.r.t. the first argument:  $\forall f \in C^\infty(M), \nabla_{fv} = f\nabla_u v$
- obeys Leibniz rule w.r.t. the second argument:  $\forall f \in C^\infty(M), \nabla_u(fv) = df(u)v + f\nabla_u v$

The affine connection  $\nabla$  gives birth to the *covariant derivative operator* acting on tensor fields, denoted by the same symbol:

$$\begin{aligned} \nabla : T^{(k,l)}(M) &\longrightarrow T^{(k,l+1)}(M) \\ t &\longmapsto \nabla t \end{aligned}$$

where  $T^{(k,l)}(M)$  stands for the  $C^\infty(M)$ -module of tensor fields of type  $(k,l)$  on  $M$  (cf. `TensorFieldModule`), with the convention  $T^{(0,0)}(M) := C^\infty(M)$ . For a vector field  $v$ , the covariant derivative  $\nabla v$  is a type-(1,1) tensor field such that

$$\forall u \in \mathcal{X}(M), \nabla_u v = \nabla v(\cdot, u)$$

More generally for any tensor field  $t \in T^{(k,l)}(M)$ , we have

$$\forall u \in \mathcal{X}(M), \nabla_u t = \nabla t(\dots, u)$$

**Note:** The above convention means that, in terms of index notation, the “derivation index” in  $\nabla t$  is the *last* one:

$$\nabla_c t^{a_1 \dots a_k}_{b_1 \dots b_l} = (\nabla t)^{a_1 \dots a_k}_{b_1 \dots b_l c}$$

INPUT:

- `domain` – the manifold on which the connection is defined (must be an instance of class `DifferentiableManifold`)
- `name` – name given to the affine connection
- `latex_name` – (default: None) LaTeX symbol to denote the affine connection; if None, it is set to name.

EXAMPLES:

Affine connection on a 3-dimensional manifold:

```
sage: M = Manifold(3, 'M', start_index=1)
sage: c_xyz.<x,y,z> = M.chart()
sage: nab = M.affine_connection('nabla', r'\nabla') ; nab
Affine connection nabla on the 3-dimensional differentiable manifold M
```

A just-created connection has no connection coefficients:

```
sage: nab._coefficients
{}
```

The connection coefficients relative to the manifold’s default frame [here  $(\partial/\partial x, \partial/\partial y, \partial/\partial z)$ ], are created by providing the relevant indices inside square brackets:

```
sage: nab[1,1,2], nab[3,2,3] = x^2, y*z # Gamma^1_{12} = x^2, Gamma^3_{23} = yz
sage: nab._coefficients
{Coordinate frame (M, (d/dx,d/dy,d/dz)): 3-indices components w.r.t.
Coordinate frame (M, (d/dx,d/dy,d/dz))}
```

If not the default one, the vector frame w.r.t. which the connection coefficients are defined can be specified as the first argument inside the square brackets; hence the above definition is equivalent to:

```
sage: nab[c_xyz.frame(), 1,1,2], nab[c_xyz.frame(), 3,2,3] = x^2, y*z
sage: nab._coefficients
{Coordinate frame (M, (d/dx,d/dy,d/dz)): 3-indices components w.r.t.
Coordinate frame (M, (d/dx,d/dy,d/dz))}
```

Unset components are initialized to zero:

```
sage: nab[:] # list of coefficients relative to the manifold's default vector_
↳frame
[[[0, x^2, 0], [0, 0, 0], [0, 0, 0]],
 [[0, 0, 0], [0, 0, 0], [0, 0, 0]],
 [[0, 0, 0], [0, 0, y*z], [0, 0, 0]]]
```

The treatment of connection coefficients in a given vector frame is similar to that of tensor components; see therefore the class `TensorField` for the documentation. In particular, the square brackets return the connection coefficients as instances of `CoordFunction` (of the subclass `CoordFunctionSymb` in the current example), while the double square brackets return a scalar field:

```
sage: nab[1,1,2]
x^2
sage: nab[1,1,2].display()
(x, y, z) |--> x^2
sage: type(nab[1,1,2])
<class 'sage.manifolds.coord_func_symb.CoordFunctionSymbRing_with_category.
↳element_class'>
sage: nab[[1,1,2]]
Scalar field on the 3-dimensional differentiable manifold M
sage: nab[[1,1,2]].display()
M --> R
(x, y, z) |--> x^2
sage: nab[[1,1,2]].coord_function() is nab[1,1,2]
True
```

Action on a scalar field:

```
sage: f = M.scalar_field(x^2 - y^2, name='f')
sage: Df = nab(f) ; Df
1-form df on the 3-dimensional differentiable manifold M
sage: Df[:]
[2*x, -2*y, 0]
```

The action of an affine connection connection on a scalar field must coincide with the differential:

```
sage: Df == f.differential()
True
```

A generic affine connection has some torsion:

```
sage: DDf = nab(Df) ; DDf
Tensor field nabla(df) of type (0,2) on the 3-dimensional
differentiable manifold M
sage: DDf.antisymmetrize()[:] # nabla does not commute on scalar fields:
[ 0 -x^3  0]
[ x^3  0  0]
[ 0  0  0]
```

Let us check the standard formula

$$\nabla_j \nabla_i f - \nabla_i \nabla_j f = T_{ij}^k \nabla_k f,$$

where the  $T_{ij}^k$ 's are the components of the connection's torsion tensor:

```
sage: 2*DDf.antisymmetrize() == nab.torsion().contract(0,Df)
True
```

The connection acting on a vector field:

```
sage: v = M.vector_field('v')
sage: v[:] = (y*z, x*z, x*y)
sage: Dv = nab(v) ; Dv
Tensor field nabla(v) of type (1,1) on the 3-dimensional differentiable
manifold M
sage: Dv[:]
[
  0 (x^2*y + 1)*z          y]
[
  z          0          x]
[
  y          x          x*y*z^2]
```

Another example: connection on a non-parallelizable 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U') ; V = M.open_subset('V')
sage: M.declare_union(U,V) # M is the union of U and V
sage: c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart()
sage: transf = c_xy.transition_map(c_uv, (x+y, x-y), intersection_name='W',
....:                                restrictions1= x>0, restrictions2= u+v>0)
sage: inv = transf.inverse()
sage: W = U.intersection(V)
sage: eU = c_xy.frame() ; eV = c_uv.frame()
sage: c_xyW = c_xy.restrict(W) ; c_uvW = c_uv.restrict(W)
sage: eUW = c_xyW.frame() ; eVW = c_uvW.frame()
sage: nab = M.affine_connection('nabla', r'\nabla')
```

The connection is first defined on the open subset  $U$  by means of its coefficients w.r.t. the frame  $eU$  (the manifold's default frame):

```
sage: nab[0,0,0], nab[1,0,1] = x, x*y
```

The coefficients w.r.t the frame  $eV$  are deduced by continuation of the coefficients w.r.t. the frame  $eVW$  on the open subset  $W = U \cap V$ :

```
sage: for i in M.irange():
....:     for j in M.irange():
....:         for k in M.irange():
....:             nab.add_coef(eV)[i,j,k] = nab.coef(eVW)[i,j,k,c_uvW].expr()
....:
```

At this stage, the connection is fully defined on all the manifold:

```
sage: nab.coef(eU)[:]
[[[x, 0], [0, 0]], [[0, x*y], [0, 0]]]
sage: nab.coef(eV)[:]
[[[1/16*u^2 - 1/16*v^2 + 1/8*u + 1/8*v, -1/16*u^2 + 1/16*v^2 + 1/8*u + 1/8*v],
 [1/16*u^2 - 1/16*v^2 + 1/8*u + 1/8*v, -1/16*u^2 + 1/16*v^2 + 1/8*u + 1/8*v]],
 [[-1/16*u^2 + 1/16*v^2 + 1/8*u + 1/8*v, 1/16*u^2 - 1/16*v^2 + 1/8*u + 1/8*v],
 [-1/16*u^2 + 1/16*v^2 + 1/8*u + 1/8*v, 1/16*u^2 - 1/16*v^2 + 1/8*u + 1/8*v]]]
```

We may let it act on a vector field defined globally on  $M$ :

```
sage: a = M.vector_field('a')
sage: a[eU,:] = [-y, x]
sage: a[eV,0] = a[eVW,0,c_uvW].expr()
sage: a[eV,1] = a[eVW,1,c_uvW].expr()
sage: a.display(eU)
```

```

a = -y d/dx + x d/dy
sage: a.display(eV)
a = v d/du - u d/dv
sage: da = nab(a) ; da
Tensor field nabla(a) of type (1,1) on the 2-dimensional differentiable
manifold M
sage: da.display(eU)
nabla(a) = -x*y d/dx*dx - d/dx*dy + d/dy*dx - x*y^2 d/dy*dy
sage: da.display(eV)
nabla(a) = (-1/16*u^3 + 1/16*u^2*v + 1/16*(u + 2)*v^2 - 1/16*v^3 - 1/8*u^2) d/
↪du*du
+ (1/16*u^3 - 1/16*u^2*v - 1/16*(u - 2)*v^2 + 1/16*v^3 - 1/8*u^2 + 1) d/du*dv
+ (1/16*u^3 - 1/16*u^2*v - 1/16*(u - 2)*v^2 + 1/16*v^3 - 1/8*u^2 - 1) d/dv*du
+ (-1/16*u^3 + 1/16*u^2*v + 1/16*(u + 2)*v^2 - 1/16*v^3 - 1/8*u^2) d/dv*dv

```

A few tests:

```

sage: nab(a.restrict(V)) == da.restrict(V)
True
sage: nab.restrict(V)(a) == da.restrict(V)
True
sage: nab.restrict(V)(a.restrict(U)) == da.restrict(W)
True
sage: nab.restrict(U)(a.restrict(V)) == da.restrict(W)
True

```

#### `add_coef` (*frame=None*)

Return the connection coefficients in a given frame for assignment, keeping the coefficients in other frames.

See method `coef()` for details about the definition of the connection coefficients.

To delete the connection coefficients in other frames, use the method `set_coef()` instead.

INPUT:

- `frame` – (default: `None`) vector frame in which the connection coefficients are defined; if `None`, the default frame of the connection's domain is assumed.

**Warning:** If the connection has already coefficients in other frames, it is the user's responsibility to make sure that the coefficients to be added are consistent with them.

OUTPUT:

- connection coefficients in the given frame, as an instance of the class `Components`; if such connection coefficients did not exist previously, they are created. See method `coef()` for the storage convention of the connection coefficients.

EXAMPLES:

Setting the coefficients of an affine connection w.r.t. some coordinate frame:

```

sage: M = Manifold(2, 'M', start_index=1)
sage: X.<x,y> = M.chart()
sage: nab = M.affine_connection('nabla', latex_name=r'\nabla')
sage: eX = X.frame(); eX
Coordinate frame (M, (d/dx,d/dy))
sage: nab.add_coef(eX)
3-indices components w.r.t. Coordinate frame (M, (d/dx,d/dy))

```

```
sage: nab.add_coef(eX) [1,2,1] = x*y
sage: nab.display(eX)
Gam^x_yx = x*y
```

Since `eX` is the manifold's default vector frame, its mention may be omitted:

```
sage: nab.add_coef() [1,2,1] = x*y
sage: nab.add_coef()
3-indices components w.r.t. Coordinate frame (M, (d/dx,d/dy))
sage: nab.add_coef() [1,2,1] = x*y
sage: nab.display()
Gam^x_yx = x*y
```

Adding connection coefficients w.r.t. to another vector frame:

```
sage: e = M.vector_frame('e')
sage: nab.add_coef(e)
3-indices components w.r.t. Vector frame (M, (e_1,e_2))
sage: nab.add_coef(e) [2,1,1] = x+y
sage: nab.add_coef(e) [2,1,2] = x-y
sage: nab.display(e)
Gam^2_11 = x + y
Gam^2_12 = x - y
```

The coefficients w.r.t. the frame `eX` have been kept:

```
sage: nab.display(eX)
Gam^x_yx = x*y
```

To delete them, use the method `set_coef()` instead.

#### **coef** (*frame=None*)

Return the connection coefficients relative to the given frame.

$n$  being the manifold's dimension, the connection coefficients relative to the vector frame  $(e_i)$  are the  $n^3$  scalar fields  $\Gamma_{ij}^k$ , defined by

$$\nabla_{e_j} e_i = \Gamma_{ij}^k e_k$$

If the connection coefficients are not known already, they are computed from the above formula.

INPUT:

- `frame` – (default: `None`) vector frame relative to which the connection coefficients are required; if none is provided, the domain's default frame is assumed

OUTPUT:

- connection coefficients relative to the frame `frame`, as an instance of the class `Components` with 3 indices ordered as  $(k, i, j)$

EXAMPLES:

Connection coefficient of an affine connection on a 3-dimensional manifold:

```
sage: M = Manifold(3, 'M', start_index=1)
sage: c_xyz.<x,y,z> = M.chart()
sage: nab = M.affine_connection('nabla', r'\nabla')
sage: nab[1,1,2], nab[3,2,3] = x^2, y*z # Gamma^1_{12} = x^2, Gamma^3_{23} =
↪ yz
```

```

sage: nab.coef()
3-indices components w.r.t. Coordinate frame (M, (d/dx,d/dy,d/dz))
sage: type(nab.coef())
<class 'sage.tensor.modules.comp.Components'>
sage: M.default_frame()
Coordinate frame (M, (d/dx,d/dy,d/dz))
sage: nab.coef() is nab.coef(c_xyz.frame())
True
sage: nab.coef()[:] # full list of coefficients:
[[[0, x^2, 0], [0, 0, 0], [0, 0, 0]],
 [[0, 0, 0], [0, 0, 0], [0, 0, 0]],
 [[0, 0, 0], [0, 0, y*z], [0, 0, 0]]]

```

**connection\_form**(*i, j, frame=None*)

Return the connection 1-form corresponding to the given index and vector frame.

The connection 1-forms with respect to the frame  $(e_i)$  are the  $n^2$  1-forms  $\omega^i_j$  defined by

$$\nabla_v e_j = \langle \omega^i_j, v \rangle e_i$$

for any vector  $v$ .

The components of  $\omega^i_j$  in the coframe  $(e^i)$  dual to  $(e_i)$  are nothing but the connection coefficients  $\Gamma^i_{jk}$  relative to the frame  $(e_i)$ :

$$\omega^i_j = \Gamma^i_{jk} e^k$$

INPUT:

- $i, j$  – indices identifying the 1-form  $\omega^i_j$
- `frame` – (default: `None`) vector frame relative to which the connection 1-forms are defined; if `None`, the default frame of the connection's domain is assumed.

OUTPUT:

- the 1-form  $\omega^i_j$ , as an instance of `DiffForm`

EXAMPLES:

Connection 1-forms on a 3-dimensional manifold:

```

sage: M = Manifold(3, 'M', start_index=1)
sage: c_xyz.<x,y,z> = M.chart()
sage: nab = M.affine_connection('nabla', r'\nabla')
sage: nab[1,1,1], nab[1,1,2], nab[1,1,3] = x*y*z, x^2, -y*z
sage: nab[1,2,3], nab[1,3,1], nab[1,3,2] = -x^3, y^2*z, y^2-x^2
sage: nab[2,1,1], nab[2,1,2], nab[2,2,1] = z^2, x*y*z^2, -x^2
sage: nab[2,3,1], nab[2,3,3], nab[3,1,2] = x^2+y^2+z^2, y^2-z^2, x*y+z^2
sage: nab[3,2,1], nab[3,2,2], nab[3,3,3] = x*y+z, z^3 -y^2, x*z^2 - z*y^2
sage: nab.connection_form(1,1) # connection 1-form (i,j)=(1,1) w.r.t. M's
↳ default frame
1-form nabla connection 1-form (1,1) on the 3-dimensional
differentiable manifold M
sage: nab.connection_form(1,1)[:]
[x*y*z, x^2, -y*z]

```

The result is cached (until the connection is modified via `set_coef()` or `add_coef()`):

```
sage: nab.connection_form(1,1) is nab.connection_form(1,1)
True
```

Connection 1-forms w.r.t. a non-holonomic frame:

```
sage: ch_basis = M.automorphism_field()
sage: ch_basis[1,1], ch_basis[2,2], ch_basis[3,3] = y, z, x
sage: e = M.default_frame().new_frame(ch_basis, 'e')
sage: e[1][:], e[2][:], e[3][:]
([y, 0, 0], [0, z, 0], [0, 0, x])
sage: nab.connection_form(1,1,e)
1-form nabla connection 1-form (1,1) on the 3-dimensional
differentiable manifold M
sage: nab.connection_form(1,1,e).comp(e)[:]
[x*y^2*z, (x^2*y + 1)*z/y, -x*y*z]
```

Check of the formula  $\omega^i_j = \Gamma^i_{jk}e^k$ :

```
sage: #... on the manifold's default frame (d/dx, d/dy, d/dz)
sage: dx = M.default_frame().coframe() ; dx
Coordinate coframe (M, (dx,dy,dz))
sage: check = []
sage: for i in M.irange():
....:     for j in M.irange():
....:         check.append( nab.connection_form(i,j) == \
....:             sum( nab[[i,j,k]]*dx[k] for k in M.irange() ) )
....:
sage: check
[True, True, True, True, True, True, True, True, True]
sage: #... on the frame e
sage: ef = e.coframe() ; ef
Coframe (M, (e^1,e^2,e^3))
sage: check = []
sage: for i in M.irange():
....:     for j in M.irange():
....:         s = nab.connection_form(i,j,e).comp(c_xyz.frame(), from_basis=e)
....:         check.append( nab.connection_form(i,j,e) == sum( nab.coef(e)[[i,
↪j,k]]*ef[k] for k in M.irange() ) )
....:
sage: check
[True, True, True, True, True, True, True, True, True]
```

Check of the formula  $\nabla_v e_j = \langle \omega^i_j, v \rangle e_i$ :

```
sage: v = M.vector_field()
sage: v[:] = (x*y, z^2-3*x, z+2*y)
sage: b = M.default_frame()
sage: for j in M.irange(): # check on M's default frame
....:     nab(b[j]).contract(v) == \
....:         sum( nab.connection_form(i,j)(v)*b[i] for i in M.irange() )
True
True
True
sage: for j in M.irange(): # check on frame e
....:     nab(e[j]).contract(v) == \
....:         sum( nab.connection_form(i,j,e)(v)*e[i] for i in M.irange() )
True
True
```

```
True
```

**curvature\_form**(*i, j, frame=None*)

Return the curvature 2-form corresponding to the given index and vector frame.

The curvature 2-forms with respect to the frame  $(e_i)$  are the  $n^2$  2-forms  $\Omega^i_j$  defined by

$$\Omega^i_j(u, v) = R(e^i, e_j, u, v)$$

where  $R$  is the connection's Riemann curvature tensor (cf. `riemann()`),  $(e^i)$  is the coframe dual to  $(e_i)$  and  $(u, v)$  is a generic pair of vectors.

INPUT:

- *i, j* – indices identifying the 2-form  $\Omega^i_j$
- *frame* – (default: None) vector frame relative to which the curvature 2-forms are defined; if None, the default frame of the connection's domain is assumed.

OUTPUT:

- the 2-form  $\Omega^i_j$ , as an instance of `DiffForm`

EXAMPLES:

Curvature 2-forms on a 3-dimensional manifold:

```
sage: M = Manifold(3, 'M', start_index=1)
sage: c_xyz.<x,y,z> = M.chart()
sage: nab = M.affine_connection('nabla', r'\nabla')
sage: nab[1,1,1], nab[1,1,2], nab[1,1,3] = x*y*z, x^2, -y*z
sage: nab[1,2,3], nab[1,3,1], nab[1,3,2] = -x^3, y^2*z, y^2-x^2
sage: nab[2,1,1], nab[2,1,2], nab[2,2,1] = z^2, x*y*z^2, -x^2
sage: nab[2,3,1], nab[2,3,3], nab[3,1,2] = x^2+y^2+z^2, y^2-z^2, x*y+z^2
sage: nab[3,2,1], nab[3,2,2], nab[3,3,3] = x*y+z, z^3 -y^2, x*z^2 - z*y^2
sage: nab.curvature_form(1,1) # long time
2-form curvature (1,1) of connection nabla w.r.t. Coordinate frame
(M, (d/dx,d/dy,d/dz)) on the 3-dimensional differentiable manifold M
sage: nab.curvature_form(1,1).display() # long time (if above is skipped)
curvature (1,1) of connection nabla w.r.t. Coordinate frame
(M, (d/dx,d/dy,d/dz)) = (y^2*z^3 + (x*y^3 - x)*z + 2*x) dx/\dy
+ (x^3*z^2 - x*y) dx/\dz + (x^4*y*z^2 - z) dy/\dz
```

Curvature 2-forms w.r.t. a non-holonomic frame:

```
sage: ch_basis = M.automorphism_field()
sage: ch_basis[1,1], ch_basis[2,2], ch_basis[3,3] = y, z, x
sage: e = M.default_frame().new_frame(ch_basis, 'e')
sage: e[1].display(), e[2].display(), e[3].display()
(e_1 = y d/dx, e_2 = z d/dy, e_3 = x d/dz)
sage: ef = e.coframe()
sage: ef[1].display(), ef[2].display(), ef[3].display()
(e^1 = 1/y dx, e^2 = 1/z dy, e^3 = 1/x dz)
sage: nab.curvature_form(1,1,e) # long time
2-form curvature (1,1) of connection nabla w.r.t. Vector frame
(M, (e_1,e_2,e_3)) on the 3-dimensional differentiable manifold M
sage: nab.curvature_form(1,1,e).display(e) # long time (if above is skipped)
curvature (1,1) of connection nabla w.r.t. Vector frame
(M, (e_1,e_2,e_3)) =
(y^3*z^4 + 2*x*y*z + (x*y^4 - x*y)*z^2) e^1/\e^2
+ (x^4*y*z^2 - x^2*y^2) e^1/\e^3 + (x^5*y*z^3 - x*z^2) e^2/\e^3
```

Cartan's second structure equation is

$$\Omega^i_j = d\omega^i_j + \omega^i_k \wedge \omega^k_j$$

where the  $\omega^i_j$ 's are the connection 1-forms (cf. `connection_form()`). Let us check it on the frame `e`:

```
sage: omega = nab.connection_form
sage: check = []
sage: for i in M.irange(): # long time
....:     for j in M.irange():
....:         check.append( nab.curvature_form(i,j,e) == \
....:             omega(i,j,e).exterior_derivative() + \
....:             sum( omega(i,k,e).wedge(omega(k,j,e)) for k in M.irange()) )
....:
sage: check # long time
[True, True, True, True, True, True, True, True, True]
```

**del\_other\_coef** (*frame=None*)

Delete all the coefficients but those corresponding to `frame`.

INPUT:

- `frame` – (default: `None`) vector frame, the connection coefficients w.r.t. which are to be kept; if `None`, the default frame of the connection's domain is assumed.

EXAMPLES:

We first create two sets of connection coefficients:

```
sage: M = Manifold(2, 'M', start_index=1)
sage: X.<x,y> = M.chart()
sage: nab = M.affine_connection('nabla', latex_name=r'\nabla')
sage: eX = X.frame()
sage: nab.set_coef(eX)[1,2,1] = x*y
sage: e = M.vector_frame('e')
sage: nab.add_coef(e)[2,1,1] = x+y
sage: nab.display(eX)
Gam^x_yx = x*y
sage: nab.display(e)
Gam^2_11 = x + y
```

Let us delete the connection coefficients w.r.t. all frames except for frame `eX`:

```
sage: nab.del_other_coef(eX)
sage: nab.display(eX)
Gam^x_yx = x*y
```

The connection coefficients w.r.t. frame `e` have indeed been deleted:

```
sage: nab.display(e)
Traceback (most recent call last):
...
ValueError: no common frame found for the computation
```

**display** (*frame=None, chart=None, symbol=None, latex\_symbol=None, index\_labels=None, index\_latex\_labels=None, coordinate\_labels=True, only\_nonzero=True, only\_nonredundant=False*)

Display all the connection coefficients w.r.t. to a given frame, one per line.

The output is either text-formatted (console mode) or LaTeX-formatted (notebook mode).

## INPUT:

- `frame` – (default: `None`) vector frame relative to which the connection coefficients are defined; if `None`, the default frame of the connection’s domain is used
- `chart` – (default: `None`) chart specifying the coordinate expression of the connection coefficients; if `None`, the default chart of the connection’s domain is used
- `symbol` – (default: `None`) string specifying the symbol of the connection coefficients; if `None`, ‘`Gam`’ is used
- `latex_symbol` – (default: `None`) string specifying the LaTeX symbol for the components; if `None`, ‘`\Gamma`’ is used
- `index_labels` – (default: `None`) list of strings representing the labels of each index; if `None`, integer labels are used, except if `frame` is a coordinate frame and `coordinate_symbols` is set to `True`, in which case the coordinate symbols are used
- `index_latex_labels` – (default: `None`) list of strings representing the LaTeX labels of each index; if `None`, integer labels are used, except if `frame` is a coordinate frame and `coordinate_symbols` is set to `True`, in which case the coordinate LaTeX symbols are used
- `coordinate_labels` – (default: `True`) boolean; if `True`, coordinate symbols are used by default (instead of integers) as index labels whenever `frame` is a coordinate frame
- `only_nonzero` – (default: `True`) boolean; if `True`, only nonzero connection coefficients are displayed
- `only_nonredundant` – (default: `False`) boolean; if `True`, only nonredundant connection coefficients are displayed in case of symmetries

## EXAMPLES:

Coefficients of a connection on a 3-dimensional manifold:

```
sage: M = Manifold(3, 'M', start_index=1)
sage: c_xyz.<x,y,z> = M.chart()
sage: nab = M.affine_connection('nabla', r'\nabla')
sage: nab[1,1,2], nab[3,2,3] = x^2, y*z
```

By default, only the nonzero connection coefficients are displayed:

```
sage: nab.display()
Gam^x_xy = x^2
Gam^z_yz = y*z
sage: latex(nab.display())
\begin{array}{lcl} \Gamma_{\phantom{\}, x} \, \, x \, \, y \, \, \phantom{\}, x} \\ \leftrightarrow \phantom{\}, y} \} \\ & = & x^2 \, \, \\ \Gamma_{\phantom{\}, z} \, \, y \, \, z \, \, \phantom{\}, z} \\ & = & y z \end{array}
```

By default, the displayed connection coefficients are those w.r.t. to the default frame of the connection’s domain, so the above is equivalent to:

```
sage: nab.display(frame=M.default_frame())
Gam^x_xy = x^2
Gam^z_yz = y*z
```

Since the default frame is a coordinate frame, coordinate symbols are used to label the indices, but one may ask for integers instead:

```

sage: M.default_frame() is c_xyz.frame()
True
sage: nab.display(coordinate_labels=False)
Gam^1_12 = x^2
Gam^3_23 = y*z

```

The index labels can also be customized:

```

sage: nab.display(index_labels=['(1)', '(2)', '(3)'])
Gam^(1)_(1), (2) = x^2
Gam^(3)_(2), (3) = y*z

```

The symbol ‘Gam’ can be changed:

```

sage: nab.display(symbol='C', latex_symbol='C')
C^x_xy = x^2
C^z_yz = y*z
sage: latex(nab.display(symbol='C', latex_symbol='C'))
\begin{array}{lcl} C_{\phantom{\,}, x} \, , x \, , y \}^{\, , x \phantom{\,}, x} \\ \rightarrow \phantom{\,}, y \} \\ & = & x^2 \\ C_{\phantom{\,}, z} \, , y \, , z \}^{\, , z \phantom{\,}, y} \phantom{\,}, z \} \\ & = & y z \end{array}

```

Display of Christoffel symbols, skipping the redundancy associated with the symmetry of the last two indices:

```

sage: M = Manifold(3, 'R^3', start_index=1)
sage: c_spher.<r,th,ph> = M.chart(r'r:(0,+oo) th:(0,pi):\theta ph:(0,
\rightarrow 2*pi):\phi')
sage: g = M.metric('g')
sage: g[1,1], g[2,2], g[3,3] = 1, r^2, (r*sin(th))^2
sage: g.display()
g = dr*dr + r^2 dth*dth + r^2*sin(th)^2 dph*dph
sage: g.connection().display(only_nonredundant=True)
Gam^r_th,th = -r
Gam^r_ph,ph = -r*sin(th)^2
Gam^th_r,th = 1/r
Gam^th_ph,ph = -cos(th)*sin(th)
Gam^ph_r,ph = 1/r
Gam^ph_th,ph = cos(th)/sin(th)

```

By default, the parameter `only_nonredundant` is set to `False`:

```

sage: g.connection().display()
Gam^r_th,th = -r
Gam^r_ph,ph = -r*sin(th)^2
Gam^th_r,th = 1/r
Gam^th_th,r = 1/r
Gam^th_ph,ph = -cos(th)*sin(th)
Gam^ph_r,ph = 1/r
Gam^ph_th,ph = cos(th)/sin(th)
Gam^ph_ph,r = 1/r
Gam^ph_ph,th = cos(th)/sin(th)

```

**domain()**

Return the manifold subset on which the affine connection is defined.

OUTPUT:

- instance of class *DifferentiableManifold* representing the manifold on which *self* is defined.

EXAMPLES:

```
sage: M = Manifold(3, 'M', start_index=1)
sage: c_xyz.<x,y,z> = M.chart()
sage: nab = M.affine_connection('nabla', r'\nabla')
sage: nab.domain()
3-dimensional differentiable manifold M
sage: U = M.open_subset('U', coord_def={c_xyz: x>0})
sage: nabU = U.affine_connection('D')
sage: nabU.domain()
Open subset U of the 3-dimensional differentiable manifold M
```

**restrict** (*subdomain*)

Return the restriction of the connection to some subdomain.

If such restriction has not been defined yet, it is constructed here.

INPUT:

- subdomain – open subset  $U$  of the connection's domain (must be an instance of *DifferentiableManifold*)

OUTPUT:

- instance of *AffineConnection* representing the restriction.

EXAMPLES:

Restriction of a connection on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', start_index=1)
sage: c_xy.<x,y> = M.chart()
sage: nab = M.affine_connection('nabla', r'\nabla')
sage: nab[1,1,2], nab[2,1,1] = x^2, x+y
sage: nab[:]
[[[0, x^2], [0, 0]], [[x + y, 0], [0, 0]]]
sage: U = M.open_subset('U', coord_def={c_xy: x>0})
sage: nabU = nab.restrict(U) ; nabU
Affine connection nabla on the Open subset U of the 2-dimensional
differentiable manifold M
sage: nabU.domain()
Open subset U of the 2-dimensional differentiable manifold M
sage: nabU[:]
[[[0, x^2], [0, 0]], [[x + y, 0], [0, 0]]]
```

The result is cached:

```
sage: nab.restrict(U) is nabU
True
```

until the connection is modified:

```
sage: nab[1,2,2] = -y
sage: nab.restrict(U) is nabU
False
sage: nab.restrict(U)[:]
[[[0, x^2], [0, -y]], [[x + y, 0], [0, 0]]]
```

**ricci()**

Return the connection's Ricci tensor.

The *Ricci tensor* is the tensor field  $Ric$  of type (0,2) defined from the Riemann curvature tensor  $R$  by

$$Ric(u, v) = R(e^i, u, e_i, v)$$

for any vector fields  $u$  and  $v$ ,  $(e_i)$  being any vector frame and  $(e^i)$  the dual coframe.

OUTPUT:

- the Ricci tensor  $Ric$ , as an instance of *TensorField*

EXAMPLES:

Ricci tensor of an affine connection on a 3-dimensional manifold:

```
sage: M = Manifold(3, 'M', start_index=1)
sage: c_xyz.<x,y,z> = M.chart()
sage: nab = M.affine_connection('nabla', r'\nabla') ; nab
Affine connection nabla on the 3-dimensional differentiable
manifold M
sage: nab[1,1,2], nab[3,2,3] = x^2, y*z # Gamma^1_{12} = x^2, Gamma^3_{23} =
↪yz
sage: r = nab.ricci() ; r
Tensor field of type (0,2) on the 3-dimensional differentiable
manifold M
sage: r[:]
[ 0 2*x  0]
[ 0  -z  0]
[ 0  0  0]
```

The result is cached (until the connection is modified via *set\_coef()* or *add\_coef()*):

```
sage: nab.ricci() is r
True
```

**riemann()**

Return the connection's Riemann curvature tensor.

The *Riemann curvature tensor* is the tensor field  $R$  of type (1,3) defined by

$$R(\omega, w, u, v) = \langle \omega, \nabla_u \nabla_v w - \nabla_v \nabla_u w - \nabla_{[u,v]} w \rangle$$

for any 1-form  $\omega$  and any vector fields  $u, v$  and  $w$ .

OUTPUT:

- the Riemann curvature tensor  $R$ , as an instance of *TensorField*

EXAMPLES:

Curvature of an affine connection on a 3-dimensional manifold:

```
sage: M = Manifold(3, 'M', start_index=1)
sage: c_xyz.<x,y,z> = M.chart()
sage: nab = M.affine_connection('nabla', r'\nabla') ; nab
Affine connection nabla on the 3-dimensional differentiable
manifold M
sage: nab[1,1,2], nab[3,2,3] = x^2, y*z # Gamma^1_{12} = x^2, Gamma^3_{23} =
↪yz
```

```

sage: r = nab.riemann() ; r
Tensor field of type (1,3) on the 3-dimensional differentiable
manifold M
sage: r.parent()
Free module T^(1,3)(M) of type-(1,3) tensors fields on the
3-dimensional differentiable manifold M

```

By construction, the Riemann tensor is antisymmetric with respect to its last two arguments (denoted  $u$  and  $v$  in the definition above), which are at positions 2 and 3 (the first argument being at position 0):

```

sage: r.symmetries()
no symmetry; antisymmetry: (2, 3)

```

The components:

```

sage: r[:]
[[[0, 2*x, 0], [-2*x, 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, 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, 0, 0], [0, 0, 0]],
 [[0, 0, 0], [0, 0, 0], [0, 0, 0]],
 [[0, 0, 0], [0, 0, z], [0, -z, 0]],
 [[0, 0, 0], [0, 0, 0], [0, 0, 0]]]

```

The result is cached (until the connection is modified via `set_coef()` or `add_coef()`):

```

sage: nab.riemann() is r
True

```

Another example: Riemann curvature tensor of some connection on a non-parallelizable 2-dimensional manifold:

```

sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U') ; V = M.open_subset('V')
sage: M.declare_union(U,V) # M is the union of U and V
sage: c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart()
sage: transf = c_xy.transition_map(c_uv, (x+y, x-y), intersection_name='W',
....:                               restrictions1= x>0, restrictions2= u+v>0)
sage: inv = transf.inverse()
sage: W = U.intersection(V)
sage: eU = c_xy.frame() ; eV = c_uv.frame()
sage: c_xyW = c_xy.restrict(W) ; c_uvW = c_uv.restrict(W)
sage: eUW = c_xyW.frame() ; eVW = c_uvW.frame()
sage: nab = M.affine_connection('nabla', r'\nabla')
sage: nab[0,0,0], nab[0,1,0], nab[1,0,1] = x, x-y, x*y
sage: for i in M.irange():
....:     for j in M.irange():
....:         for k in M.irange():
....:             nab.add_coef(eV)[i,j,k] = nab.coef(eVW)[i,j,k,c_uvW].expr()
....:
sage: r = nab.riemann() ; r
Tensor field of type (1,3) on the 2-dimensional differentiable
manifold M
sage: r.parent()
Module T^(1,3)(M) of type-(1,3) tensors fields on the 2-dimensional
differentiable manifold M

```

```

sage: r.display(eU)
(x^2*y - x*y^2) d/dx*dx*dx*dy + (-x^2*y + x*y^2) d/dx*dx*dy*dx + d/dx*dy*dx*dy
- d/dx*dy*dy*dx - (x^2 - 1)*y d/dy*dx*dx*dy + (x^2 - 1)*y d/dy*dx*dy*dx
+ (-x^2*y + x*y^2) d/dy*dy*dx*dy + (x^2*y - x*y^2) d/dy*dy*dy*dx
sage: r.display(eV)
(1/32*u^3 - 1/32*u*v^2 - 1/32*v^3 + 1/32*(u^2 + 4)*v - 1/8*u - 1/4) d/
↪du*du*du*dv
+ (-1/32*u^3 + 1/32*u*v^2 + 1/32*v^3 - 1/32*(u^2 + 4)*v + 1/8*u + 1/4) d/
↪du*du*dv*du
+ (1/32*u^3 - 1/32*u*v^2 + 3/32*v^3 - 1/32*(3*u^2 - 4)*v - 1/8*u + 1/4) d/
↪du*dv*du*dv
+ (-1/32*u^3 + 1/32*u*v^2 - 3/32*v^3 + 1/32*(3*u^2 - 4)*v + 1/8*u - 1/4) d/
↪du*dv*dv*du
+ (-1/32*u^3 + 1/32*u*v^2 + 5/32*v^3 - 1/32*(5*u^2 + 4)*v + 1/8*u - 1/4) d/
↪dv*du*du*dv
+ (1/32*u^3 - 1/32*u*v^2 - 5/32*v^3 + 1/32*(5*u^2 + 4)*v - 1/8*u + 1/4) d/
↪dv*du*dv*du
+ (-1/32*u^3 + 1/32*u*v^2 + 1/32*v^3 - 1/32*(u^2 + 4)*v + 1/8*u + 1/4) d/
↪dv*dv*du*dv
+ (1/32*u^3 - 1/32*u*v^2 - 1/32*v^3 + 1/32*(u^2 + 4)*v - 1/8*u - 1/4) d/
↪dv*dv*dv*du

```

The same computation parallelized on 2 cores:

```

sage: Parallelism().set(nproc=2)
sage: r_backup = r
sage: nab = M.affine_connection('nabla', r'\nabla')
sage: nab[0,0,0], nab[0,1,0], nab[1,0,1] = x, x-y, x*y
sage: for i in M.irange():
....:     for j in M.irange():
....:         for k in M.irange():
....:             nab.add_coef(eV)[i,j,k] = nab.coef(eVW)[i,j,k,c_uvW].expr()
....:
sage: r = nab.riemann(); r
Tensor field of type (1,3) on the 2-dimensional differentiable
manifold M
sage: r.parent()
Module T^(1,3)(M) of type-(1,3) tensors fields on the 2-dimensional
differentiable manifold M
sage: r == r_backup
True
sage: Parallelism().set(nproc=1) # switch off parallelization

```

### set\_coef (frame=None)

Return the connection coefficients in a given frame for assignment.

See method `coef()` for details about the definition of the connection coefficients.

The connection coefficients with respect to other frames are deleted, in order to avoid any inconsistency. To keep them, use the method `add_coef()` instead.

INPUT:

- `frame` – (default: None) vector frame in which the connection coefficients are defined; if None, the default frame of the connection's domain is assumed.

OUTPUT:

- connection coefficients in the given frame, as an instance of the class `Components`; if such connection coefficients did not exist previously, they are created. See method `coef()` for the storage

convention of the connection coefficients.

#### EXAMPLES:

Setting the coefficients of an affine connection w.r.t. some coordinate frame:

```
sage: M = Manifold(2, 'M', start_index=1)
sage: X.<x,y> = M.chart()
sage: nab = M.affine_connection('nabla', latex_name=r'\nabla')
sage: eX = X.frame(); eX
Coordinate frame (M, (d/dx,d/dy))
sage: nab.set_coef(eX)
3-indices components w.r.t. Coordinate frame (M, (d/dx,d/dy))
sage: nab.set_coef(eX)[1,2,1] = x*y
sage: nab.display(eX)
Gam^x_yx = x*y
```

Since `eX` is the manifold's default vector frame, its mention may be omitted:

```
sage: nab.set_coef()[1,2,1] = x*y
sage: nab.set_coef()
3-indices components w.r.t. Coordinate frame (M, (d/dx,d/dy))
sage: nab.set_coef()[1,2,1] = x*y
sage: nab.display()
Gam^x_yx = x*y
```

To set the coefficients in the default frame, one can even bypass the method `set_coef()` and call directly the operator `[]` on the connection object:

```
sage: nab[1,2,1] = x*y
sage: nab.display()
Gam^x_yx = x*y
```

Setting the connection coefficients w.r.t. to another vector frame:

```
sage: e = M.vector_frame('e')
sage: nab.set_coef(e)
3-indices components w.r.t. Vector frame (M, (e_1,e_2))
sage: nab.set_coef(e)[2,1,1] = x+y
sage: nab.set_coef(e)[2,1,2] = x-y
sage: nab.display(e)
Gam^2_11 = x + y
Gam^2_12 = x - y
```

The coefficients w.r.t. the frame `eX` have been deleted:

```
sage: nab.display(eX)
Traceback (most recent call last):
...
ValueError: no common frame found for the computation
```

To keep them, use the method `add_coef()` instead.

#### **torsion()**

Return the connection's torsion tensor.

The torsion tensor is the tensor field  $T$  of type (1,2) defined by

$$T(\omega, u, v) = \langle \omega, \nabla_u v - \nabla_v u - [u, v] \rangle$$

for any 1-form  $\omega$  and any vector fields  $u$  and  $v$ .

OUTPUT:

- the torsion tensor  $T$ , as an instance of *TensorField*

EXAMPLES:

Torsion of an affine connection on a 3-dimensional manifold:

```
sage: M = Manifold(3, 'M', start_index=1)
sage: c_xyz.<x,y,z> = M.chart()
sage: nab = M.affine_connection('nabla', r'\nabla')
sage: nab[1,1,2], nab[3,2,3] = x^2, y*z # Gamma^1_{12} = x^2, Gamma^3_{23} = yz
sage: t = nab.torsion() ; t
Tensor field of type (1,2) on the 3-dimensional differentiable
manifold M
sage: t.symmetries()
no symmetry; antisymmetry: (1, 2)
sage: t[:]
[[[0, -x^2, 0], [x^2, 0, 0], [0, 0, 0]],
 [[0, 0, 0], [0, 0, 0], [0, 0, 0]],
 [[0, 0, 0], [0, 0, -y*z], [0, y*z, 0]]]
```

The torsion expresses the lack of commutativity of two successive derivatives of a scalar field:

```
sage: f = M.scalar_field(x*z^2 + y^2 - z^2, name='f')
sage: DDf = nab(nab(f)) ; DDf
Tensor field nabla(df) of type (0,2) on the 3-dimensional
differentiable manifold M
sage: DDf.antisymmetrize()[:] # two successive derivatives do not commute:
[
  0 -1/2*x^2*z^2 0
[ 1/2*x^2*z^2 0 -(x - 1)*y*z^2
[ 0 (x - 1)*y*z^2 0
sage: 2*DDf.antisymmetrize() == nab.torsion().contract(0,nab(f))
True
```

The above identity is the standard formula

$$\nabla_j \nabla_i f - \nabla_i \nabla_j f = T^k_{ij} \nabla_k f,$$

where the  $T^k_{ij}$ 's are the components of the torsion tensor.

The result is cached:

```
sage: nab.torsion() is t
True
```

as long as the connection remains unchanged:

```
sage: nab[2,1,3] = 1+x # changing the connection
sage: nab.torsion() is t # a new computation of the torsion has been made
False
sage: (nab.torsion() - t).display()
(-x - 1) d/dy*dx*dz + (x + 1) d/dy*dz*dx
```

Another example: torsion of some connection on a non-parallelizable 2-dimensional manifold:

```

sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U') ; V = M.open_subset('V')
sage: M.declare_union(U,V) # M is the union of U and V
sage: c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart()
sage: transf = c_xy.transition_map(c_uv, (x+y, x-y), intersection_name='W',
....:                               restrictions1= x>0, restrictions2= u+v>0)
sage: inv = transf.inverse()
sage: W = U.intersection(V)
sage: eU = c_xy.frame() ; eV = c_uv.frame()
sage: c_xyW = c_xy.restrict(W) ; c_uvW = c_uv.restrict(W)
sage: eUW = c_xyW.frame() ; eVW = c_uvW.frame()
sage: nab = M.affine_connection('nabla', r'\nabla')
sage: nab[0,0,0], nab[0,1,0], nab[1,0,1] = x, x-y, x*y
sage: for i in M.irange():
....:     for j in M.irange():
....:         for k in M.irange():
....:             nab.add_coef(eV)[i,j,k] = nab.coef(eVW)[i,j,k,c_uvW].expr()
....:
sage: t = nab.torsion() ; t
Tensor field of type (1,2) on the 2-dimensional differentiable
manifold M
sage: t.parent()
Module T^(1,2)(M) of type-(1,2) tensors fields on the 2-dimensional
differentiable manifold M
sage: t[eU,:]
[[[0, x - y], [-x + y, 0]], [[0, -x*y], [x*y, 0]]]
sage: t[eV,:]
[[[0, 1/8*u^2 - 1/8*v^2 - 1/2*v], [-1/8*u^2 + 1/8*v^2 + 1/2*v, 0]],
 [[0, -1/8*u^2 + 1/8*v^2 - 1/2*v], [1/8*u^2 - 1/8*v^2 + 1/2*v, 0]]]

```

Check of the torsion formula:

```

sage: f = M.scalar_field({c_xy: (x+y)^2, c_uv: u^2}, name='f')
sage: DDf = nab(nab(f)) ; DDf
Tensor field nabla(df) of type (0,2) on the 2-dimensional
differentiable manifold M
sage: DDf.antisymmetrize().display(eU)
(-x^2*y - (x + 1)*y^2 + x^2) dx/\dy
sage: DDf.antisymmetrize().display(eV)
(1/8*u^3 - 1/8*u*v^2 - 1/2*u*v) du/\dv
sage: 2*DDf.antisymmetrize() == nab(f).contract(nab.torsion())
True

```

**torsion\_form**(*i*, *frame=None*)

Return the torsion 2-form corresponding to the given index and vector frame.

The *torsion 2-forms* with respect to the frame  $(e_i)$  are the  $n$  2-forms  $\theta^i$  defined by

$$\theta^i(u, v) = T(e^i, u, v)$$

where  $T$  is the connection's torsion tensor (cf. `torsion()`),  $(e^i)$  is the coframe dual to  $(e_i)$  and  $(u, v)$  is a generic pair of vectors.

INPUT:

- *i* – index identifying the 2-form  $\theta^i$
- *frame* – (default: `None`) vector frame relative to which the torsion 2-forms are defined; if `None`, the default frame of the connection's domain is assumed.

OUTPUT:

- the 2-form  $\theta^i$ , as an instance of *DiffForm*

EXAMPLES:

Torsion 2-forms on a 3-dimensional manifold:

```
sage: M = Manifold(3, 'M', start_index=1)
sage: c_xyz.<x,y,z> = M.chart()
sage: nab = M.affine_connection('nabla', r'\nabla')
sage: nab[1,1,1], nab[1,1,2], nab[1,1,3] = x*y*z, x^2, -y*z
sage: nab[1,2,3], nab[1,3,1], nab[1,3,2] = -x^3, y^2*z, y^2-x^2
sage: nab[2,1,1], nab[2,1,2], nab[2,2,1] = z^2, x*y*z^2, -x^2
sage: nab[2,3,1], nab[2,3,3], nab[3,1,2] = x^2+y^2+z^2, y^2-z^2, x*y+z^2
sage: nab[3,2,1], nab[3,2,2], nab[3,3,3] = x*y+z, z^3 -y^2, x*z^2 - z*y^2
sage: nab.torsion_form(1)
2-form torsion (1) of connection nabla w.r.t. Coordinate frame
(M, (d/dx,d/dy,d/dz)) on the 3-dimensional differentiable manifold M
sage: nab.torsion_form(1)[: ]
[
  0          -x^2      (y^2 + y)*z]
[
  x^2          0      x^3 - x^2 + y^2]
[
  -(y^2 + y)*z -x^3 + x^2 - y^2      0]
```

Torsion 2-forms w.r.t. a non-holonomic frame:

```
sage: ch_basis = M.automorphism_field()
sage: ch_basis[1,1], ch_basis[2,2], ch_basis[3,3] = y, z, x
sage: e = M.default_frame().new_frame(ch_basis, 'e')
sage: e[1][: ], e[2][: ], e[3][: ]
([y, 0, 0], [0, z, 0], [0, 0, x])
sage: ef = e.coframe()
sage: ef[1][: ], ef[2][: ], ef[3][: ]
([1/y, 0, 0], [0, 1/z, 0], [0, 0, 1/x])
sage: nab.torsion_form(1, e)
2-form torsion (1) of connection nabla w.r.t. Vector frame
(M, (e_1,e_2,e_3)) on the 3-dimensional differentiable manifold M
sage: nab.torsion_form(1, e).comp(e)[: ]
[
  0          -x^2*z      (x*y^2 + x*y)*z]
[
  x^2*z          0      (x^4 - x^3 + x*y^2)*z/y]
[
  -(x*y^2 + x*y)*z -(x^4 - x^3 + x*y^2)*z/y      0]
```

Cartan's first structure equation is

$$\theta^i = de^i + \omega^i_j \wedge e^j$$

where the  $\omega^i_j$ 's are the connection 1-forms (cf. *connection\_form()*). Let us check it on the frame e:

```
sage: for i in M.irange(): # long time
....:     nab.torsion_form(i, e) == ef[i].exterior_derivative() + \
....:         sum(nab.connection_form(i,j,e).wedge(ef[j]) for j in M.irange())
....:
True
True
True
```

## PSEUDO-RIEMANNIAN MANIFOLDS

### 3.1 Pseudo-Riemannian Metrics

The class `PseudoRiemannianMetric` implements pseudo-Riemannian metrics on differentiable manifolds over  $\mathbf{R}$ . The derived class `PseudoRiemannianMetricParal` is devoted to metrics with values on a parallelizable manifold.

AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2013-2015) : initial version
- Pablo Angulo (2016): Schouten, Cotton and Cotton-York tensors

REFERENCES:

- [KN1963]
- [Lee1997]
- [ONe1983]

**class** `sage.manifolds.differentiable.metric.PseudoRiemannianMetric` (*vector\_field\_module*,  
*name*, *signature=None*, *latex\_name=None*)

Bases: `sage.manifolds.differentiable.tensorfield.TensorField`

Pseudo-Riemannian metric with values on an open subset of a differentiable manifold.

An instance of this class is a field of nondegenerate symmetric bilinear forms (metric field) along a differentiable manifold  $U$  with values on a differentiable manifold  $M$  over  $\mathbf{R}$ , via a differentiable mapping  $\Phi : U \rightarrow M$ . The standard case of a metric field on a manifold corresponds to  $U = M$  and  $\Phi = \text{Id}_M$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $M$  ( $U$  is then an open interval of  $\mathbf{R}$ ).

A metric  $g$  is a field on  $U$ , such that at each point  $p \in U$ ,  $g(p)$  is a bilinear map of the type:

$$g(p) : T_q M \times T_q M \longrightarrow \mathbf{R}$$

where  $T_q M$  stands for the tangent space to the manifold  $M$  at the point  $q = \Phi(p)$ , such that  $g(p)$  is symmetric:  $\forall (u, v) \in T_q M \times T_q M$ ,  $g(p)(v, u) = g(p)(u, v)$  and nondegenerate:  $(\forall v \in T_q M, g(p)(u, v) = 0) \implies u = 0$ .

---

**Note:** If  $M$  is parallelizable, the class `PseudoRiemannianMetricParal` should be used instead.

---

INPUT:

- `vector_field_module` – module  $\mathcal{X}(U, \Phi)$  of vector fields along  $U$  with values on  $\Phi(U) \subset M$

- `name` – name given to the metric
- `signature` – (default: None) signature  $S$  of the metric as a single integer:  $S = n_+ - n_-$ , where  $n_+$  (resp.  $n_-$ ) is the number of positive terms (resp. number of negative terms) in any diagonal writing of the metric components; if `signature` is None,  $S$  is set to the dimension of manifold  $M$  (Riemannian signature)
- `latex_name` – (default: None) LaTeX symbol to denote the metric; if None, it is formed from `name`

## EXAMPLES:

Standard metric on the sphere  $S^2$ :

```
sage: M = Manifold(2, 'S^2', start_index=1)
sage: # The two open domains covered by stereographic coordinates (North and
↪South):
sage: U = M.open_subset('U') ; V = M.open_subset('V')
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart() # stereographic coord
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                                     intersection_name='W', restrictions1= x^2+y^2!=0,
....:                                     restrictions2= u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: W = U.intersection(V) # The complement of the two poles
sage: eU = c_xy.frame() ; eV = c_uv.frame()
sage: c_xyW = c_xy.restrict(W) ; c_uvW = c_uv.restrict(W)
sage: eUW = c_xyW.frame() ; eVW = c_uvW.frame()
sage: g = M.metric('g') ; g
Riemannian metric g on the 2-dimensional differentiable manifold S^2
```

The metric is considered as a tensor field of type (0,2) on  $S^2$ :

```
sage: g.parent()
Module T^(0,2)(S^2) of type-(0,2) tensors fields on the 2-dimensional
differentiable manifold S^2
```

We define  $g$  by its components on domain  $U$  (factorizing them to have a nicer view):

```
sage: g[eU,1,1], g[eU,2,2] = 4/(1+x^2+y^2)^2, 4/(1+x^2+y^2)^2
sage: g.display(eU)
g = 4/(x^2 + y^2 + 1)^2 dx*dx + 4/(x^2 + y^2 + 1)^2 dy*dy
```

A matrix view of the components:

```
sage: g[eU,:]
[4/(x^2 + y^2 + 1)^2      0]
[      0 4/(x^2 + y^2 + 1)^2]
```

The components of  $g$  on domain  $V$  expressed in terms of  $(u,v)$  coordinates are similar to those on domain  $U$  expressed in  $(x,y)$  coordinates, as we can check explicitly by asking for the component transformation on the common subdomain  $W$ :

```
sage: g.display(eVW, c_uvW)
g = 4/(u^4 + v^4 + 2*(u^2 + 1)*v^2 + 2*u^2 + 1) du*du
+ 4/(u^4 + v^4 + 2*(u^2 + 1)*v^2 + 2*u^2 + 1) dv*dv
```

Therefore, we set:

```

sage: g[eV,1,1], g[eV,2,2] = 4/(1+u^2+v^2)^2, 4/(1+u^2+v^2)^2
sage: g[eV,1,1].factor() ; g[eV,2,2].factor()
4/(u^2 + v^2 + 1)^2
4/(u^2 + v^2 + 1)^2
sage: g.display(eV)
g = 4/(u^2 + v^2 + 1)^2 du*du + 4/(u^2 + v^2 + 1)^2 dv*dv

```

At this stage, the metric is fully defined on the whole sphere. Its restriction to some subdomain is itself a metric (by default, it bears the same symbol):

```

sage: g.restrict(U)
Riemannian metric g on the Open subset U of the 2-dimensional
differentiable manifold S^2
sage: g.restrict(U).parent()
Free module T^(0,2)(U) of type-(0,2) tensors fields on the Open subset
U of the 2-dimensional differentiable manifold S^2

```

The parent of  $g|_U$  is a free module because  $U$  is a parallelizable domain, contrary to  $S^2$ . Actually,  $g$  and  $g|_U$  have different Python type:

```

sage: type(g)
<class 'sage.manifolds.differentiable.metric.PseudoRiemannianMetric'>
sage: type(g.restrict(U))
<class 'sage.manifolds.differentiable.metric.PseudoRiemannianMetricParal'>

```

As a field of bilinear forms, the metric acts on pairs of tensor fields, yielding a scalar field:

```

sage: a = M.vector_field('a')
sage: a[eU,:] = [x, 2+y]
sage: a.add_comp_by_continuation(eV, W, chart=c_uv)
sage: b = M.vector_field('b')
sage: b[eU,:] = [-y, x]
sage: b.add_comp_by_continuation(eV, W, chart=c_uv)
sage: s = g(a,b) ; s
Scalar field g(a,b) on the 2-dimensional differentiable manifold S^2
sage: s.display()
g(a,b): S^2 --> R
on U: (x, y) |--> 8*x/(x^4 + y^4 + 2*(x^2 + 1)*y^2 + 2*x^2 + 1)
on V: (u, v) |--> 8*(u^3 + u*v^2)/(u^4 + v^4 + 2*(u^2 + 1)*v^2 + 2*u^2 + 1)

```

The inverse metric is:

```

sage: ginv = g.inverse() ; ginv
Tensor field inv_g of type (2,0) on the 2-dimensional differentiable
manifold S^2
sage: ginv.parent()
Module T^(2,0)(S^2) of type-(2,0) tensors fields on the 2-dimensional
differentiable manifold S^2
sage: latex(ginv)
g^{-1}
sage: ginv.display(eU) # again the components are expanded
inv_g = (1/4*x^4 + 1/4*y^4 + 1/2*(x^2 + 1)*y^2 + 1/2*x^2 + 1/4) d/dx*d/dx
+ (1/4*x^4 + 1/4*y^4 + 1/2*(x^2 + 1)*y^2 + 1/2*x^2 + 1/4) d/dy*d/dy
sage: ginv.display(eV)
inv_g = (1/4*u^4 + 1/4*v^4 + 1/2*(u^2 + 1)*v^2 + 1/2*u^2 + 1/4) d/du*d/du
+ (1/4*u^4 + 1/4*v^4 + 1/2*(u^2 + 1)*v^2 + 1/2*u^2 + 1/4) d/dv*d/dv

```

We have:

```

sage: ginv.restrict(U) is g.restrict(U).inverse()
True
sage: ginv.restrict(V) is g.restrict(V).inverse()
True
sage: ginv.restrict(W) is g.restrict(W).inverse()
True

```

The volume form (Levi-Civita tensor) associated with  $g$ :

```

sage: eps = g.volume_form() ; eps
2-form eps_g on the 2-dimensional differentiable manifold S^2
sage: eps.display(eU)
eps_g = 4/(x^4 + y^4 + 2*(x^2 + 1)*y^2 + 2*x^2 + 1) dx/\dy
sage: eps.display(eV)
eps_g = 4/(u^4 + v^4 + 2*(u^2 + 1)*v^2 + 2*u^2 + 1) du/\dv

```

The unique non-trivial component of the volume form is nothing but the square root of the determinant of  $g$  in the corresponding frame:

```

sage: eps[[eU,1,2]] == g.sqrt_abs_det(eU)
True
sage: eps[[eV,1,2]] == g.sqrt_abs_det(eV)
True

```

The Levi-Civita connection associated with the metric  $g$ :

```

sage: nabla = g.connection() ; nabla
Levi-Civita connection nabla_g associated with the Riemannian metric g
on the 2-dimensional differentiable manifold S^2
sage: latex(nabla)
\nabla_{g}

```

The Christoffel symbols  $\Gamma^i_{jk}$  associated with some coordinates:

```

sage: g.christoffel_symbols(c_xy)
3-indices components w.r.t. Coordinate frame (U, (d/dx,d/dy)), with
symmetry on the index positions (1, 2)
sage: g.christoffel_symbols(c_xy)[:]
[[[-2*x/(x^2 + y^2 + 1), -2*y/(x^2 + y^2 + 1)],
 [-2*y/(x^2 + y^2 + 1), 2*x/(x^2 + y^2 + 1)]],
 [[2*y/(x^2 + y^2 + 1), -2*x/(x^2 + y^2 + 1)],
 [-2*x/(x^2 + y^2 + 1), -2*y/(x^2 + y^2 + 1)]]]
sage: g.christoffel_symbols(c_uv)[:]
[[[-2*u/(u^2 + v^2 + 1), -2*v/(u^2 + v^2 + 1)],
 [-2*v/(u^2 + v^2 + 1), 2*u/(u^2 + v^2 + 1)]],
 [[2*v/(u^2 + v^2 + 1), -2*u/(u^2 + v^2 + 1)],
 [-2*u/(u^2 + v^2 + 1), -2*v/(u^2 + v^2 + 1)]]]

```

The Christoffel symbols are nothing but the connection coefficients w.r.t. the coordinate frame:

```

sage: g.christoffel_symbols(c_xy) is nabla.coef(c_xy.frame())
True
sage: g.christoffel_symbols(c_uv) is nabla.coef(c_uv.frame())
True

```

Test that  $\nabla$  is the connection compatible with  $g$ :

```

sage: t = nabla(g) ; t
Tensor field nabla_g(g) of type (0,3) on the 2-dimensional
differentiable manifold S^2
sage: t.display(eU)
nabla_g(g) = 0
sage: t.display(eV)
nabla_g(g) = 0
sage: t == 0
True

```

The Riemann curvature tensor of  $g$ :

```

sage: riem = g.riemann() ; riem
Tensor field Riem(g) of type (1,3) on the 2-dimensional differentiable
manifold S^2
sage: riem.display(eU)
Riem(g) = 4/(x^4 + y^4 + 2*(x^2 + 1)*y^2 + 2*x^2 + 1) d/dx*dy*dx*dy
- 4/(x^4 + y^4 + 2*(x^2 + 1)*y^2 + 2*x^2 + 1) d/dx*dy*dy*dx
- 4/(x^4 + y^4 + 2*(x^2 + 1)*y^2 + 2*x^2 + 1) d/dy*dx*dx*dy
+ 4/(x^4 + y^4 + 2*(x^2 + 1)*y^2 + 2*x^2 + 1) d/dy*dx*dy*dx
sage: riem.display(eV)
Riem(g) = 4/(u^4 + v^4 + 2*(u^2 + 1)*v^2 + 2*u^2 + 1) d/du*dv*du*dv
- 4/(u^4 + v^4 + 2*(u^2 + 1)*v^2 + 2*u^2 + 1) d/du*dv*dv*du
- 4/(u^4 + v^4 + 2*(u^2 + 1)*v^2 + 2*u^2 + 1) d/dv*du*du*dv
+ 4/(u^4 + v^4 + 2*(u^2 + 1)*v^2 + 2*u^2 + 1) d/dv*du*dv*du

```

The Ricci tensor of  $g$ :

```

sage: ric = g.ricci() ; ric
Field of symmetric bilinear forms Ric(g) on the 2-dimensional
differentiable manifold S^2
sage: ric.display(eU)
Ric(g) = 4/(x^4 + y^4 + 2*(x^2 + 1)*y^2 + 2*x^2 + 1) dx*dx
+ 4/(x^4 + y^4 + 2*(x^2 + 1)*y^2 + 2*x^2 + 1) dy*dy
sage: ric.display(eV)
Ric(g) = 4/(u^4 + v^4 + 2*(u^2 + 1)*v^2 + 2*u^2 + 1) du*du
+ 4/(u^4 + v^4 + 2*(u^2 + 1)*v^2 + 2*u^2 + 1) dv*dv
sage: ric == g
True

```

The Ricci scalar of  $g$ :

```

sage: r = g.ricci_scalar() ; r
Scalar field r(g) on the 2-dimensional differentiable manifold S^2
sage: r.display()
r(g): S^2 --> R
on U: (x, y) |--> 2
on V: (u, v) |--> 2

```

In dimension 2, the Riemann tensor can be expressed entirely in terms of the Ricci scalar  $r$ :

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

This formula can be checked here, with the r.h.s. rewritten as  $-r g_{j[k} \delta^i{}_{l]}$ :

```

sage: delta = M.tangent_identity_field()
sage: riem == - r*(g*delta).antisymmetrize(2,3)
True

```

**christoffel\_symbols** (*chart=None*)

Christoffel symbols of `self` with respect to a chart.

INPUT:

- `chart` – (default: `None`) chart with respect to which the Christoffel symbols are required; if none is provided, the default chart of the metric's domain is assumed.

OUTPUT:

- the set of Christoffel symbols in the given chart, as an instance of `CompWithSym`

EXAMPLES:

Christoffel symbols of the flat metric on  $\mathbf{R}^3$  with respect to spherical coordinates:

```
sage: M = Manifold(3, 'R3', r'\RR^3', start_index=1)
sage: U = M.open_subset('U') # the complement of the half-plane (y=0, x>=0)
sage: X.<r,th,ph> = U.chart(r'r:(0,+oo) th:(0,pi):\theta ph:(0,2*pi):\phi')
sage: g = U.metric('g')
sage: g[1,1], g[2,2], g[3,3] = 1, r^2, r^2*sin(th)^2
sage: g.display() # the standard flat metric expressed in spherical_
↪coordinates
g = dr*dr + r^2 dth*dth + r^2*sin(th)^2 dph*dph
sage: Gam = g.christoffel_symbols() ; Gam
3-indices components w.r.t. Coordinate frame (U, (d/dr,d/dth,d/dph)),
with symmetry on the index positions (1, 2)
sage: type(Gam)
<class 'sage.tensor.modules.comp.CompWithSym'>
sage: Gam[:]
[[[0, 0, 0], [0, -r, 0], [0, 0, -r*sin(th)^2]],
 [[0, 1/r, 0], [1/r, 0, 0], [0, 0, -cos(th)*sin(th)]],
 [[0, 0, 1/r], [0, 0, cos(th)/sin(th)], [1/r, cos(th)/sin(th), 0]]]
sage: Gam[1,2,2]
-r
sage: Gam[2,1,2]
1/r
sage: Gam[3,1,3]
1/r
sage: Gam[3,2,3]
cos(th)/sin(th)
sage: Gam[2,3,3]
-cos(th)*sin(th)
```

Note that a better display of the Christoffel symbols is provided by the method `christoffel_symbols_display()`:

```
sage: g.christoffel_symbols_display()
Gam^r_th,th = -r
Gam^r_ph,ph = -r*sin(th)^2
Gam^th_r,th = 1/r
Gam^th_ph,ph = -cos(th)*sin(th)
Gam^ph_r,ph = 1/r
Gam^ph_th,ph = cos(th)/sin(th)
```

**christoffel\_symbols\_display** (*chart=None*, *symbol=None*, *latex\_symbol=None*,  
*index\_labels=None*, *index\_latex\_labels=None*,  
*coordinate\_labels=True*, *only\_nonzero=True*,  
*only\_nonredundant=True*)

Display the Christoffel symbols w.r.t. to a given chart, one per line.

The output is either text-formatted (console mode) or LaTeX-formatted (notebook mode).

INPUT:

- `chart` – (default: None) chart with respect to which the Christoffel symbols are defined; if none is provided, the default chart of the metric’s domain is assumed.
- `symbol` – (default: None) string specifying the symbol of the connection coefficients; if None, ‘`Gam`’ is used
- `latex_symbol` – (default: None) string specifying the LaTeX symbol for the components; if None, ‘`\Gamma`’ is used
- `index_labels` – (default: None) list of strings representing the labels of each index; if None, coordinate symbols are used except if `coordinate_symbols` is set to `False`, in which case integer labels are used
- `index_latex_labels` – (default: None) list of strings representing the LaTeX labels of each index; if None, coordinate LaTeX symbols are used, except if `coordinate_symbols` is set to `False`, in which case integer labels are used
- `coordinate_labels` – (default: `True`) boolean; if `True`, coordinate symbols are used by default (instead of integers)
- `only_nonzero` – (default: `True`) boolean; if `True`, only nonzero connection coefficients are displayed
- `only_nonredundant` – (default: `True`) boolean; if `True`, only nonredundant (w.r.t. the symmetry of the last two indices) connection coefficients are displayed

EXAMPLES:

Christoffel symbols of the flat metric on  $\mathbf{R}^3$  with respect to spherical coordinates:

```
sage: M = Manifold(3, 'R3', r'\RR^3', start_index=1)
sage: U = M.open_subset('U') # the complement of the half-plane (y=0, x>=0)
sage: X.<r,th,ph> = U.chart(r'r:(0,+oo) th:(0,pi):\theta ph:(0,2*pi):\phi')
sage: g = U.metric('g')
sage: g[1,1], g[2,2], g[3,3] = 1, r^2, r^2*sin(th)^2
sage: g.display() # the standard flat metric expressed in spherical_
↳coordinates
g = dr*dr + r^2 dth*dth + r^2*sin(th)^2 dph*dph
sage: g.christoffel_symbols_display()
Gam^r_th,th = -r
Gam^r_ph,ph = -r*sin(th)^2
Gam^th_r,th = 1/r
Gam^th_ph,ph = -cos(th)*sin(th)
Gam^ph_r,ph = 1/r
Gam^ph_th,ph = cos(th)/sin(th)
```

To list all nonzero Christoffel symbols, including those that can be deduced by symmetry, use `only_nonredundant=False`:

```
sage: g.christoffel_symbols_display(only_nonredundant=False)
Gam^r_th,th = -r
Gam^r_ph,ph = -r*sin(th)^2
Gam^th_r,th = 1/r
Gam^th_th,r = 1/r
Gam^th_ph,ph = -cos(th)*sin(th)
Gam^ph_r,ph = 1/r
Gam^ph_th,ph = cos(th)/sin(th)
```

```
Gam^ph_ph,r = 1/r
Gam^ph_ph,th = cos(th)/sin(th)
```

Listing all Christoffel symbols (except those that can be deduced by symmetry), including the vanishing one:

```
sage: g.christoffel_symbols_display(only_nonzero=False)
Gam^r_r,r = 0
Gam^r_r,th = 0
Gam^r_r,ph = 0
Gam^r_th,th = -r
Gam^r_th,ph = 0
Gam^r_ph,ph = -r*sin(th)^2
Gam^th_r,r = 0
Gam^th_r,th = 1/r
Gam^th_r,ph = 0
Gam^th_th,th = 0
Gam^th_th,ph = 0
Gam^th_ph,ph = -cos(th)*sin(th)
Gam^ph_r,r = 0
Gam^ph_r,th = 0
Gam^ph_r,ph = 1/r
Gam^ph_th,th = 0
Gam^ph_th,ph = cos(th)/sin(th)
Gam^ph_ph,ph = 0
```

Using integer labels:

```
sage: g.christoffel_symbols_display(coordinate_labels=False)
Gam^1_22 = -r
Gam^1_33 = -r*sin(th)^2
Gam^2_12 = 1/r
Gam^2_33 = -cos(th)*sin(th)
Gam^3_13 = 1/r
Gam^3_23 = cos(th)/sin(th)
```

**connection** (*name=None, latex\_name=None*)

Return the unique torsion-free affine connection compatible with `self`.

This is the so-called Levi-Civita connection.

INPUT:

- `name` – (default: `None`) name given to the Levi-Civita connection; if `None`, it is formed from the metric name
- `latex_name` – (default: `None`) LaTeX symbol to denote the Levi-Civita connection; if `None`, it is set to `name`, or if the latter is `None` as well, it formed from the symbol  $\nabla$  and the metric symbol

OUTPUT:

- the Levi-Civita connection, as an instance of `LeviCivitaConnection`.

EXAMPLES:

Levi-Civita connection associated with the Euclidean metric on  $\mathbb{R}^3$ :

```
sage: M = Manifold(3, 'R^3', start_index=1)
sage: # Let us use spherical coordinates on R^3:
sage: U = M.open_subset('U') # the complement of the half-plane (y=0, x>=0)
```

```

sage: c_spher.<r,th,ph> = U.chart(r'r:(0,+oo) th:(0,pi):\theta ph:(0,
↪2*pi):\phi')
sage: g = U.metric('g')
sage: g[1,1], g[2,2], g[3,3] = 1, r^2 , (r*sin(th))^2 # the Euclidean metric
sage: g.connection()
Levi-Civita connection nabla_g associated with the Riemannian
metric g on the Open subset U of the 3-dimensional differentiable
manifold R^3
sage: g.connection().display() # Nonzero connection coefficients
Gam^r_th,th = -r
Gam^r_ph,ph = -r*sin(th)^2
Gam^th_r,th = 1/r
Gam^th_th,r = 1/r
Gam^th_ph,ph = -cos(th)*sin(th)
Gam^ph_r,ph = 1/r
Gam^ph_th,ph = cos(th)/sin(th)
Gam^ph_ph,r = 1/r
Gam^ph_ph,th = cos(th)/sin(th)

```

Test of compatibility with the metric:

```

sage: Dg = g.connection()(g) ; Dg
Tensor field nabla_g(g) of type (0,3) on the Open subset U of the
3-dimensional differentiable manifold R^3
sage: Dg == 0
True
sage: Dig = g.connection()(g.inverse()) ; Dig
Tensor field nabla_g(inv_g) of type (2,1) on the Open subset U of
the 3-dimensional differentiable manifold R^3
sage: Dig == 0
True

```

**cotton** (*name=None, latex\_name=None*)

Return the Cotton conformal tensor associated with the metric. The tensor has type (0,3) and is defined in terms of the Schouten tensor  $S$  (see `schouten()`):

$$C_{ijk} = (n - 2) (\nabla_k S_{ij} - \nabla_j S_{ik})$$

INPUT:

- `name` – (default: `None`) name given to the Cotton conformal tensor; if `None`, it is set to “Cot(g)”, where “g” is the metric’s name
- `latex_name` – (default: `None`) LaTeX symbol to denote the Cotton conformal tensor; if `None`, it is set to “ $\mathrm{Cot}(g)$ ”, where “g” is the metric’s name

OUTPUT:

- the Cotton conformal tensor  $Cot$ , as an instance of `TensorField`

EXAMPLES:

Checking that the Cotton tensor identically vanishes on a conformally flat 3-dimensional manifold, for instance the hyperbolic space  $H^3$ :

```

sage: M = Manifold(3, 'H^3', start_index=1)
sage: U = M.open_subset('U') # the complement of the half-plane (y=0, x>=0)
sage: X.<rh,th,ph> = U.chart(r'rh:(0,+oo):\rho th:(0,pi):\theta ph:(0,
↪2*pi):\phi')
sage: g = U.metric('g')

```

```

sage: b = var('b')
sage: g[1,1], g[2,2], g[3,3] = b^2, (b*sinh(rh))^2, (b*sinh(rh)*sin(th))^2
sage: g.display() # standard metric on H^3:
g = b^2 drh*drh + b^2*sinh(rh)^2 dth*dth
  + b^2*sin(th)^2*sinh(rh)^2 dph*dph
sage: Cot = g.cotton() ; Cot # long time
Tensor field Cot(g) of type (0,3) on the Open subset U of the
  3-dimensional differentiable manifold H^3
sage: Cot == 0 # long time
True

```

**cotton\_york** (*name=None, latex\_name=None*)

Return the Cotton-York conformal tensor associated with the metric. The tensor has type (0,2) and is only defined for manifolds of dimension 3. It is defined in terms of the Cotton tensor  $C$  (see `cotton()`) or the Schouten tensor  $S$  (see `schouten()`):

$$CY_{ij} = \frac{1}{2} \epsilon^{kl} C_{jlk} = \epsilon^{kl} \nabla_k S_{lj}$$

INPUT:

- `name` – (default: `None`) name given to the Cotton-York tensor; if `None`, it is set to “CY(g)”, where “g” is the metric’s name
- `latex_name` – (default: `None`) LaTeX symbol to denote the Cotton-York tensor; if `None`, it is set to “ $\mathrm{CY}(g)$ ”, where “g” is the metric’s name

OUTPUT:

- the Cotton-York conformal tensor  $CY$ , as an instance of *TensorField*

EXAMPLES:

Compute the determinant of the Cotton-York tensor for the Heisenberg group with the left invariant metric:

```

sage: M = Manifold(3, 'Nil', start_index=1)
sage: X.<x,y,z> = M.chart()
sage: g = M.riemannian_metric('g')
sage: g[1,1], g[2,2], g[2,3], g[3,3] = 1, 1+x^2, -x, 1
sage: g.display()
g = dx*dx + (x^2 + 1) dy*dy - x dy*dz - x dz*dy + dz*dz
sage: CY = g.cotton_york() ; CY # long time
Tensor field CY(g) of type (0,2) on the 3-dimensional
  differentiable manifold Nil
sage: CY.display() # long time
CY(g) = 1/2 dx*dx + (-x^2 + 1/2) dy*dy + x dy*dz + x dz*dy - dz*dz
sage: det(CY[:]) # long time
-1/4

```

**determinant** (*frame=None*)

Determinant of the metric components in the specified frame.

INPUT:

- `frame` – (default: `None`) vector frame with respect to which the components  $g_{ij}$  of the metric are defined; if `None`, the default frame of the metric’s domain is used. If a chart is provided instead of a frame, the associated coordinate frame is used

OUTPUT:

- the determinant  $\det(g_{ij})$ , as an instance of *DiffScalarField*

## EXAMPLES:

Metric determinant on a 2-dimensional manifold:

```

sage: M = Manifold(2, 'M', start_index=1)
sage: X.<x,y> = M.chart()
sage: g = M.metric('g')
sage: g[1,1], g[1, 2], g[2, 2] = 1+x, x*y , 1-y
sage: g[:]
[ x + 1      x*y]
[   x*y -y + 1]
sage: s = g.determinant() # determinant in M's default frame
sage: s.expr()
-x^2*y^2 - (x + 1)*y + x + 1

```

Determinant in a frame different from the default's one:

```

sage: Y.<u,v> = M.chart()
sage: ch_X_Y = X.transition_map(Y, [x+y, x-y])
sage: ch_X_Y.inverse()
Change of coordinates from Chart (M, (u, v)) to Chart (M, (x, y))
sage: g.comp(Y.frame())[:, Y]
[ 1/8*u^2 - 1/8*v^2 + 1/4*v + 1/2                                1/4*u]
[                                1/4*u -1/8*u^2 + 1/8*v^2 + 1/4*v + 1/2]
sage: g.determinant(Y.frame()).expr()
-1/4*x^2*y^2 - 1/4*(x + 1)*y + 1/4*x + 1/4
sage: g.determinant(Y.frame()).expr(Y)
-1/64*u^4 - 1/64*v^4 + 1/32*(u^2 + 2)*v^2 - 1/16*u^2 + 1/4*v + 1/4

```

A chart can be passed instead of a frame:

```

sage: g.determinant(X) is g.determinant(X.frame())
True
sage: g.determinant(Y) is g.determinant(Y.frame())
True

```

The metric determinant depends on the frame:

```

sage: g.determinant(X.frame()) == g.determinant(Y.frame())
False

```

**hodge\_star** (*pform*)

Compute the Hodge dual of a differential form with respect to the metric.

If the differential form is a  $p$ -form  $A$ , its *Hodge dual* with respect to the metric  $g$  is the  $(n - p)$ -form  $*A$  defined by

$$*A_{i_1 \dots i_{n-p}} = \frac{1}{p!} A_{k_1 \dots k_p} \epsilon^{k_1 \dots k_p}_{i_1 \dots i_{n-p}}$$

where  $n$  is the manifold's dimension,  $\epsilon$  is the volume  $n$ -form associated with  $g$  (see `volume_form()`) and the indices  $k_1, \dots, k_p$  are raised with  $g$ .

INPUT:

- `pform`: a  $p$ -form  $A$ ; must be an instance of `DiffScalarField` for  $p = 0$  and of `DiffForm` or `DiffFormParal` for  $p \geq 1$ .

OUTPUT:

- the  $(n - p)$ -form  $*A$

## EXAMPLES:

Hodge dual of a 1-form in the Euclidean space  $R^3$ :

```

sage: M = Manifold(3, 'M', start_index=1)
sage: X.<x,y,z> = M.chart()
sage: g = M.metric('g')
sage: g[1,1], g[2,2], g[3,3] = 1, 1, 1
sage: a = M.one_form('A')
sage: var('Ax Ay Az')
(Ax, Ay, Az)
sage: a[:] = (Ax, Ay, Az)
sage: sa = g.hodge_star(a) ; sa
2-form *A on the 3-dimensional differentiable manifold M
sage: sa.display()
*A = Az dx/\dy - Ay dx/\dz + Ax dy/\dz
sage: ssa = g.hodge_star(sa) ; ssa
1-form **A on the 3-dimensional differentiable manifold M
sage: ssa.display()
**A = Ax dx + Ay dy + Az dz
sage: ssa == a # must hold for a Riemannian metric in dimension 3
True

```

Hodge dual of a 0-form (scalar field) in  $R^3$ :

```

sage: f = M.scalar_field(function('F')(x,y,z), name='f')
sage: sf = g.hodge_star(f) ; sf
3-form *f on the 3-dimensional differentiable manifold M
sage: sf.display()
*f = F(x, y, z) dx/\dy/\dz
sage: ssf = g.hodge_star(sf) ; ssf
Scalar field **f on the 3-dimensional differentiable manifold M
sage: ssf.display()
**f: M --> R
      (x, y, z) |--> F(x, y, z)
sage: ssf == f # must hold for a Riemannian metric
True

```

Hodge dual of a 0-form in Minkowski spacetime:

```

sage: M = Manifold(4, 'M')
sage: X.<t,x,y,z> = M.chart()
sage: g = M.lorentzian_metric('g')
sage: g[0,0], g[1,1], g[2,2], g[3,3] = -1, 1, 1, 1
sage: g.display() # Minkowski metric
g = -dt*dt + dx*dx + dy*dy + dz*dz
sage: var('f0')
f0
sage: f = M.scalar_field(f0, name='f')
sage: sf = g.hodge_star(f) ; sf
4-form *f on the 4-dimensional differentiable manifold M
sage: sf.display()
*f = f0 dt/\dx/\dy/\dz
sage: ssf = g.hodge_star(sf) ; ssf
Scalar field **f on the 4-dimensional differentiable manifold M
sage: ssf.display()
**f: M --> R
      (t, x, y, z) |--> -f0
sage: ssf == -f # must hold for a Lorentzian metric

```

```
True
```

Hodge dual of a 1-form in Minkowski spacetime:

```
sage: a = M.one_form('A')
sage: var('At Ax Ay Az')
(At, Ax, Ay, Az)
sage: a[:] = (At, Ax, Ay, Az)
sage: a.display()
A = At dt + Ax dx + Ay dy + Az dz
sage: sa = g.hodge_star(a) ; sa
3-form *A on the 4-dimensional differentiable manifold M
sage: sa.display()
*A = -Az dt/\dx/\dy + Ay dt/\dx/\dz - Ax dt/\dy/\dz - At dx/\dy/\dz
sage: ssa = g.hodge_star(sa) ; ssa
1-form **A on the 4-dimensional differentiable manifold M
sage: ssa.display()
**A = At dt + Ax dx + Ay dy + Az dz
sage: ssa == a # must hold for a Lorentzian metric in dimension 4
True
```

Hodge dual of a 2-form in Minkowski spacetime:

```
sage: F = M.diff_form(2, 'F')
sage: var('Ex Ey Ez Bx By Bz')
(Ex, Ey, Ez, Bx, By, Bz)
sage: F[0,1], F[0,2], F[0,3] = -Ex, -Ey, -Ez
sage: F[1,2], F[1,3], F[2,3] = Bz, -By, Bx
sage: F[:]
[ 0 -Ex -Ey -Ez]
[ Ex  0 Bz -By]
[ Ey -Bz  0 Bx]
[ Ez By -Bx  0]
sage: sF = g.hodge_star(F) ; sF
2-form *F on the 4-dimensional differentiable manifold M
sage: sF[:]
[ 0 Bx By Bz]
[-Bx  0 Ez -Ey]
[-By -Ez  0 Ex]
[-Bz Ey -Ex  0]
sage: ssF = g.hodge_star(sF) ; ssF
2-form **F on the 4-dimensional differentiable manifold M
sage: ssF[:]
[ 0 Ex Ey Ez]
[-Ex  0 -Bz By]
[-Ey Bz  0 -Bx]
[-Ez -By Bx  0]
sage: ssF.display()
**F = Ex dt/\dx + Ey dt/\dy + Ez dt/\dz - Bz dx/\dy + By dx/\dz
- Bx dy/\dz
sage: F.display()
F = -Ex dt/\dx - Ey dt/\dy - Ez dt/\dz + Bz dx/\dy - By dx/\dz
+ Bx dy/\dz
sage: ssF == -F # must hold for a Lorentzian metric in dimension 4
True
```

Test of the standard identity

$$*(A \wedge B) = \epsilon(A^\sharp, B^\sharp, \dots)$$

where  $A$  and  $B$  are any 1-forms and  $A^\sharp$  and  $B^\sharp$  the vectors associated to them by the metric  $g$  (index raising):

```
sage: b = M.one_form('B')
sage: var('Bt Bx By Bz')
(Bt, Bx, By, Bz)
sage: b[:] = (Bt, Bx, By, Bz) ; b.display()
B = Bt dt + Bx dx + By dy + Bz dz
sage: epsilon = g.volume_form()
sage: g.hodge_star(a.wedge(b)) == epsilon.contract(0, a.up(g)).contract(0, b.
↪up(g))
True
```

**inverse()**

Return the inverse metric.

OUTPUT:

- instance of *TensorField* with tensor\_type = (2,0) representing the inverse metric

EXAMPLES:

Inverse of the standard metric on the 2-sphere:

```
sage: M = Manifold(2, 'S^2', start_index=1)
sage: U = M.open_subset('U') ; V = M.open_subset('V')
sage: M.declare_union(U,V) # S^2 is the union of U and V
sage: c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart() # stereographic coord.
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....: intersection_name='W', restrictions1= x^2+y^2!=0,
....: restrictions2= u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: W = U.intersection(V) # the complement of the two poles
sage: eU = c_xy.frame() ; eV = c_uv.frame()
sage: g = M.metric('g')
sage: g[eU,1,1], g[eU,2,2] = 4/(1+x^2+y^2)^2, 4/(1+x^2+y^2)^2
sage: g.add_comp_by_continuation(eV, W, c_uv)
sage: ginv = g.inverse(); ginv
Tensor field inv_g of type (2,0) on the 2-dimensional differentiable manifold_
↪S^2
sage: ginv.display(eU)
inv_g = (1/4*x^4 + 1/4*y^4 + 1/2*(x^2 + 1)*y^2 + 1/2*x^2 + 1/4) d/dx*d/dx
+ (1/4*x^4 + 1/4*y^4 + 1/2*(x^2 + 1)*y^2 + 1/2*x^2 + 1/4) d/dy*d/dy
sage: ginv.display(eV)
inv_g = (1/4*u^4 + 1/4*v^4 + 1/2*(u^2 + 1)*v^2 + 1/2*u^2 + 1/4) d/du*d/du
+ (1/4*u^4 + 1/4*v^4 + 1/2*(u^2 + 1)*v^2 + 1/2*u^2 + 1/4) d/dv*d/dv
```

Let us check that  $ginv$  is indeed the inverse of  $g$ :

```
sage: s = g.contract(ginv); s # contraction of last index of g with first_
↪index of ginv
Tensor field of type (1,1) on the 2-dimensional differentiable manifold S^2
sage: s == M.tangent_identity_field()
True
```

**restrict** (*subdomain*, *dest\_map=None*)

Return the restriction of the metric to some subdomain.

If the restriction has not been defined yet, it is constructed here.

INPUT:

- *subdomain* – open subset  $U$  of the metric’s domain (must be an instance of *DifferentiableManifold*)
- *dest\_map* – (default: None) destination map  $\Phi : U \rightarrow V$ , where  $V$  is a subdomain of *self*. *\_codomain* (type: *DiffMap*) If None, the restriction of *self*.*\_vmodule*.*\_dest\_map* to  $U$  is used.

OUTPUT:

- instance of *PseudoRiemannianMetric* representing the restriction.

EXAMPLES:

```
sage: M = Manifold(5, 'M')
sage: g = M.metric('g', signature=3)
sage: U = M.open_subset('U')
sage: g.restrict(U)
Lorentzian metric g on the Open subset U of the
5-dimensional differentiable manifold M
sage: g.restrict(U).signature()
3
```

See the top documentation of *PseudoRiemannianMetric* for more examples.

**ricci** (*name=None*, *latex\_name=None*)

Return the Ricci tensor associated with the metric.

This method is actually a shortcut for *self*.*connection*()*.ricci*()

The Ricci tensor is the tensor field  $Ric$  of type (0,2) defined from the Riemann curvature tensor  $R$  by

$$Ric(u, v) = R(e^i, u, e_i, v)$$

for any vector fields  $u$  and  $v$ ,  $(e_i)$  being any vector frame and  $(e^i)$  the dual coframe.

INPUT:

- *name* – (default: None) name given to the Ricci tensor; if none, it is set to “Ric(g)”, where “g” is the metric’s name
- *latex\_name* – (default: None) LaTeX symbol to denote the Ricci tensor; if none, it is set to “ $\mathrm{Ric}(g)$ ”, where “g” is the metric’s name

OUTPUT:

- the Ricci tensor  $Ric$ , as an instance of *TensorField* of tensor type (0,2) and symmetric

EXAMPLES:

Ricci tensor of the standard metric on the 2-sphere:

```
sage: M = Manifold(2, 'S^2', start_index=1)
sage: U = M.open_subset('U') # the complement of a meridian (domain of
↔ spherical coordinates)
sage: c_spher.<th,ph> = U.chart(r'th:(0,pi):\theta ph:(0,2*pi):\phi')
sage: a = var('a') # the sphere radius
sage: g = U.metric('g')
```

```

sage: g[1,1], g[2,2] = a^2, a^2*sin(th)^2
sage: g.display() # standard metric on the 2-sphere of radius a:
g = a^2 dth*dth + a^2*sin(th)^2 dph*dph
sage: g.ricci()
Field of symmetric bilinear forms Ric(g) on the Open subset U of
the 2-dimensional differentiable manifold S^2
sage: g.ricci()[:]:
[      1      0]
[      0 sin(th)^2]
sage: g.ricci() == a^(-2) * g
True

```

**ricci\_scalar** (*name=None, latex\_name=None*)

Return the Ricci scalar associated with the metric.

The Ricci scalar is the scalar field  $r$  defined from the Ricci tensor  $Ric$  and the metric tensor  $g$  by

$$r = g^{ij} Ric_{ij}$$

INPUT:

- *name* – (default: None) name given to the Ricci scalar; if none, it is set to “r(g)”, where “g” is the metric’s name
- *latex\_name* – (default: None) LaTeX symbol to denote the Ricci scalar; if none, it is set to “ $\mathrm{r}(g)$ ”, where “g” is the metric’s name

OUTPUT:

- the Ricci scalar  $r$ , as an instance of *DiffScalarField*

EXAMPLES:

Ricci scalar of the standard metric on the 2-sphere:

```

sage: M = Manifold(2, 'S^2', start_index=1)
sage: U = M.open_subset('U') # the complement of a meridian (domain of
↳ spherical coordinates)
sage: c_spher.<th,ph> = U.chart(r'th:(0,pi):\theta ph:(0,2*pi):\phi')
sage: a = var('a') # the sphere radius
sage: g = U.metric('g')
sage: g[1,1], g[2,2] = a^2, a^2*sin(th)^2
sage: g.display() # standard metric on the 2-sphere of radius a:
g = a^2 dth*dth + a^2*sin(th)^2 dph*dph
sage: g.ricci_scalar()
Scalar field r(g) on the Open subset U of the 2-dimensional
differentiable manifold S^2
sage: g.ricci_scalar().display() # The Ricci scalar is constant:
r(g): U --> R
(th, ph) |--> 2/a^2

```

**riemann** (*name=None, latex\_name=None*)

Return the Riemann curvature tensor associated with the metric.

This method is actually a shortcut for `self.connection().riemann()`

The Riemann curvature tensor is the tensor field  $R$  of type (1,3) defined by

$$R(\omega, u, v, w) = \langle \omega, \nabla_u \nabla_v w - \nabla_v \nabla_u w - \nabla_{[u,v]} w \rangle$$

for any 1-form  $\omega$  and any vector fields  $u, v$  and  $w$ .

INPUT:

- `name` – (default: `None`) name given to the Riemann tensor; if none, it is set to “`Riem(g)`”, where “`g`” is the metric’s name
- `latex_name` – (default: `None`) LaTeX symbol to denote the Riemann tensor; if none, it is set to “`\mathrm{Riem}(g)`”, where “`g`” is the metric’s name

OUTPUT:

- the Riemann curvature tensor  $R$ , as an instance of `TensorField`

EXAMPLES:

Riemann tensor of the standard metric on the 2-sphere:

```
sage: M = Manifold(2, 'S^2', start_index=1)
sage: U = M.open_subset('U') # the complement of a meridian (domain of_
↳spherical coordinates)
sage: c_spher.<th,ph> = U.chart(r'th:(0,pi):\theta ph:(0,2*pi):\phi')
sage: a = var('a') # the sphere radius
sage: g = U.metric('g')
sage: g[1,1], g[2,2] = a^2, a^2*sin(th)^2
sage: g.display() # standard metric on the 2-sphere of radius a:
g = a^2 dth*dth + a^2*sin(th)^2 dph*dph
sage: g.riemann()
Tensor field Riem(g) of type (1,3) on the Open subset U of the
2-dimensional differentiable manifold S^2
sage: g.riemann()[:]
[[[[0, 0], [0, 0]], [[0, sin(th)^2], [-sin(th)^2, 0]]],
 [[0, (cos(th)^2 - 1)/sin(th)^2], [1, 0]], [[0, 0], [0, 0]]]]
```

In dimension 2, the Riemann tensor can be expressed entirely in terms of the Ricci scalar  $r$ :

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

This formula can be checked here, with the r.h.s. rewritten as  $-r g_{j[k} \delta^i{}_{l]}$ :

```
sage: g.riemann() == \
....: -g.ricci_scalar()*(g*U.tangent_identity_field()).antisymmetrize(2,3)
True
```

**schouten** (`name=None`, `latex_name=None`)

Return the Schouten tensor associated with the metric.

The Schouten tensor is the tensor field  $Sc$  of type (0,2) defined from the Ricci curvature tensor  $Ric$  (see `ricci()`) and the scalar curvature  $r$  (see `ricci_scalar()`) and the metric  $g$  by

$$Sc(u, v) = \frac{1}{n-2} \left( Ric(u, v) + \frac{r}{2(n-1)} g(u, v) \right)$$

for any vector fields  $u$  and  $v$ .

INPUT:

- `name` – (default: `None`) name given to the Schouten tensor; if none, it is set to “`Schouten(g)`”, where “`g`” is the metric’s name
- `latex_name` – (default: `None`) LaTeX symbol to denote the Schouten tensor; if none, it is set to “`\mathrm{Schouten}(g)`”, where “`g`” is the metric’s name

OUTPUT:

- the Schouten tensor  $Sc$ , as an instance of *TensorField* of tensor type (0,2) and symmetric

EXAMPLES:

Schouten tensor of the left invariant metric of Heisenberg's Nil group:

```
sage: M = Manifold(3, 'Nil', start_index=1)
sage: X.<x,y,z> = M.chart()
sage: g = M.riemannian_metric('g')
sage: g[1,1], g[2,2], g[2,3], g[3,3] = 1, 1+x^2, -x, 1
sage: g.display()
g = dx*dx + (x^2 + 1) dy*dy - x dy*dz - x dz*dy + dz*dz
sage: g.schouten()
Field of symmetric bilinear forms Schouten(g) on the 3-dimensional
differentiable manifold Nil
sage: g.schouten().display()
Schouten(g) = -3/8 dx*dx + (5/8*x^2 - 3/8) dy*dy - 5/8*x dy*dz
- 5/8*x dz*dy + 5/8 dz*dz
```

**set** (*symbiform*)

Defines the metric from a field of symmetric bilinear forms

INPUT:

- symbiform* – instance of *TensorField* representing a field of symmetric bilinear forms

EXAMPLES:

Metric defined from a field of symmetric bilinear forms on a non-parallelizable 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U') ; V = M.open_subset('V')
sage: M.declare_union(U,V) # M is the union of U and V
sage: c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart()
sage: xy_to_uv = c_xy.transition_map(c_uv, (x+y, x-y), intersection_name='W',
....:                               restrictions1= x>0, restrictions2= u+v>0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: W = U.intersection(V)
sage: eU = c_xy.frame() ; eV = c_uv.frame()
sage: h = M.sym_bilin_form_field(name='h')
sage: h[eU,0,0], h[eU,0,1], h[eU,1,1] = 1+x, x*y, 1-y
sage: h.add_comp_by_continuation(eV, W, c_uv)
sage: h.display(eU)
h = (x + 1) dx*dx + x*y dx*dy + x*y dy*dx + (-y + 1) dy*dy
sage: h.display(eV)
h = (1/8*u^2 - 1/8*v^2 + 1/4*v + 1/2) du*du + 1/4*u du*dv
+ 1/4*u dv*du + (-1/8*u^2 + 1/8*v^2 + 1/4*v + 1/2) dv*dv
sage: g = M.metric('g')
sage: g.set(h)
sage: g.display(eU)
g = (x + 1) dx*dx + x*y dx*dy + x*y dy*dx + (-y + 1) dy*dy
sage: g.display(eV)
g = (1/8*u^2 - 1/8*v^2 + 1/4*v + 1/2) du*du + 1/4*u du*dv
+ 1/4*u dv*du + (-1/8*u^2 + 1/8*v^2 + 1/4*v + 1/2) dv*dv
```

**signature** ()

Signature of the metric.

OUTPUT:

- signature  $S$  of the metric, defined as the integer  $S = n_+ - n_-$ , where  $n_+$  (resp.  $n_-$ ) is the number of positive terms (resp. number of negative terms) in any diagonal writing of the metric components

EXAMPLES:

Signatures on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: g = M.metric('g') # if not specified, the signature is Riemannian
sage: g.signature()
2
sage: h = M.metric('h', signature=0)
sage: h.signature()
0
```

**sqrt\_abs\_det** (frame=None)

Square root of the absolute value of the determinant of the metric components in the specified frame.

INPUT:

- frame – (default: None) vector frame with respect to which the components  $g_{ij}$  of `self` are defined; if None, the domain's default frame is used. If a chart is provided, the associated coordinate frame is used

OUTPUT:

- $\sqrt{|\det(g_{ij})|}$ , as an instance of *DiffScalarField*

EXAMPLES:

Standard metric in the Euclidean space  $\mathbf{R}^3$  with spherical coordinates:

```
sage: M = Manifold(3, 'M', start_index=1)
sage: U = M.open_subset('U') # the complement of the half-plane (y=0, x>=0)
sage: c_spher.<r,th,ph> = U.chart(r'r:(0,+oo) th:(0,pi):\theta ph:(0,
↔2*pi):\phi')
sage: g = U.metric('g')
sage: g[1,1], g[2,2], g[3,3] = 1, r^2, (r*sin(th))^2
sage: g.display()
g = dr*dr + r^2 dth*dth + r^2*sin(th)^2 dph*dph
sage: g.sqrt_abs_det().expr()
r^2*sin(th)
```

Metric determinant on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', start_index=1)
sage: X.<x,y> = M.chart()
sage: g = M.metric('g')
sage: g[1,1], g[1,2], g[2,2] = 1+x, x*y, 1-y
sage: g[:]
[ x + 1    x*y]
[  x*y  -y + 1]
sage: s = g.sqrt_abs_det() ; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.expr()
sqrt(-x^2*y^2 - (x + 1)*y + x + 1)
```

Determinant in a frame different from the default's one:

```
sage: Y.<u,v> = M.chart()
sage: ch_X_Y = X.transition_map(Y, [x+y, x-y])
```

```

sage: ch_X_Y.inverse()
Change of coordinates from Chart (M, (u, v)) to Chart (M, (x, y))
sage: g[Y.frame(), :, Y]
[ 1/8*u^2 - 1/8*v^2 + 1/4*v + 1/2                                1/4*u]
[                                1/4*u -1/8*u^2 + 1/8*v^2 + 1/4*v + 1/2]
sage: g.sqrt_abs_det(Y.frame()).expr()
1/2*sqrt(-x^2*y^2 - (x + 1)*y + x + 1)
sage: g.sqrt_abs_det(Y.frame()).expr(Y)
1/8*sqrt(-u^4 - v^4 + 2*(u^2 + 2)*v^2 - 4*u^2 + 16*v + 16)

```

A chart can be passed instead of a frame:

```

sage: g.sqrt_abs_det(Y) is g.sqrt_abs_det(Y.frame())
True

```

The metric determinant depends on the frame:

```

sage: g.sqrt_abs_det(X.frame()) == g.sqrt_abs_det(Y.frame())
False

```

### **volume\_form** (*contra=0*)

Volume form (Levi-Civita tensor)  $\epsilon$  associated with the metric.

This assumes that the manifold is orientable.

The volume form  $\epsilon$  is a  $n$ -form ( $n$  being the manifold's dimension) such that for any vector basis  $(e_i)$  that is orthonormal with respect to the metric,

$$\epsilon(e_1, \dots, e_n) = \pm 1$$

There are only two such  $n$ -forms, which are opposite of each other. The volume form  $\epsilon$  is selected such that the domain's default frame is right-handed with respect to it.

INPUT:

- *contra* – (default: 0) number of contravariant indices of the returned tensor

OUTPUT:

- if *contra* = 0 (default value): the volume  $n$ -form  $\epsilon$ , as an instance of *DiffForm*
- if *contra* =  $k$ , with  $1 \leq k \leq n$ , the tensor field of type  $(k, n-k)$  formed from  $\epsilon$  by raising the first  $k$  indices with the metric (see method *up()*); the output is then an instance of *TensorField*, with the appropriate antisymmetries

EXAMPLES:

Volume form on  $\mathbf{R}^3$  with spherical coordinates:

```

sage: M = Manifold(3, 'M', start_index=1)
sage: U = M.open_subset('U') # the complement of the half-plane (y=0, x>=0)
sage: c_spher.<r,th,ph> = U.chart(r'r:(0,+oo) th:(0,pi):\theta ph:(0,
↪2*pi):\phi')
sage: g = U.metric('g')
sage: g[1,1], g[2,2], g[3,3] = 1, r^2, (r*sin(th))^2
sage: g.display()
g = dr*dr + r^2 dth*dth + r^2*sin(th)^2 dph*dph
sage: eps = g.volume_form() ; eps
3-form eps_g on the Open subset U of the 3-dimensional
differentiable manifold M
sage: eps.display()

```

```

eps_g = r^2*sin(th) dr/\dth/\dph
sage: eps[[1,2,3]] == g.sqrt_abs_det()
True
sage: latex(eps)
\epsilon_{g}

```

The tensor field of components  $\epsilon^i_{jk}$  (contra=1):

```

sage: eps1 = g.volume_form(1) ; eps1
Tensor field of type (1,2) on the Open subset U of the
3-dimensional differentiable manifold M
sage: eps1.symmetries()
no symmetry; antisymmetry: (1, 2)
sage: eps1[:]
[[[0, 0, 0], [0, 0, r^2*sin(th)], [0, -r^2*sin(th), 0]],
 [[0, 0, -sin(th)], [0, 0, 0], [sin(th), 0, 0]],
 [[0, 1/sin(th), 0], [-1/sin(th), 0, 0], [0, 0, 0]]]

```

The tensor field of components  $\epsilon^{ij}_k$  (contra=2):

```

sage: eps2 = g.volume_form(2) ; eps2
Tensor field of type (2,1) on the Open subset U of the
3-dimensional differentiable manifold M
sage: eps2.symmetries()
no symmetry; antisymmetry: (0, 1)
sage: eps2[:]
[[[0, 0, 0], [0, 0, sin(th)], [0, -1/sin(th), 0]],
 [[0, 0, -sin(th)], [0, 0, 0], [1/(r^2*sin(th)), 0, 0]],
 [[0, 1/sin(th), 0], [-1/(r^2*sin(th)), 0, 0], [0, 0, 0]]]

```

The tensor field of components  $\epsilon^{ijk}$  (contra=3):

```

sage: eps3 = g.volume_form(3) ; eps3
Tensor field of type (3,0) on the Open subset U of the
3-dimensional differentiable manifold M
sage: eps3.symmetries()
no symmetry; antisymmetry: (0, 1, 2)
sage: eps3[:]
[[[0, 0, 0], [0, 0, 1/(r^2*sin(th))], [0, -1/(r^2*sin(th)), 0]],
 [[0, 0, -1/(r^2*sin(th))], [0, 0, 0], [1/(r^2*sin(th)), 0, 0]],
 [[0, 1/(r^2*sin(th)), 0], [-1/(r^2*sin(th)), 0, 0], [0, 0, 0]]]
sage: eps3[1,2,3]
1/(r^2*sin(th))
sage: eps3[[1,2,3]] * g.sqrt_abs_det() == 1
True

```

**weyl** (*name=None, latex\_name=None*)

Return the Weyl conformal tensor associated with the metric.

The Weyl conformal tensor is the tensor field  $C$  of type (1,3) defined as the trace-free part of the Riemann curvature tensor  $R$

INPUT:

- *name* – (default: None) name given to the Weyl conformal tensor; if None, it is set to “C(g)”, where “g” is the metric’s name
- *latex\_name* – (default: None) LaTeX symbol to denote the Weyl conformal tensor; if None, it is set to “ $\mathrm{C}(g)$ ”, where “g” is the metric’s name

OUTPUT:

- the Weyl conformal tensor  $C$ , as an instance of `TensorField`

EXAMPLES:

Checking that the Weyl tensor identically vanishes on a 3-dimensional manifold, for instance the hyperbolic space  $H^3$ :

```
sage: M = Manifold(3, 'H^3', start_index=1)
sage: U = M.open_subset('U') # the complement of the half-plane (y=0, x>=0)
sage: X.<rh,th,ph> = U.chart(r'rh:(0,+oo):\rho th:(0,pi):\theta ph:(0,
↪2*pi):\phi')
sage: g = U.metric('g')
sage: b = var('b')
sage: g[1,1], g[2,2], g[3,3] = b^2, (b*sinh(rh))^2, (b*sinh(rh)*sin(th))^2
sage: g.display() # standard metric on H^3:
g = b^2 drh*drh + b^2*sinh(rh)^2 dth*dth
+ b^2*sin(th)^2*sinh(rh)^2 dph*dph
sage: C = g.weyl() ; C
Tensor field C(g) of type (1,3) on the Open subset U of the
3-dimensional differentiable manifold H^3
sage: C == 0
True
```

**class** `sage.manifolds.differentiable.metric.PseudoRiemannianMetricParal` (*vector\_field\_module*, *name*, *signature=None*, *label=None*, *tex\_name=None*)

Bases: `sage.manifolds.differentiable.metric.PseudoRiemannianMetric`, `sage.manifolds.differentiable.tensorfield_paral.TensorFieldParal`

Pseudo-Riemannian metric with values on a parallelizable manifold.

An instance of this class is a field of nondegenerate symmetric bilinear forms (metric field) along a differentiable manifold  $U$  with values in a parallelizable manifold  $M$  over  $\mathbf{R}$ , via a differentiable mapping  $\Phi : U \rightarrow M$ . The standard case of a metric field on a manifold corresponds to  $U = M$  and  $\Phi = \text{Id}_M$ . Other common cases are  $\Phi$  being an immersion and  $\Phi$  being a curve in  $M$  ( $U$  is then an open interval of  $\mathbf{R}$ ).

A metric  $g$  is a field on  $U$ , such that at each point  $p \in U$ ,  $g(p)$  is a bilinear map of the type:

$$g(p) : T_q M \times T_q M \longrightarrow \mathbf{R}$$

where  $T_q M$  stands for the tangent space to manifold  $M$  at the point  $q = \Phi(p)$ , such that  $g(p)$  is symmetric:  $\forall (u, v) \in T_q M \times T_q M$ ,  $g(p)(v, u) = g(p)(u, v)$  and nondegenerate:  $(\forall v \in T_q M, g(p)(u, v) = 0) \implies u = 0$ .

---

**Note:** If  $M$  is not parallelizable, the class `PseudoRiemannianMetric` should be used instead.

---

INPUT:

- `vector_field_module` – free module  $\mathcal{X}(U, \Phi)$  of vector fields along  $U$  with values on  $\Phi(U) \subset M$
- `name` – name given to the metric
- `signature` – (default: `None`) signature  $S$  of the metric as a single integer:  $S = n_+ - n_-$ , where  $n_+$  (resp.  $n_-$ ) is the number of positive terms (resp. number of negative terms) in any diagonal writing of the metric components; if `signature` is `None`,  $S$  is set to the dimension of manifold  $M$  (Riemannian signature)

- `latex_name` – (default: None) LaTeX symbol to denote the metric; if None, it is formed from name

## EXAMPLES:

Metric on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', start_index=1)
sage: c_xy.<x,y> = M.chart()
sage: g = M.metric('g') ; g
Riemannian metric g on the 2-dimensional differentiable manifold M
sage: latex(g)
g
```

A metric is a special kind of tensor field and therefore inherits all the properties from class *TensorField*:

```
sage: g.parent()
Free module T^(0,2)(M) of type-(0,2) tensors fields on the
2-dimensional differentiable manifold M
sage: g.tensor_type()
(0, 2)
sage: g.symmetries() # g is symmetric:
symmetry: (0, 1); no antisymmetry
```

Setting the metric components in the manifold's default frame:

```
sage: g[1,1], g[1,2], g[2,2] = 1+x, x*y, 1-x
sage: g[:]
[ x + 1    x*y]
[  x*y -x + 1]
sage: g.display()
g = (x + 1) dx*dx + x*y dx*dy + x*y dy*dx + (-x + 1) dy*dy
```

Metric components in a frame different from the manifold's default one:

```
sage: c_uv.<u,v> = M.chart() # new chart on M
sage: xy_to_uv = c_xy.transition_map(c_uv, [x+y, x-y]) ; xy_to_uv
Change of coordinates from Chart (M, (x, y)) to Chart (M, (u, v))
sage: uv_to_xy = xy_to_uv.inverse() ; uv_to_xy
Change of coordinates from Chart (M, (u, v)) to Chart (M, (x, y))
sage: M.atlas()
[Chart (M, (x, y)), Chart (M, (u, v))]
sage: M.frames()
[Coordinate frame (M, (d/dx,d/dy)), Coordinate frame (M, (d/du,d/dv))]
sage: g[c_uv.frame(),:] # metric components in frame c_uv.frame() expressed in M
↳'s default chart (x,y)
[ 1/2*x*y + 1/2    1/2*x]
[    1/2*x -1/2*x*y + 1/2]
sage: g.display(c_uv.frame())
g = (1/2*x*y + 1/2) du*du + 1/2*x du*dv + 1/2*x dv*du
+ (-1/2*x*y + 1/2) dv*dv
sage: g[c_uv.frame(),:,c_uv] # metric components in frame c_uv.frame()
↳expressed in chart (u,v)
[ 1/8*u^2 - 1/8*v^2 + 1/2    1/4*u + 1/4*v]
[    1/4*u + 1/4*v -1/8*u^2 + 1/8*v^2 + 1/2]
sage: g.display(c_uv.frame(), c_uv)
g = (1/8*u^2 - 1/8*v^2 + 1/2) du*du + (1/4*u + 1/4*v) du*dv
+ (1/4*u + 1/4*v) dv*du + (-1/8*u^2 + 1/8*v^2 + 1/2) dv*dv
```

The inverse metric is obtained via *inverse()*:

```

sage: ig = g.inverse() ; ig
Tensor field inv_g of type (2,0) on the 2-dimensional differentiable
manifold M
sage: ig[:]
[ (x - 1)/(x^2*y^2 + x^2 - 1)      x*y/(x^2*y^2 + x^2 - 1)]
[      x*y/(x^2*y^2 + x^2 - 1)  -(x + 1)/(x^2*y^2 + x^2 - 1)]
sage: ig.display()
inv_g = (x - 1)/(x^2*y^2 + x^2 - 1) d/dx*d/dx
+ x*y/(x^2*y^2 + x^2 - 1) d/dx*d/dy + x*y/(x^2*y^2 + x^2 - 1) d/dy*d/dx
- (x + 1)/(x^2*y^2 + x^2 - 1) d/dy*d/dy

```

**inverse()**

Return the inverse metric.

OUTPUT:

- instance of *TensorFieldParal* with `tensor_type = (2,0)` representing the inverse metric

EXAMPLES:

Inverse metric on a 2-dimensional manifold:

```

sage: M = Manifold(2, 'M', start_index=1)
sage: c_xy.<x,y> = M.chart()
sage: g = M.metric('g')
sage: g[1,1], g[1,2], g[2,2] = 1+x, x*y, 1-x
sage: g[:] # components in the manifold's default frame
[ x + 1      x*y]
[  x*y  -x + 1]
sage: ig = g.inverse() ; ig
Tensor field inv_g of type (2,0) on the 2-dimensional
differentiable manifold M
sage: ig[:]
[ (x - 1)/(x^2*y^2 + x^2 - 1)      x*y/(x^2*y^2 + x^2 - 1)]
[      x*y/(x^2*y^2 + x^2 - 1)  -(x + 1)/(x^2*y^2 + x^2 - 1)]

```

If the metric is modified, the inverse metric is automatically updated:

```

sage: g[1,2] = 0 ; g[:]
[ x + 1      0]
[  0  -x + 1]
sage: g.inverse()[:]
[ 1/(x + 1)      0]
[      0  -1/(x - 1)]

```

**restrict** (*subdomain*, *dest\_map=None*)

Return the restriction of the metric to some subdomain.

If the restriction has not been defined yet, it is constructed here.

INPUT:

- subdomain* – open subset  $U$  of `self._domain` (must be an instance of *DifferentiableManifold*)
- dest\_map* – (default: `None`) destination map  $\Phi : U \rightarrow V$ , where  $V$  is a subdomain of `self`. `_codomain` (type: *DiffMap*) If `None`, the restriction of `self._vmodule._dest_map` to  $U$  is used.

OUTPUT:

- instance of *PseudoRiemannianMetricParal* representing the restriction.

## EXAMPLES:

Restriction of a Lorentzian metric on  $\mathbf{R}^2$  to the upper half plane:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: g = M.lorentzian_metric('g')
sage: g[0,0], g[1,1] = -1, 1
sage: U = M.open_subset('U', coord_def={X: y>0})
sage: gU = g.restrict(U); gU
Lorentzian metric g on the Open subset U of the 2-dimensional
differentiable manifold M
sage: gU.signature()
0
sage: gU.display()
g = -dx*dx + dy*dy
```

**ricci\_scalar** (*name=None, latex\_name=None*)

Return the metric's Ricci scalar.

The Ricci scalar is the scalar field  $r$  defined from the Ricci tensor  $Ric$  and the metric tensor  $g$  by

$$r = g^{ij} Ric_{ij}$$

## INPUT:

- name – (default: None) name given to the Ricci scalar; if none, it is set to “r(g)”, where “g” is the metric's name
- latex\_name – (default: None) LaTeX symbol to denote the Ricci scalar; if none, it is set to “\mathrm{r}(g)”, where “g” is the metric's name

## OUTPUT:

- the Ricci scalar  $r$ , as an instance of *DiffScalarField*

## EXAMPLES:

Ricci scalar of the standard metric on the 2-sphere:

```
sage: M = Manifold(2, 'S^2', start_index=1)
sage: U = M.open_subset('U') # the complement of a meridian (domain of
↪spherical coordinates)
sage: c_spher.<th,ph> = U.chart(r'th:(0,pi):\theta ph:(0,2*pi):\phi')
sage: a = var('a') # the sphere radius
sage: g = U.metric('g')
sage: g[1,1], g[2,2] = a^2, a^2*sin(th)^2
sage: g.display() # standard metric on the 2-sphere of radius a
g = a^2 dth*dth + a^2*sin(th)^2 dph*dph
sage: g.ricci_scalar()
Scalar field r(g) on the Open subset U of the 2-dimensional
differentiable manifold S^2
sage: g.ricci_scalar().display() # The Ricci scalar is constant:
r(g): U --> R
(th, ph) |--> 2/a^2
```

**set** (*symbiform*)

Define the metric from a field of symmetric bilinear forms.

## INPUT:

- `symbiform` – instance of `TensorFieldParal` representing a field of symmetric bilinear forms

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: s = M.sym_bilin_form_field(name='s')
sage: s[0,0], s[0,1], s[1,1] = 1+x^2, x*y, 1+y^2
sage: g = M.metric('g')
sage: g.set(s)
sage: g.display()
g = (x^2 + 1) dx*dx + x*y dx*dy + x*y dy*dx + (y^2 + 1) dy*dy
```

## 3.2 Levi-Civita Connections

The class `LeviCivitaConnection` implements the Levi-Civita connection associated with some pseudo-Riemannian metric on a smooth manifold.

AUTHORS:

- Ericourgoulhon, Michal Bejger (2013-2015) : initial version
- Marco Mancini (2015) : parallelization of some computations

REFERENCES:

- [KN1963]
- [Lee1997]
- [ONe1983]

**class** `sage.manifolds.differentiable.levi_civita_connection.LeviCivitaConnection` (*metric*, *name*, *la-tex\_name=None*, *init\_coef=True*)

Bases: `sage.manifolds.differentiable.affine_connection.AffineConnection`

Levi-Civita connection on a pseudo-Riemannian manifold.

Let  $M$  be a differentiable manifold of class  $C^\infty$  (smooth manifold) over  $\mathbf{R}$  endowed with with a pseudo-Riemannian metric  $g$ . Let  $C^\infty(M)$  be the algebra of smooth functions  $M \rightarrow \mathbf{R}$  (cf. `DiffScalarFieldAlgebra`) and let  $\mathcal{X}(M)$  be the  $C^\infty(M)$ -module of vector fields on  $M$  (cf. `VectorFieldModule`). The *Levi-Civita connection associated with  $g$*  is the unique operator

$$\begin{aligned} \nabla : \mathcal{X}(M) \times \mathcal{X}(M) &\longrightarrow \mathcal{X}(M) \\ (u, v) &\longmapsto \nabla_u v \end{aligned}$$

that

- is  $\mathbf{R}$ -bilinear, i.e. is bilinear when considering  $\mathcal{X}(M)$  as a vector space over  $\mathbf{R}$
- is  $C^\infty(M)$ -linear w.r.t. the first argument:  $\forall f \in C^\infty(M), \nabla_{fu}v = f\nabla_uv$
- obeys Leibniz rule w.r.t. the second argument:  $\forall f \in C^\infty(M), \nabla_u(fv) = df(u)v + f\nabla_uv$
- is torsion-free
- is compatible with  $g$ :  $\forall (u, v, w) \in \mathcal{X}(M)^3, u(g(v, w)) = g(\nabla_uv, w) + g(v, \nabla_uw)$

The Levi-Civita connection  $\nabla$  gives birth to the *covariant derivative operator* acting on tensor fields, denoted by the same symbol:

$$\begin{array}{ccc} \nabla : T^{(k,l)}(M) & \longrightarrow & T^{(k,l+1)}(M) \\ t & \longmapsto & \nabla t \end{array}$$

where  $T^{(k,l)}(M)$  stands for the  $C^\infty(M)$ -module of tensor fields of type  $(k,l)$  on  $M$  (cf. *TensorFieldModule*), with the convention  $T^{(0,0)}(M) := C^\infty(M)$ . For a vector field  $v$ , the covariant derivative  $\nabla v$  is a type-(1,1) tensor field such that

$$\forall u \in \mathcal{X}(M), \nabla_u v = \nabla v(\cdot, u)$$

More generally for any tensor field  $t \in T^{(k,l)}(M)$ , we have

$$\forall u \in \mathcal{X}(M), \nabla_u t = \nabla t(\dots, u)$$

---

**Note:** The above convention means that, in terms of index notation, the “derivation index” in  $\nabla t$  is the *last* one:

$$\nabla_c t^{a_1 \dots a_k}_{b_1 \dots b_l} = (\nabla t)^{a_1 \dots a_k}_{b_1 \dots b_l c}$$


---

INPUT:

- `metric` – the metric  $g$  defining the Levi-Civita connection, as an instance of class *PseudoRiemannianMetric*
- `name` – name given to the connection
- `latex_name` – (default: None) LaTeX symbol to denote the connection
- `init_coef` – (default: True) determines whether the Christoffel symbols are initialized (in the top charts on the domain, i.e. disregarding the subcharts)

EXAMPLES:

Levi-Civita connection associated with the Euclidean metric on  $\mathbf{R}^3$  expressed in spherical coordinates:

```
sage: forget() # for doctests only
sage: M = Manifold(3, 'R^3', start_index=1)
sage: c_spher.<r,th,ph> = M.chart(r'r:(0,+oo) th:(0,pi):\theta ph:(0,2*pi):\phi')
sage: g = M.metric('g')
sage: g[1,1], g[2,2], g[3,3] = 1, r^2, (r*sin(th))^2
sage: g.display()
g = dr*dr + r^2 dth*dth + r^2*sin(th)^2 dph*dph
sage: nab = g.connection(name='nabla', latex_name=r'\nabla') ; nab
Levi-Civita connection nabla associated with the Riemannian metric g on
the 3-dimensional differentiable manifold R^3
```

Let us check that the connection is compatible with the metric:

```
sage: Dg = nab(g) ; Dg
Tensor field nabla(g) of type (0,3) on the 3-dimensional
differentiable manifold R^3
sage: Dg == 0
True
```

and that it is torsionless:

```
sage: nab.torsion() == 0
True
```

As a check, let us enforce the computation of the torsion:

```
sage: sage.manifolds.differentiable.affine_connection.AffineConnection.
↳torsion(nab) == 0
True
```

The connection coefficients in the manifold's default frame are Christoffel symbols, since the default frame is a coordinate frame:

```
sage: M.default_frame()
Coordinate frame (R^3, (d/dr,d/dth,d/dph))
sage: nab.coef()
3-indices components w.r.t. Coordinate frame (R^3, (d/dr,d/dth,d/dph)),
with symmetry on the index positions (1, 2)
```

We note that the Christoffel symbols are symmetric with respect to their last two indices (positions (1,2)); their expression is:

```
sage: nab.coef()[:] # display as a array
[[[0, 0, 0], [0, -r, 0], [0, 0, -r*sin(th)^2]],
 [[0, 1/r, 0], [1/r, 0, 0], [0, 0, -cos(th)*sin(th)]],
 [[0, 0, 1/r], [0, 0, cos(th)/sin(th)], [1/r, cos(th)/sin(th), 0]]]
sage: nab.display() # display only the non-vanishing symbols
Gam^r_th,th = -r
Gam^r_ph,ph = -r*sin(th)^2
Gam^th_r,th = 1/r
Gam^th_th,r = 1/r
Gam^th_ph,ph = -cos(th)*sin(th)
Gam^ph_r,ph = 1/r
Gam^ph_th,ph = cos(th)/sin(th)
Gam^ph_ph,r = 1/r
Gam^ph_ph,th = cos(th)/sin(th)
sage: nab.display(only_nonredundant=True) # skip redundancy due to symmetry
Gam^r_th,th = -r
Gam^r_ph,ph = -r*sin(th)^2
Gam^th_r,th = 1/r
Gam^th_ph,ph = -cos(th)*sin(th)
Gam^ph_r,ph = 1/r
Gam^ph_th,ph = cos(th)/sin(th)
```

The same display can be obtained via the function `christoffel_symbols_display()` acting on the metric:

```
sage: g.christoffel_symbols_display(chart=c_spher)
Gam^r_th,th = -r
Gam^r_ph,ph = -r*sin(th)^2
Gam^th_r,th = 1/r
Gam^th_ph,ph = -cos(th)*sin(th)
Gam^ph_r,ph = 1/r
Gam^ph_th,ph = cos(th)/sin(th)
```

**coef** (*frame=None*)

Return the connection coefficients relative to the given frame.

$n$  being the manifold's dimension, the connection coefficients relative to the vector frame  $(e_i)$  are the  $n^3$

scalar fields  $\Gamma_{ij}^k$  defined by

$$\nabla_{e_j} e_i = \Gamma_{ij}^k e_k$$

If the connection coefficients are not known already, they are computed

- as Christoffel symbols if the frame  $(e_i)$  is a coordinate frame
- from the above formula otherwise

INPUT:

- `frame` – (default: None) vector frame relative to which the connection coefficients are required; if none is provided, the domain's default frame is assumed

OUTPUT:

- connection coefficients relative to the frame `frame`, as an instance of the class `Components` with 3 indices ordered as  $(k, i, j)$ ; for Christoffel symbols, an instance of the subclass `CompWithSym` is returned.

EXAMPLES:

Christoffel symbols of the Levi-Civita connection associated to the Euclidean metric on  $\mathbf{R}^3$  expressed in spherical coordinates:

```
sage: M = Manifold(3, 'R^3', start_index=1)
sage: c_spher.<r,th,ph> = M.chart(r'r:(0,+oo) th:(0,pi):\theta ph:(0,
↪2*pi):\phi')
sage: g = M.metric('g')
sage: g[1,1], g[2,2], g[3,3] = 1, r^2 , (r*sin(th))^2
sage: g.display()
g = dr*dr + r^2 dth*dth + r^2*sin(th)^2 dph*dph
sage: nab = g.connection()
sage: gam = nab.coef() ; gam
3-indices components w.r.t. Coordinate frame (R^3, (d/dr,d/dth,d/dph)),
with symmetry on the index positions (1, 2)
sage: gam[:]
[[[0, 0, 0], [0, -r, 0], [0, 0, -r*sin(th)^2]],
 [[0, 1/r, 0], [1/r, 0, 0], [0, 0, -cos(th)*sin(th)]],
 [[0, 0, 1/r], [0, 0, cos(th)/sin(th)], [1/r, cos(th)/sin(th), 0]]]
sage: # The only non-zero Christoffel symbols:
sage: gam[1,2,2], gam[1,3,3]
(-r, -r*sin(th)^2)
sage: gam[2,1,2], gam[2,3,3]
(1/r, -cos(th)*sin(th))
sage: gam[3,1,3], gam[3,2,3]
(1/r, cos(th)/sin(th))
```

Connection coefficients of the same connection with respect to the orthonormal frame associated to spherical coordinates:

```
sage: ch_basis = M.automorphism_field()
sage: ch_basis[1,1], ch_basis[2,2], ch_basis[3,3] = 1, 1/r, 1/(r*sin(th))
sage: e = c_spher.frame().new_frame(ch_basis, 'e')
sage: gam_e = nab.coef(e) ; gam_e
3-indices components w.r.t. Vector frame (R^3, (e_1,e_2,e_3))
sage: gam_e[:]
[[[0, 0, 0], [0, -1/r, 0], [0, 0, -1/r]],
 [[0, 1/r, 0], [0, 0, 0], [0, 0, -cos(th)/(r*sin(th))]],
 [[0, 0, 1/r], [0, 0, cos(th)/(r*sin(th))], [0, 0, 0]]]
```

```

sage: # The only non-zero connection coefficients:
sage: gam_e[1,2,2], gam_e[2,1,2]
(-1/r, 1/r)
sage: gam_e[1,3,3], gam_e[3,1,3]
(-1/r, 1/r)
sage: gam_e[2,3,3], gam_e[3,2,3]
(-cos(th)/(r*sin(th)), cos(th)/(r*sin(th)))

```

**restrict** (*subdomain*)

Return the restriction of the connection to some subdomain.

If such restriction has not been defined yet, it is constructed here.

INPUT:

- *subdomain* – open subset  $U$  of the connection’s domain (must be an instance of *DifferentiableManifold*)

OUTPUT:

- instance of *LeviCivitaConnection* representing the restriction.

EXAMPLES:

```

sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: g = M.metric('g')
sage: g[0,0], g[1,1] = 1+y^2, 1+x^2
sage: nab = g.connection()
sage: nab[:]
[[[0, y/(y^2 + 1)], [y/(y^2 + 1), -x/(y^2 + 1)]],
 [[-y/(x^2 + 1), x/(x^2 + 1)], [x/(x^2 + 1), 0]]]
sage: U = M.open_subset('U', coord_def={X: x>0})
sage: nabU = nab.restrict(U); nabU
Levi-Civita connection nabla_g associated with the Riemannian
metric g on the Open subset U of the 2-dimensional differentiable
manifold M
sage: nabU[:]
[[[0, y/(y^2 + 1)], [y/(y^2 + 1), -x/(y^2 + 1)]],
 [[-y/(x^2 + 1), x/(x^2 + 1)], [x/(x^2 + 1), 0]]]

```

Let us check that the restriction is the connection compatible with the restriction of the metric:

```

sage: nabU(g.restrict(U)).display()
nabla_g(g) = 0

```

**ricci** (*name=None, latex\_name=None*)

Return the connection’s Ricci tensor.

This method redefines `sage.manifolds.differentiable.affine_connection.AffineConnection.ricci()` to take into account the symmetry of the Ricci tensor for a Levi-Civita connection.

The Ricci tensor is the tensor field  $Ric$  of type (0,2) defined from the Riemann curvature tensor  $R$  by

$$Ric(u, v) = R(e^i, u, e_i, v)$$

for any vector fields  $u$  and  $v$ ,  $(e_i)$  being any vector frame and  $(e^i)$  the dual coframe.

INPUT:

- `name` – (default: `None`) name given to the Ricci tensor; if none, it is set to “Ric(g)”, where “g” is the metric’s name
- `latex_name` – (default: `None`) LaTeX symbol to denote the Ricci tensor; if none, it is set to “ $\mathrm{Ric}(g)$ ”, where “g” is the metric’s name

OUTPUT:

- the Ricci tensor  $Ric$ , as an instance of `TensorField` of tensor type (0,2) and symmetric

EXAMPLES:

Ricci tensor of the standard connection on the 2-dimensional sphere:

```
sage: M = Manifold(2, 'S^2', start_index=1)
sage: c_spher.<th,ph> = M.chart(r'th:(0,pi):\theta ph:(0,2*pi):\phi')
sage: g = M.metric('g')
sage: g[1,1], g[2,2] = 1, sin(th)^2
sage: g.display() # standard metric on S^2:
g = dth*dth + sin(th)^2 dph*dph
sage: nab = g.connection() ; nab
Levi-Civita connection nabla_g associated with the Riemannian
metric g on the 2-dimensional differentiable manifold S^2
sage: ric = nab.ricci() ; ric
Field of symmetric bilinear forms Ric(g) on the 2-dimensional
differentiable manifold S^2
sage: ric.display()
Ric(g) = dth*dth + sin(th)^2 dph*dph
```

Checking that the Ricci tensor of the Levi-Civita connection associated to Schwarzschild metric is identically zero (as a solution of the Einstein equation):

```
sage: M = Manifold(4, 'M')
sage: c_BL.<t,r,th,ph> = M.chart(r't r:(0,+oo) th:(0,pi):\theta ph:(0,
↔2*pi):\phi') # Schwarzschild-Droste coordinates
sage: g = M.lorentzian_metric('g')
sage: m = var('m') # mass in Schwarzschild metric
sage: g[0,0], g[1,1] = -(1-2*m/r), 1/(1-2*m/r)
sage: g[2,2], g[3,3] = r^2, (r*sin(th))^2
sage: g.display()
g = (2*m/r - 1) dt*dt - 1/(2*m/r - 1) dr*dr + r^2 dth*dth
+ r^2*sin(th)^2 dph*dph
sage: nab = g.connection() ; nab
Levi-Civita connection nabla_g associated with the Lorentzian
metric g on the 4-dimensional differentiable manifold M
sage: ric = nab.ricci() ; ric
Field of symmetric bilinear forms Ric(g) on the 4-dimensional
differentiable manifold M
sage: ric == 0
True
```

**riemann** (*name=None, latex\_name=None*)

Return the Riemann curvature tensor of the connection.

This method redefines `sage.manifolds.differentiable.affine_connection.AffineConnection.riemann()` to set some name and the `latex_name` to the output.

The Riemann curvature tensor is the tensor field  $R$  of type (1,3) defined by

$$R(\omega, w, u, v) = \langle \omega, \nabla_u \nabla_v w - \nabla_v \nabla_u w - \nabla_{[u,v]} w \rangle$$

for any 1-form  $\omega$  and any vector fields  $u, v$  and  $w$ .

INPUT:

- `name` – (default: `None`) name given to the Riemann tensor; if none, it is set to “Riem( $g$ )”, where “ $g$ ” is the metric’s name
- `latex_name` – (default: `None`) LaTeX symbol to denote the Riemann tensor; if none, it is set to “ $\mathrm{Riem}(g)$ ”, where “ $g$ ” is the metric’s name

OUTPUT:

- the Riemann curvature tensor  $R$ , as an instance of *TensorField*

EXAMPLES:

Riemann tensor of the Levi-Civita connection associated with the metric of the hyperbolic plane (Poincaré disk model):

```
sage: M = Manifold(2, 'M', start_index=1)
sage: X.<x,y> = M.chart('x:(-1,1) y:(-1,1)') # Cartesian coord. on the
↔Poincaré disk
sage: X.add_restrictions(x^2+y^2<1)
sage: g = M.metric('g')
sage: g[1,1], g[2,2] = 4/(1-x^2-y^2)^2, 4/(1-x^2-y^2)^2
sage: nab = g.connection()
sage: riem = nab.riemann(); riem
Tensor field Riem(g) of type (1,3) on the 2-dimensional
differentiable manifold M
sage: riem.display_comp()
Riem(g)^x_yxy = -4/(x^4 + y^4 + 2*(x^2 - 1)*y^2 - 2*x^2 + 1)
Riem(g)^x_yyx = 4/(x^4 + y^4 + 2*(x^2 - 1)*y^2 - 2*x^2 + 1)
Riem(g)^y_xxy = 4/(x^4 + y^4 + 2*(x^2 - 1)*y^2 - 2*x^2 + 1)
Riem(g)^y_yxx = -4/(x^4 + y^4 + 2*(x^2 - 1)*y^2 - 2*x^2 + 1)
```

**torsion()**

Return the connection’s torsion tensor (identically zero for a Levi-Civita connection).

See `sage.manifolds.differentiable.affine_connection.AffineConnection.torsion()` for the general definition of the torsion tensor.

OUTPUT:

- the torsion tensor  $T$ , as a vanishing instance of *TensorField*

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: g = M.metric('g')
sage: g[0,0], g[1,1] = 1+y^2, 1+x^2
sage: nab = g.connection()
sage: t = nab.torsion(); t
Tensor field of type (1,2) on the 2-dimensional differentiable
manifold M
```

The torsion of a Levi-Civita connection is always zero:

```
sage: t.display()
0
```

## UTILITIES FOR CALCULUS

This module defines helper functions which are used for simplifications and display of symbolic expressions.

AUTHORS:

- Michal Bejger (2015) : class *ExpressionNice*
- Ericourgoulhon (2015) : simplification functions
- Travis Scrimshaw (2016): review tweaks

**class** `sage.manifolds.utilities.ExpressionNice` (*ex*)

Bases: `sage.symbolic.expression.Expression`

Subclass of `Expression` for a “human-friendly” display of partial derivatives and the possibility to shorten the display by skipping the arguments of symbolic functions.

INPUT:

- *ex* – symbolic expression

EXAMPLES:

An expression formed with callable symbolic expressions:

```

sage: var('x y z')
(x, y, z)
sage: f = function('f')(x, y)
sage: g = f.diff(y).diff(x)
sage: h = function('h')(y, z)
sage: k = h.diff(z)
sage: fun = x*g + y*(k-z)^2
```

The standard Pynac display of partial derivatives:

```

sage: fun
y*(z - diff(h(y, z), z))^2 + x*diff(f(x, y), x, y)
sage: latex(fun)
y {\left(z - \frac{\partial}{\partial z}h\left(y, z\right)\right)}^2 + x \frac{\partial^2}{\partial x \partial y}f\left(x, y\right)
```

With *ExpressionNice*, the Pynac notation `D[...]` is replaced by textbook-like notation:

```

sage: from sage.manifolds.utilities import ExpressionNice
sage: ExpressionNice(fun)
y*(z - d(h)/dz)^2 + x*d^2(f)/dxdy
sage: latex(ExpressionNice(fun))
y {\left(z - \frac{\partial}{\partial h}\right)}^2 + x \frac{\partial^2}{\partial f \partial y}
```

An example when function variables are themselves functions:

```
sage: f = function('f')(x, y)
sage: g = function('g')(x, f) # the second variable is the function f
sage: fun = (g.diff(x))*x - x^2*f.diff(x, y)
sage: fun
-x^2*diff(f(x, y), x, y) + (diff(f(x, y), x)*D[1](g)(x, f(x, y)) + D[0](g)(x, f(x,
→ y))) * x
sage: ExpressionNice(fun)
-x^2*d^2(f)/dxdy + (d(f)/dx*d(g)/d(f(x, y)) + d(g)/dx)*x
sage: latex(ExpressionNice(fun))
-x^{2} \frac{\partial^2 f}{\partial x \partial y}
+ \left( \frac{\partial f}{\partial x} \frac{\partial g}{\partial \left( f \left( x, y \right) \right)} \right)
+ \frac{\partial g}{\partial x} \right) x
```

Note that  $D[1](g)(x, f(x, y))$  is rendered as  $d(g)/d(f(x, y))$ .

An example with multiple differentiations:

```
sage: fun = f.diff(x, x, y, y, x)*x
sage: fun
x*diff(f(x, y), x, x, x, y, y)
sage: ExpressionNice(fun)
x*d^5(f)/dx^3dy^2
sage: latex(ExpressionNice(fun))
x \frac{\partial^5 f}{\partial x^3 \partial y^2}
```

Parentheses are added around powers of partial derivatives to avoid any confusion:

```
sage: fun = f.diff(y)^2
sage: fun
diff(f(x, y), y)^2
sage: ExpressionNice(fun)
(d(f)/dy)^2
sage: latex(ExpressionNice(fun))
\left( \frac{\partial f}{\partial y} \right)^2
```

The explicit mention of function arguments can be omitted for the sake of brevity:

```
sage: fun = fun*f
sage: ExpressionNice(fun)
f(x, y)*(d(f)/dy)^2
sage: Manifold.options.omit_function_arguments=True
sage: ExpressionNice(fun)
f*(d(f)/dy)^2
sage: latex(ExpressionNice(fun))
f \left( \frac{\partial f}{\partial y} \right)^2
sage: Manifold.options._reset()
sage: ExpressionNice(fun)
f(x, y)*(d(f)/dy)^2
sage: latex(ExpressionNice(fun))
f \left( x, y \right) \left( \frac{\partial f}{\partial y} \right)^2
```

`sage.manifolds.utilities.exterior_derivative` (*form*)  
Exterior derivative of a differential form.

INPUT:

- `form` – a differential form; this must be an instance of either

- DiffScalarField* for a 0-form (scalar field)
- DiffFormParal* for a  $p$ -form ( $p \geq 1$ ) on a parallelizable manifold
- DiffForm* for a  $p$ -form ( $p \geq 1$ ) on a non-parallelizable manifold

OUTPUT:

- the  $(p + 1)$ -form that is the exterior derivative of `form`

EXAMPLES:

Exterior derivative of a scalar field (0-form):

```
sage: from sage.manifolds.utilities import exterior_derivative
sage: M = Manifold(3, 'M')
sage: X.<x,y,z> = M.chart()
sage: f = M.scalar_field({X: x+y^2+z^3}, name='f')
sage: df = exterior_derivative(f); df
1-form df on the 3-dimensional differentiable manifold M
sage: df.display()
df = dx + 2*y dy + 3*z^2 dz
```

An alias is `xder`:

```
sage: from sage.manifolds.utilities import xder
sage: df == xder(f)
True
```

Exterior derivative of a 1-form:

```
sage: a = M.one_form(name='a')
sage: a[:] = [x+y*z, x-y*z, x*y*z]
sage: da = xder(a); da
2-form da on the 3-dimensional differentiable manifold M
sage: da.display()
da = (-z + 1) dx/\dy + (y*z - y) dx/\dz + (x*z + y) dy/\dz
sage: dda = xder(da); dda
3-form dda on the 3-dimensional differentiable manifold M
sage: dda.display()
dda = 0
```

See also:

*sage.manifolds.differentiable.diff\_form.DiffFormParal.exterior\_derivative* or *sage.manifolds.differentiable.diff\_form.DiffForm.exterior\_derivative* for more examples.

`sage.manifolds.utilities.set_axes_labels` (*graph*, *xlabel*, *ylabel*, *zlabel*, *\*\*kwds*)

Set axes labels for a 3D graphics object `graph`.

This is a workaround for the lack of axes labels in 3D plots. This sets the labels as `text3d()` objects at locations determined from the bounding box of the graphic object `graph`.

INPUT:

- `graph` – `Graphics3d`; a 3D graphic object
- `xlabel` – string for the x-axis label
- `ylabel` – string for the y-axis label
- `zlabel` – string for the z-axis label

•\*\*kwds – options (e.g. color) for text3d

OUTPUT:

•the 3D graphic object with text3d labels added

EXAMPLES:

```
sage: g = sphere()
sage: g.all
[Graphics3d Object]
sage: from sage.manifolds.utilities import set_axes_labels
sage: ga = set_axes_labels(g, 'X', 'Y', 'Z', color='red')
sage: ga.all # the 3D frame has now axes labels
[Graphics3d Object, Graphics3d Object,
 Graphics3d Object, Graphics3d Object]
```

sage.manifolds.utilities.**simplify\_abs\_trig**(*expr*)  
Simplify  $\text{abs}(\sin(\dots))$  in symbolic expressions.

EXAMPLES:

```
sage: forget() # for doctests only
sage: M = Manifold(3, 'M', structure='topological')
sage: X.<x,y,z> = M.chart(r'x y:(0,pi) z:(-pi/3,0)')
sage: X.coord_range()
x: (-oo, +oo); y: (0, pi); z: (-1/3*pi, 0)
```

Since  $x$  spans all  $\mathbf{R}$ , no simplification of  $\text{abs}(\sin(x))$  occurs, while  $\text{abs}(\sin(y))$  and  $\text{abs}(\sin(3*z))$  are correctly simplified, given that  $y \in (0, \pi)$  and  $z \in (-\pi/3, 0)$ :

```
sage: from sage.manifolds.utilities import simplify_abs_trig
sage: simplify_abs_trig( abs(sin(x)) + abs(sin(y)) + abs(sin(3*z)) )
abs(sin(x)) + sin(y) - sin(3*z)
```

Note that neither Sage's function `simplify_trig()` nor `simplify_full()` works in this case:

```
sage: s = abs(sin(x)) + abs(sin(y)) + abs(sin(3*z))
sage: s.simplify_trig()
abs(4*cos(z)^2 - 1)*abs(sin(z)) + abs(sin(x)) + abs(sin(y))
sage: s.simplify_full()
abs(4*cos(z)^2 - 1)*abs(sin(z)) + abs(sin(x)) + abs(sin(y))
```

despite the following assumptions hold:

```
sage: assumptions()
[x is real, y is real, y > 0, y < pi, z is real, z > -1/3*pi, z < 0]
```

Additional checks are:

```
sage: simplify_abs_trig( abs(sin(y/2)) ) # shall simplify
sin(1/2*y)
sage: simplify_abs_trig( abs(sin(2*y)) ) # must not simplify
abs(sin(2*y))
sage: simplify_abs_trig( abs(sin(z/2)) ) # shall simplify
-sin(1/2*z)
sage: simplify_abs_trig( abs(sin(4*z)) ) # must not simplify
abs(sin(4*z))
```

`sage.manifolds.utilities.simplify_chain_generic(expr)`

Apply a chain of simplifications to a symbolic expression.

This is the simplification chain used in calculus involving coordinate functions on manifolds over fields different from  $\mathbf{R}$ , as implemented in *CoordFunctionSymb*.

The chain is formed by the following functions, called successively:

- 1.simplify\_factorial()
- 2.simplify\_rectform()
- 3.simplify\_trig()
- 4.simplify\_rational()
- 5.expand\_sum()

NB: for the time being, this is identical to `simplify_full()`.

EXAMPLES:

We consider variables that are coordinates of a chart on a complex manifold:

```
sage: forget() # for doctest only
sage: M = Manifold(2, 'M', structure='topological', field='complex')
sage: X.<x,y> = M.chart()
```

Then neither  $x$  nor  $y$  is assumed to be real:

```
sage: assumptions()
[]
```

Accordingly, `simplify_chain_generic` does not simplify `sqrt(x^2)` to `abs(x)`:

```
sage: from sage.manifolds.utilities import simplify_chain_generic
sage: s = sqrt(x^2)
sage: simplify_chain_generic(s)
sqrt(x^2)
```

This contrasts with the behavior of `simplify_chain_real()`.

Other simplifications:

```
sage: s = (x+y)^2 - x^2 - 2*x*y - y^2
sage: simplify_chain_generic(s)
0
sage: s = (x^2 - 2*x + 1) / (x^2 - 1)
sage: simplify_chain_generic(s)
(x - 1)/(x + 1)
sage: s = cos(2*x) - 2*cos(x)^2 + 1
sage: simplify_chain_generic(s)
0
```

`sage.manifolds.utilities.simplify_chain_real(expr)`

Apply a chain of simplifications to a symbolic expression, assuming the real domain.

This is the simplification chain used in calculus involving coordinate functions on real manifolds, as implemented in *CoordFunctionSymb*.

The chain is formed by the following functions, called successively:

- 1.simplify\_factorial()

```

2.simplify_trig()
3.simplify_rational()
4.simplify_sqrt_real()
5.simplify_abs_trig()
6.canonicalize_radical()
7.simplify_log()
8.simplify_rational()
9.simplify_trig()

```

**EXAMPLES:**

We consider variables that are coordinates of a chart on a real manifold:

```

sage: forget() # for doctest only
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart('x:(0,1) y')

```

The following assumptions then hold:

```

sage: assumptions()
[x is real, x > 0, x < 1, y is real]

```

and we have:

```

sage: from sage.manifolds.utilities import simplify_chain_real
sage: s = sqrt(y^2)
sage: simplify_chain_real(s)
abs(y)

```

The above result is correct since  $y$  is real. It is obtained by `simplify_real()` as well, but not by `simplify_full()`:

```

sage: s.simplify_real()
abs(y)
sage: s.simplify_full()
sqrt(y^2)

```

Furthermore, we have:

```

sage: s = sqrt(x^2-2*x+1)
sage: simplify_chain_real(s)
-x + 1

```

which is correct since  $x \in (0, 1)$ . On this example, neither `simplify_real()` nor `simplify_full()`, nor `canonicalize_radical()` give satisfactory results:

```

sage: s.simplify_real() # unsimplified output
sqrt(x^2 - 2*x + 1)
sage: s.simplify_full() # unsimplified output
sqrt(x^2 - 2*x + 1)
sage: s.canonicalize_radical() # wrong output since x in (0,1)
x - 1

```

Other simplifications:

```

sage: s = abs(sin(pi*x))
sage: simplify_chain_real(s) # correct output since x in (0,1)
sin(pi*x)
sage: s.simplify_real() # unsimplified output
abs(sin(pi*x))
sage: s.simplify_full() # unsimplified output
abs(sin(pi*x))

```

```

sage: s = cos(y)^2 + sin(y)^2
sage: simplify_chain_real(s)
1
sage: s.simplify_real() # unsimplified output
cos(y)^2 + sin(y)^2
sage: s.simplify_full() # OK
1

```

`sage.manifolds.utilities.simplify_sqrt_real(expr)`  
Simplify sqrt in symbolic expressions in the real domain.

#### EXAMPLES:

Simplifications of basic expressions:

```

sage: from sage.manifolds.utilities import simplify_sqrt_real
sage: simplify_sqrt_real( sqrt(x^2) )
abs(x)
sage: assume(x<0)
sage: simplify_sqrt_real( sqrt(x^2) )
-x
sage: simplify_sqrt_real( sqrt(x^2-2*x+1) )
-x + 1
sage: simplify_sqrt_real( sqrt(x^2) + sqrt(x^2-2*x+1) )
-2*x + 1

```

This improves over Sage's `canonicalize_radical()`, which yields incorrect results when  $x < 0$ :

```

sage: forget() # removes the assumption x<0
sage: sqrt(x^2).canonicalize_radical()
x
sage: assume(x<0)
sage: sqrt(x^2).canonicalize_radical() # wrong output
x
sage: sqrt(x^2-2*x+1).canonicalize_radical() # wrong output
x - 1
sage: ( sqrt(x^2) + sqrt(x^2-2*x+1) ).canonicalize_radical() # wrong output
2*x - 1

```

Simplification of nested sqrt's:

```

sage: forget() # removes the assumption x<0
sage: simplify_sqrt_real( sqrt(1 + sqrt(x^2)) )
sqrt(abs(x) + 1)
sage: assume(x<0)
sage: simplify_sqrt_real( sqrt(1 + sqrt(x^2)) )
sqrt(-x + 1)
sage: simplify_sqrt_real( sqrt(x^2 + sqrt(4*x^2) + 1) )
-x + 1

```

Again, `canonicalize_radical()` fails on the last one:

```
sage: (sqrt(x^2 + sqrt(4*x^2) + 1)).canonicalize_radical() # wrong output
x + 1
```

`sage.manifolds.utilities.xder` (*form*)

Exterior derivative of a differential form.

INPUT:

- `form` – a differential form; this must be an instance of either
  - `DiffScalarField` for a 0-form (scalar field)
  - `DiffFormParal` for a  $p$ -form ( $p \geq 1$ ) on a parallelizable manifold
  - `DiffForm` for a  $p$ -form ( $p \geq 1$ ) on a non-parallelizable manifold

OUTPUT:

- the  $(p + 1)$ -form that is the exterior derivative of `form`

EXAMPLES:

Exterior derivative of a scalar field (0-form):

```
sage: from sage.manifolds.utilities import exterior_derivative
sage: M = Manifold(3, 'M')
sage: X.<x,y,z> = M.chart()
sage: f = M.scalar_field({X: x+y^2+z^3}, name='f')
sage: df = exterior_derivative(f); df
1-form df on the 3-dimensional differentiable manifold M
sage: df.display()
df = dx + 2*y dy + 3*z^2 dz
```

An alias is `xder`:

```
sage: from sage.manifolds.utilities import xder
sage: df == xder(f)
True
```

Exterior derivative of a 1-form:

```
sage: a = M.one_form(name='a')
sage: a[:] = [x+y*z, x-y*z, x*y*z]
sage: da = xder(a); da
2-form da on the 3-dimensional differentiable manifold M
sage: da.display()
da = (-z + 1) dx/\dy + (y*z - y) dx/\dz + (x*z + y) dy/\dz
sage: dda = xder(da); dda
3-form dda on the 3-dimensional differentiable manifold M
sage: dda.display()
dda = 0
```

See also:

`sage.manifolds.differentiable.diff_form.DiffFormParal.exterior_derivative`  
or `sage.manifolds.differentiable.diff_form.DiffForm.exterior_derivative` for  
more examples.

## INDICES AND TABLES

- Index
- Module Index
- Search Page



## PYTHON MODULE INDEX

### m

- sage.manifolds.chart, 58
- sage.manifolds.continuous\_map, 150
- sage.manifolds.coord\_func, 92
- sage.manifolds.coord\_func\_symb, 99
- sage.manifolds.differentiable.affine\_connection, 435
- sage.manifolds.differentiable.automorphismfield, 351
- sage.manifolds.differentiable.automorphismfield\_group, 345
- sage.manifolds.differentiable.chart, 210
- sage.manifolds.differentiable.curve, 269
- sage.manifolds.differentiable.diff\_form, 422
- sage.manifolds.differentiable.diff\_form\_module, 414
- sage.manifolds.differentiable.diff\_map, 258
- sage.manifolds.differentiable.levi\_civita\_connection, 480
- sage.manifolds.differentiable.manifold, 175
- sage.manifolds.differentiable.manifold\_homset, 254
- sage.manifolds.differentiable.metric, 455
- sage.manifolds.differentiable.real\_line, 223
- sage.manifolds.differentiable.scalarfield, 239
- sage.manifolds.differentiable.scalarfield\_algebra, 233
- sage.manifolds.differentiable.tangent\_space, 282
- sage.manifolds.differentiable.tangent\_vector, 285
- sage.manifolds.differentiable.tensorfield, 365
- sage.manifolds.differentiable.tensorfield\_module, 359
- sage.manifolds.differentiable.tensorfield\_paral, 394
- sage.manifolds.differentiable.vectorfield, 312
- sage.manifolds.differentiable.vectorfield\_module, 294
- sage.manifolds.differentiable.vectorframe, 330
- sage.manifolds.manifold, 3
- sage.manifolds.manifold\_homset, 148
- sage.manifolds.point, 44
- sage.manifolds.scalarfield, 122
- sage.manifolds.scalarfield\_algebra, 116
- sage.manifolds.structure, 42
- sage.manifolds.subset, 31
- sage.manifolds.utilities, 487



## A

add\_coef() (sage.manifolds.differentiable.affine\_connection.AffineConnection method), 439  
 add\_comp() (sage.manifolds.differentiable.tensorfield.TensorField method), 369  
 add\_comp() (sage.manifolds.differentiable.tensorfield\_paral.TensorFieldParal method), 403  
 add\_comp\_by\_continuation() (sage.manifolds.differentiable.tensorfield.TensorField method), 370  
 add\_coord() (sage.manifolds.point.ManifoldPoint method), 46  
 add\_coordinates() (sage.manifolds.point.ManifoldPoint method), 47  
 add\_expr() (sage.manifolds.continuous\_map.ContinuousMap method), 156  
 add\_expr() (sage.manifolds.scalarfield.ScalarField method), 131  
 add\_expr\_by\_continuation() (sage.manifolds.scalarfield.ScalarField method), 132  
 add\_expression() (sage.manifolds.continuous\_map.ContinuousMap method), 158  
 add\_restrictions() (sage.manifolds.chart.Chart method), 61  
 add\_restrictions() (sage.manifolds.chart.RealChart method), 75  
 affine\_connection() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 184  
 AffineConnection (class in sage.manifolds.differentiable.affine\_connection), 435  
 along() (sage.manifolds.differentiable.vectorframe.VectorFrame method), 338  
 alternating\_form() (sage.manifolds.differentiable.vectorfield\_module.VectorFieldModule method), 306  
 ambient() (sage.manifolds.subset.ManifoldSubset method), 33  
 ambient\_domain() (sage.manifolds.differentiable.vectorfield\_module.VectorFieldFreeModule method), 298  
 ambient\_domain() (sage.manifolds.differentiable.vectorfield\_module.VectorFieldModule method), 306  
 ambient\_domain() (sage.manifolds.differentiable.vectorframe.VectorFrame method), 339  
 antisymmetrize() (sage.manifolds.differentiable.tensorfield.TensorField method), 371  
 arccos() (sage.manifolds.coord\_func.CoordFunction method), 93  
 arccos() (sage.manifolds.coord\_func\_symb.CoordFunctionSymb method), 103  
 arccos() (sage.manifolds.scalarfield.ScalarField method), 133  
 arccosh() (sage.manifolds.coord\_func.CoordFunction method), 93  
 arccosh() (sage.manifolds.coord\_func\_symb.CoordFunctionSymb method), 104  
 arccosh() (sage.manifolds.scalarfield.ScalarField method), 133  
 arcsin() (sage.manifolds.coord\_func.CoordFunction method), 93  
 arcsin() (sage.manifolds.coord\_func\_symb.CoordFunctionSymb method), 104  
 arcsin() (sage.manifolds.scalarfield.ScalarField method), 134  
 arcsinh() (sage.manifolds.coord\_func.CoordFunction method), 93  
 arcsinh() (sage.manifolds.coord\_func\_symb.CoordFunctionSymb method), 104  
 arcsinh() (sage.manifolds.scalarfield.ScalarField method), 134  
 arctan() (sage.manifolds.coord\_func.CoordFunction method), 93  
 arctan() (sage.manifolds.coord\_func\_symb.CoordFunctionSymb method), 105  
 arctan() (sage.manifolds.scalarfield.ScalarField method), 135

`arctanh()` (sage.manifolds.coord\_func.CoordFunction method), 93  
`arctanh()` (sage.manifolds.coord\_func\_symb.CoordFunctionSymb method), 105  
`arctanh()` (sage.manifolds.scalarfield.ScalarField method), 135  
`at()` (sage.manifolds.differentiable.automorphismfield.AutomorphismFieldParal method), 356  
`at()` (sage.manifolds.differentiable.tensorfield.TensorField method), 371  
`at()` (sage.manifolds.differentiable.tensorfield\_paral.TensorFieldParal method), 404  
`at()` (sage.manifolds.differentiable.vectorframe.CoFrame method), 334  
`at()` (sage.manifolds.differentiable.vectorframe.VectorFrame method), 339  
`atlas()` (sage.manifolds.manifold.TopologicalManifold method), 13  
`automorphism()` (sage.manifolds.differentiable.vectorfield\_module.VectorFieldModule method), 307  
`automorphism_field()` (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 185  
`automorphism_field_group()` (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 186  
`AutomorphismField` (class in sage.manifolds.differentiable.automorphismfield), 351  
`AutomorphismFieldGroup` (class in sage.manifolds.differentiable.automorphismfield\_group), 345  
`AutomorphismFieldParal` (class in sage.manifolds.differentiable.automorphismfield), 355  
`AutomorphismFieldParalGroup` (class in sage.manifolds.differentiable.automorphismfield\_group), 347

## B

`base_field()` (sage.manifolds.manifold.TopologicalManifold method), 14  
`base_field_type()` (sage.manifolds.manifold.TopologicalManifold method), 14  
`base_module()` (sage.manifolds.differentiable.automorphismfield\_group.AutomorphismFieldGroup method), 346  
`base_module()` (sage.manifolds.differentiable.diff\_form\_module.DiffFormModule method), 421  
`base_module()` (sage.manifolds.differentiable.tensorfield.TensorField method), 372  
`base_module()` (sage.manifolds.differentiable.tensorfield\_module.TensorFieldModule method), 364  
`base_point()` (sage.manifolds.differentiable.tangent\_space.TangentSpace method), 284  
`basis()` (sage.manifolds.differentiable.vectorfield\_module.VectorFieldFreeModule method), 298  
`bracket()` (sage.manifolds.differentiable.vectorfield.VectorField method), 315

## C

`canonical_chart()` (sage.manifolds.differentiable.real\_line.OpenInterval method), 228  
`canonical_coordinate()` (sage.manifolds.differentiable.real\_line.OpenInterval method), 228  
`change_of_frame()` (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 186  
`changes_of_frame()` (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 187  
`Chart` (class in sage.manifolds.chart), 58  
`chart` (sage.manifolds.structure.DifferentialStructure attribute), 42  
`chart` (sage.manifolds.structure.RealDifferentialStructure attribute), 42  
`chart` (sage.manifolds.structure.RealTopologicalStructure attribute), 43  
`chart` (sage.manifolds.structure.TopologicalStructure attribute), 43  
`chart()` (sage.manifolds.coord\_func.CoordFunction method), 93  
`chart()` (sage.manifolds.coord\_func.MultiCoordFunction method), 97  
`chart()` (sage.manifolds.differentiable.vectorframe.CoordFrame method), 336  
`chart()` (sage.manifolds.manifold.TopologicalManifold method), 15  
`christoffel_symbols()` (sage.manifolds.differentiable.metric.PseudoRiemannianMetric method), 459  
`christoffel_symbols_display()` (sage.manifolds.differentiable.metric.PseudoRiemannianMetric method), 460  
`coef()` (sage.manifolds.differentiable.affine\_connection.AffineConnection method), 440  
`coef()` (sage.manifolds.differentiable.levi\_civita\_connection.LeviCivitaConnection method), 482  
`CoFrame` (class in sage.manifolds.differentiable.vectorframe), 333  
`coframe()` (sage.manifolds.differentiable.chart.DiffChart method), 213  
`coframe()` (sage.manifolds.differentiable.vectorframe.VectorFrame method), 341  
`coframes()` (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 188

`collect()` (sage.manifolds.coord\_func\_symb.CoordFunctionSymb method), 105  
`collect_common_factors()` (sage.manifolds.coord\_func\_symb.CoordFunctionSymb method), 106  
`common_charts()` (sage.manifolds.scalarfield.ScalarField method), 136  
`comp()` (sage.manifolds.differentiable.tensorfield.TensorField method), 373  
`comp()` (sage.manifolds.differentiable.tensorfield\_paral.TensorFieldParal method), 406  
`connection()` (sage.manifolds.differentiable.metric.PseudoRiemannianMetric method), 462  
`connection_form()` (sage.manifolds.differentiable.affine\_connection.AffineConnection method), 441  
`constant_scalar_field()` (sage.manifolds.manifold.TopologicalManifold method), 16  
`continuous_map()` (sage.manifolds.manifold.TopologicalManifold method), 17  
`ContinuousMap` (class in sage.manifolds.continuous\_map), 151  
`contract()` (sage.manifolds.differentiable.tensorfield.TensorField method), 374  
`contract()` (sage.manifolds.differentiable.tensorfield\_paral.TensorFieldParal method), 406  
`coord()` (sage.manifolds.point.ManifoldPoint method), 48  
`coord_bounds()` (sage.manifolds.chart.RealChart method), 75  
`coord_change()` (sage.manifolds.manifold.TopologicalManifold method), 18  
`coord_changes()` (sage.manifolds.manifold.TopologicalManifold method), 19  
`coord_expr()` (sage.manifolds.differentiable.curve.DifferentiableCurve method), 274  
`coord_function()` (sage.manifolds.scalarfield.ScalarField method), 138  
`coord_functions()` (sage.manifolds.continuous\_map.ContinuousMap method), 160  
`coord_range()` (sage.manifolds.chart.RealChart method), 76  
`CoordChange` (class in sage.manifolds.chart), 68  
`CoordCoFrame` (class in sage.manifolds.differentiable.vectorframe), 334  
`CoordFrame` (class in sage.manifolds.differentiable.vectorframe), 335  
`CoordFunction` (class in sage.manifolds.coord\_func), 92  
`CoordFunctionSymb` (class in sage.manifolds.coord\_func\_symb), 100  
`CoordFunctionSymbRing` (class in sage.manifolds.coord\_func\_symb), 114  
`coordinates()` (sage.manifolds.point.ManifoldPoint method), 49  
`copy()` (sage.manifolds.coord\_func.CoordFunction method), 94  
`copy()` (sage.manifolds.coord\_func\_symb.CoordFunctionSymb method), 106  
`copy()` (sage.manifolds.differentiable.tensorfield.TensorField method), 377  
`copy()` (sage.manifolds.scalarfield.ScalarField method), 139  
`cos()` (sage.manifolds.coord\_func.CoordFunction method), 94  
`cos()` (sage.manifolds.coord\_func\_symb.CoordFunctionSymb method), 107  
`cos()` (sage.manifolds.scalarfield.ScalarField method), 139  
`cosh()` (sage.manifolds.coord\_func.CoordFunction method), 94  
`cosh()` (sage.manifolds.coord\_func\_symb.CoordFunctionSymb method), 107  
`cosh()` (sage.manifolds.scalarfield.ScalarField method), 139  
`cotton()` (sage.manifolds.differentiable.metric.PseudoRiemannianMetric method), 463  
`cotton_york()` (sage.manifolds.differentiable.metric.PseudoRiemannianMetric method), 464  
`curvature_form()` (sage.manifolds.differentiable.affine\_connection.AffineConnection method), 443  
`curve()` (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 188

## D

`declare_union()` (sage.manifolds.subset.ManifoldSubset method), 33  
`default_chart()` (sage.manifolds.manifold.TopologicalManifold method), 20  
`default_frame()` (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 190  
`degree()` (sage.manifolds.differentiable.diff\_form.DiffForm method), 424  
`degree()` (sage.manifolds.differentiable.diff\_form\_module.DiffFormModule method), 421  
`del_other_coef()` (sage.manifolds.differentiable.affine\_connection.AffineConnection method), 444  
`destination_map()` (sage.manifolds.differentiable.vectorfield\_module.VectorFieldFreeModule method), 299

`destination_map()` (`sage.manifolds.differentiable.vectorfield_module.VectorFieldModule` method), 307

`destination_map()` (`sage.manifolds.differentiable.vectorframe.VectorFrame` method), 341

`determinant()` (`sage.manifolds.differentiable.metric.PseudoRiemannianMetric` method), 464

`diff()` (`sage.manifolds.coord_func.CoordFunction` method), 94

`diff()` (`sage.manifolds.coord_func_symb.CoordFunctionSymb` method), 107

`diff_degree()` (`sage.manifolds.differentiable.manifold.DifferentiableManifold` method), 190

`diff_form()` (`sage.manifolds.differentiable.manifold.DifferentiableManifold` method), 190

`diff_form_module()` (`sage.manifolds.differentiable.manifold.DifferentiableManifold` method), 191

`diff_map()` (`sage.manifolds.differentiable.manifold.DifferentiableManifold` method), 192

`diff_mapping()` (`sage.manifolds.differentiable.manifold.DifferentiableManifold` method), 193

`DiffChart` (class in `sage.manifolds.differentiable.chart`), 210

`DiffCoordChange` (class in `sage.manifolds.differentiable.chart`), 217

`diffeomorphism()` (`sage.manifolds.differentiable.manifold.DifferentiableManifold` method), 193

`DifferentiableCurve` (class in `sage.manifolds.differentiable.curve`), 270

`DifferentiableCurveSet` (class in `sage.manifolds.differentiable.manifold_homset`), 254

`DifferentiableManifold` (class in `sage.manifolds.differentiable.manifold`), 182

`DifferentiableManifoldHomset` (class in `sage.manifolds.differentiable.manifold_homset`), 256

`differential()` (`sage.manifolds.differentiable.diff_map.DiffMap` method), 264

`differential()` (`sage.manifolds.differentiable.scalarfield.DiffScalarField` method), 248

`differential_functions()` (`sage.manifolds.differentiable.diff_map.DiffMap` method), 265

`DifferentialStructure` (class in `sage.manifolds.structure`), 42

`DiffForm` (class in `sage.manifolds.differentiable.diff_form`), 422

`DiffFormFreeModule` (class in `sage.manifolds.differentiable.diff_form_module`), 414

`DiffFormModule` (class in `sage.manifolds.differentiable.diff_form_module`), 417

`DiffFormParal` (class in `sage.manifolds.differentiable.diff_form`), 428

`DiffMap` (class in `sage.manifolds.differentiable.diff_map`), 258

`DiffScalarField` (class in `sage.manifolds.differentiable.scalarfield`), 239

`DiffScalarFieldAlgebra` (class in `sage.manifolds.differentiable.scalarfield_algebra`), 233

`dim()` (`sage.manifolds.differentiable.tangent_space.TangentSpace` method), 284

`dim()` (`sage.manifolds.manifold.TopologicalManifold` method), 20

`dimension()` (`sage.manifolds.differentiable.tangent_space.TangentSpace` method), 285

`dimension()` (`sage.manifolds.manifold.TopologicalManifold` method), 21

`disp()` (`sage.manifolds.chart.CoordChange` method), 68

`disp()` (`sage.manifolds.continuous_map.ContinuousMap` method), 161

`disp()` (`sage.manifolds.coord_func.CoordFunction` method), 94

`disp()` (`sage.manifolds.coord_func_symb.CoordFunctionSymb` method), 108

`disp()` (`sage.manifolds.differentiable.tensorfield.TensorField` method), 377

`disp()` (`sage.manifolds.scalarfield.ScalarField` method), 140

`display()` (`sage.manifolds.chart.CoordChange` method), 69

`display()` (`sage.manifolds.continuous_map.ContinuousMap` method), 163

`display()` (`sage.manifolds.coord_func.CoordFunction` method), 94

`display()` (`sage.manifolds.coord_func_symb.CoordFunctionSymb` method), 109

`display()` (`sage.manifolds.differentiable.affine_connection.AffineConnection` method), 444

`display()` (`sage.manifolds.differentiable.tensorfield.TensorField` method), 378

`display()` (`sage.manifolds.scalarfield.ScalarField` method), 140

`display_comp()` (`sage.manifolds.differentiable.tensorfield.TensorField` method), 379

`display_comp()` (`sage.manifolds.differentiable.tensorfield_paral.TensorFieldParal` method), 407

`domain()` (`sage.manifolds.chart.Chart` method), 61

`domain()` (`sage.manifolds.differentiable.affine_connection.AffineConnection` method), 446

`domain()` (`sage.manifolds.differentiable.tensorfield.TensorField` method), 380

domain() (sage.manifolds.differentiable.vectorfield\_module.VectorFieldFreeModule method), 299  
 domain() (sage.manifolds.differentiable.vectorfield\_module.VectorFieldModule method), 308  
 domain() (sage.manifolds.differentiable.vectorframe.VectorFrame method), 342  
 domain() (sage.manifolds.scalarfield.ScalarField method), 141  
 down() (sage.manifolds.differentiable.tensorfield.TensorField method), 381  
 dual() (sage.manifolds.differentiable.vectorfield\_module.VectorFieldModule method), 308  
 dual\_exterior\_power() (sage.manifolds.differentiable.vectorfield\_module.VectorFieldFreeModule method), 300  
 dual\_exterior\_power() (sage.manifolds.differentiable.vectorfield\_module.VectorFieldModule method), 308

## E

Element (sage.manifolds.coord\_func\_symb.CoordFunctionSymbRing attribute), 114  
 Element (sage.manifolds.differentiable.automorphismfield\_group.AutomorphismFieldGroup attribute), 346  
 Element (sage.manifolds.differentiable.automorphismfield\_group.AutomorphismFieldParalGroup attribute), 350  
 Element (sage.manifolds.differentiable.diff\_form\_module.DiffFormFreeModule attribute), 417  
 Element (sage.manifolds.differentiable.diff\_form\_module.DiffFormModule attribute), 421  
 Element (sage.manifolds.differentiable.manifold\_homset.DifferentiableCurveSet attribute), 256  
 Element (sage.manifolds.differentiable.manifold\_homset.DifferentiableManifoldHomset attribute), 258  
 Element (sage.manifolds.differentiable.scalarfield\_algebra.DiffScalarFieldAlgebra attribute), 239  
 Element (sage.manifolds.differentiable.tangent\_space.TangentSpace attribute), 284  
 Element (sage.manifolds.differentiable.tensorfield\_module.TensorFieldFreeModule attribute), 361  
 Element (sage.manifolds.differentiable.tensorfield\_module.TensorFieldModule attribute), 364  
 Element (sage.manifolds.differentiable.vectorfield\_module.VectorFieldFreeModule attribute), 298  
 Element (sage.manifolds.differentiable.vectorfield\_module.VectorFieldModule attribute), 306  
 Element (sage.manifolds.manifold\_homset.TopologicalManifoldHomset attribute), 149  
 Element (sage.manifolds.scalarfield\_algebra.ScalarFieldAlgebra attribute), 121  
 Element (sage.manifolds.subset.ManifoldSubset attribute), 33  
 exp() (sage.manifolds.coord\_func.CoordFunction method), 94  
 exp() (sage.manifolds.coord\_func\_symb.CoordFunctionSymb method), 109  
 exp() (sage.manifolds.scalarfield.ScalarField method), 141  
 expand() (sage.manifolds.coord\_func\_symb.CoordFunctionSymb method), 109  
 expr() (sage.manifolds.continuous\_map.ContinuousMap method), 164  
 expr() (sage.manifolds.coord\_func.CoordFunction method), 94  
 expr() (sage.manifolds.coord\_func.MultiCoordFunction method), 97  
 expr() (sage.manifolds.coord\_func\_symb.CoordFunctionSymb method), 110  
 expr() (sage.manifolds.scalarfield.ScalarField method), 142  
 expression() (sage.manifolds.continuous\_map.ContinuousMap method), 166  
 ExpressionNice (class in sage.manifolds.utilities), 487  
 exterior\_derivative() (in module sage.manifolds.utilities), 488  
 exterior\_derivative() (sage.manifolds.differentiable.diff\_form.DiffForm method), 425  
 exterior\_derivative() (sage.manifolds.differentiable.diff\_form.DiffFormParal method), 432  
 exterior\_derivative() (sage.manifolds.differentiable.scalarfield.DiffScalarField method), 249

## F

factor() (sage.manifolds.coord\_func\_symb.CoordFunctionSymb method), 111  
 frame() (sage.manifolds.differentiable.chart.DiffChart method), 214  
 frames() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 194  
 from\_base\_ring() (sage.manifolds.coord\_func\_symb.CoordFunctionSymbRing method), 114  
 function() (sage.manifolds.chart.Chart method), 62  
 function\_chart() (sage.manifolds.scalarfield.ScalarField method), 143  
 function\_ring() (sage.manifolds.chart.Chart method), 62

## G

`general_linear_group()` (`sage.manifolds.differentiable.vectorfield_module.VectorFieldFreeModule` method), 300  
`general_linear_group()` (`sage.manifolds.differentiable.vectorfield_module.VectorFieldModule` method), 309  
`get_chart()` (`sage.manifolds.manifold.TopologicalManifold` method), 21  
`get_subset()` (`sage.manifolds.subset.ManifoldSubset` method), 33  
`global_options()` (`sage.manifolds.manifold.TopologicalManifold` method), 22

## H

`hodge_dual()` (`sage.manifolds.differentiable.diff_form.DiffForm` method), 426  
`hodge_dual()` (`sage.manifolds.differentiable.diff_form.DiffFormParal` method), 433  
`hodge_dual()` (`sage.manifolds.differentiable.scalarfield.DiffScalarField` method), 251  
`hodge_star()` (`sage.manifolds.differentiable.metric.PseudoRiemannianMetric` method), 465  
`homeomorphism()` (`sage.manifolds.manifold.TopologicalManifold` method), 22  
`homset` (`sage.manifolds.structure.DifferentialStructure` attribute), 42  
`homset` (`sage.manifolds.structure.RealDifferentialStructure` attribute), 42  
`homset` (`sage.manifolds.structure.RealTopologicalStructure` attribute), 43  
`homset` (`sage.manifolds.structure.TopologicalStructure` attribute), 43

## I

`identity_map()` (`sage.manifolds.differentiable.vectorfield_module.VectorFieldModule` method), 309  
`identity_map()` (`sage.manifolds.manifold.TopologicalManifold` method), 23  
`index_generator()` (`sage.manifolds.manifold.TopologicalManifold` method), 23  
`inf()` (`sage.manifolds.differentiable.real_line.OpenInterval` method), 229  
`intersection()` (`sage.manifolds.subset.ManifoldSubset` method), 34  
`inverse()` (`sage.manifolds.chart.CoordChange` method), 69  
`inverse()` (`sage.manifolds.continuous_map.ContinuousMap` method), 167  
`inverse()` (`sage.manifolds.differentiable.automorphismfield.AutomorphismField` method), 352  
`inverse()` (`sage.manifolds.differentiable.automorphismfield.AutomorphismFieldParal` method), 357  
`inverse()` (`sage.manifolds.differentiable.metric.PseudoRiemannianMetric` method), 468  
`inverse()` (`sage.manifolds.differentiable.metric.PseudoRiemannianMetricParal` method), 478  
`irange()` (`sage.manifolds.manifold.TopologicalManifold` method), 24  
`is_field()` (`sage.manifolds.coord_func_symb.CoordFunctionSymbRing` method), 115  
`is_identity()` (`sage.manifolds.continuous_map.ContinuousMap` method), 168  
`is_integral_domain()` (`sage.manifolds.coord_func_symb.CoordFunctionSymbRing` method), 115  
`is_manifestly_coordinate_domain()` (`sage.manifolds.manifold.TopologicalManifold` method), 24  
`is_manifestly_parallelizable()` (`sage.manifolds.differentiable.manifold.DifferentiableManifold` method), 195  
`is_open()` (`sage.manifolds.manifold.TopologicalManifold` method), 25  
`is_open()` (`sage.manifolds.subset.ManifoldSubset` method), 35  
`is_subset()` (`sage.manifolds.subset.ManifoldSubset` method), 35

## J

`jacobian()` (`sage.manifolds.coord_func.MultiCoordFunction` method), 98  
`jacobian()` (`sage.manifolds.differentiable.chart.DiffCoordChange` method), 217  
`jacobian_det()` (`sage.manifolds.coord_func.MultiCoordFunction` method), 98  
`jacobian_det()` (`sage.manifolds.differentiable.chart.DiffCoordChange` method), 218  
`jacobian_matrix()` (`sage.manifolds.differentiable.diff_map.DiffMap` method), 267

## L

`LeviCivitaConnection` (class in `sage.manifolds.differentiable.levi_civita_connection`), 480

lie\_der() (sage.manifolds.differentiable.scalarfield.DiffScalarField method), 251  
 lie\_der() (sage.manifolds.differentiable.tensorfield.TensorField method), 383  
 lie\_der() (sage.manifolds.differentiable.tensorfield\_paral.TensorFieldParal method), 409  
 lie\_derivative() (sage.manifolds.differentiable.scalarfield.DiffScalarField method), 252  
 lie\_derivative() (sage.manifolds.differentiable.tensorfield.TensorField method), 384  
 lie\_derivative() (sage.manifolds.differentiable.tensorfield\_paral.TensorFieldParal method), 411  
 lift() (sage.manifolds.subset.ManifoldSubset method), 35  
 linear\_form() (sage.manifolds.differentiable.vectorfield\_module.VectorFieldModule method), 310  
 list\_of\_subsets() (sage.manifolds.subset.ManifoldSubset method), 36  
 log() (sage.manifolds.coord\_func.CoordFunction method), 94  
 log() (sage.manifolds.coord\_func\_symb.CoordFunctionSymb method), 111  
 log() (sage.manifolds.scalarfield.ScalarField method), 143  
 lorentz\_metric() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 195  
 lorentzian\_metric() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 196  
 lower\_bound() (sage.manifolds.differentiable.real\_line.OpenInterval method), 229

## M

Manifold() (in module sage.manifolds.manifold), 8  
 manifold() (sage.manifolds.chart.Chart method), 63  
 manifold() (sage.manifolds.subset.ManifoldSubset method), 36  
 ManifoldPoint (class in sage.manifolds.point), 44  
 ManifoldSubset (class in sage.manifolds.subset), 31  
 metric() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 196  
 metric() (sage.manifolds.differentiable.vectorfield\_module.VectorFieldFreeModule method), 301  
 metric() (sage.manifolds.differentiable.vectorfield\_module.VectorFieldModule method), 310  
 MultiCoordFunction (class in sage.manifolds.coord\_func), 96  
 multifunction() (sage.manifolds.chart.Chart method), 63

## N

new\_frame() (sage.manifolds.differentiable.vectorframe.VectorFrame method), 342

## O

one() (sage.manifolds.coord\_func\_symb.CoordFunctionSymbRing method), 115  
 one() (sage.manifolds.differentiable.automorphismfield\_group.AutomorphismFieldGroup method), 347  
 one() (sage.manifolds.manifold\_homset.TopologicalManifoldHomset method), 150  
 one() (sage.manifolds.scalarfield\_algebra.ScalarFieldAlgebra method), 121  
 one\_form() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 197  
 one\_function() (sage.manifolds.chart.Chart method), 63  
 one\_scalar\_field() (sage.manifolds.manifold.TopologicalManifold method), 25  
 open\_covers() (sage.manifolds.subset.ManifoldSubset method), 37  
 open\_interval() (sage.manifolds.differentiable.real\_line.OpenInterval method), 229  
 open\_subset() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 198  
 open\_subset() (sage.manifolds.manifold.TopologicalManifold method), 25  
 OpenInterval (class in sage.manifolds.differentiable.real\_line), 223

## P

plot() (sage.manifolds.chart.RealChart method), 77  
 plot() (sage.manifolds.differentiable.curve.DifferentiableCurve method), 274  
 plot() (sage.manifolds.differentiable.tangent\_vector.TangentVector method), 286  
 plot() (sage.manifolds.differentiable.vectorfield.VectorField method), 315

plot() (sage.manifolds.point.ManifoldPoint method), 51  
 point() (sage.manifolds.subset.ManifoldSubset method), 37  
 PseudoRiemannianMetric (class in sage.manifolds.differentiable.metric), 455  
 PseudoRiemannianMetricParal (class in sage.manifolds.differentiable.metric), 476  
 pullback() (sage.manifolds.differentiable.diff\_map.DiffMap method), 267  
 pushforward() (sage.manifolds.differentiable.diff\_map.DiffMap method), 268

## R

RealChart (class in sage.manifolds.chart), 71  
 RealDiffChart (class in sage.manifolds.differentiable.chart), 218  
 RealDifferentialStructure (class in sage.manifolds.structure), 42  
 RealLine (class in sage.manifolds.differentiable.real\_line), 231  
 RealTopologicalStructure (class in sage.manifolds.structure), 43  
 restrict() (sage.manifolds.chart.Chart method), 64  
 restrict() (sage.manifolds.chart.CoordChange method), 70  
 restrict() (sage.manifolds.chart.RealChart method), 90  
 restrict() (sage.manifolds.continuous\_map.ContinuousMap method), 169  
 restrict() (sage.manifolds.differentiable.affine\_connection.AffineConnection method), 447  
 restrict() (sage.manifolds.differentiable.automorphismfield.AutomorphismField method), 353  
 restrict() (sage.manifolds.differentiable.automorphismfield.AutomorphismFieldParal method), 358  
 restrict() (sage.manifolds.differentiable.chart.DiffChart method), 215  
 restrict() (sage.manifolds.differentiable.chart.RealDiffChart method), 222  
 restrict() (sage.manifolds.differentiable.levi\_civita\_connection.LeviCivitaConnection method), 484  
 restrict() (sage.manifolds.differentiable.metric.PseudoRiemannianMetric method), 468  
 restrict() (sage.manifolds.differentiable.metric.PseudoRiemannianMetricParal method), 478  
 restrict() (sage.manifolds.differentiable.tensorfield.TensorField method), 385  
 restrict() (sage.manifolds.differentiable.tensorfield\_paral.TensorFieldParal method), 412  
 restrict() (sage.manifolds.differentiable.vectorframe.VectorFrame method), 343  
 restrict() (sage.manifolds.scalarfield.ScalarField method), 143  
 retract() (sage.manifolds.subset.ManifoldSubset method), 38  
 ricci() (sage.manifolds.differentiable.affine\_connection.AffineConnection method), 448  
 ricci() (sage.manifolds.differentiable.levi\_civita\_connection.LeviCivitaConnection method), 484  
 ricci() (sage.manifolds.differentiable.metric.PseudoRiemannianMetric method), 469  
 ricci\_scalar() (sage.manifolds.differentiable.metric.PseudoRiemannianMetric method), 470  
 ricci\_scalar() (sage.manifolds.differentiable.metric.PseudoRiemannianMetricParal method), 479  
 riemann() (sage.manifolds.differentiable.affine\_connection.AffineConnection method), 448  
 riemann() (sage.manifolds.differentiable.levi\_civita\_connection.LeviCivitaConnection method), 485  
 riemann() (sage.manifolds.differentiable.metric.PseudoRiemannianMetric method), 470  
 riemann\_metric() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 199  
 riemannian\_metric() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 200

## S

sage.manifolds.chart (module), 58  
 sage.manifolds.continuous\_map (module), 150  
 sage.manifolds.coord\_func (module), 92  
 sage.manifolds.coord\_func\_symb (module), 99  
 sage.manifolds.differentiable.affine\_connection (module), 435  
 sage.manifolds.differentiable.automorphismfield (module), 351  
 sage.manifolds.differentiable.automorphismfield\_group (module), 345  
 sage.manifolds.differentiable.chart (module), 210

sage.manifolds.differentiable.curve (module), 269  
 sage.manifolds.differentiable.diff\_form (module), 422  
 sage.manifolds.differentiable.diff\_form\_module (module), 414  
 sage.manifolds.differentiable.diff\_map (module), 258  
 sage.manifolds.differentiable.levi\_civita\_connection (module), 480  
 sage.manifolds.differentiable.manifold (module), 175  
 sage.manifolds.differentiable.manifold\_homset (module), 254  
 sage.manifolds.differentiable.metric (module), 455  
 sage.manifolds.differentiable.real\_line (module), 223  
 sage.manifolds.differentiable.scalarfield (module), 239  
 sage.manifolds.differentiable.scalarfield\_algebra (module), 233  
 sage.manifolds.differentiable.tangent\_space (module), 282  
 sage.manifolds.differentiable.tangent\_vector (module), 285  
 sage.manifolds.differentiable.tensorfield (module), 365  
 sage.manifolds.differentiable.tensorfield\_module (module), 359  
 sage.manifolds.differentiable.tensorfield\_paral (module), 394  
 sage.manifolds.differentiable.vectorfield (module), 312  
 sage.manifolds.differentiable.vectorfield\_module (module), 294  
 sage.manifolds.differentiable.vectorframe (module), 330  
 sage.manifolds.manifold (module), 3  
 sage.manifolds.manifold\_homset (module), 148  
 sage.manifolds.point (module), 44  
 sage.manifolds.scalarfield (module), 122  
 sage.manifolds.scalarfield\_algebra (module), 116  
 sage.manifolds.structure (module), 42  
 sage.manifolds.subset (module), 31  
 sage.manifolds.utilities (module), 487  
 scalar\_field() (sage.manifolds.coord\_func.CoordFunction method), 95  
 scalar\_field() (sage.manifolds.manifold.TopologicalManifold method), 27  
 scalar\_field\_algebra (sage.manifolds.structure.DifferentialStructure attribute), 42  
 scalar\_field\_algebra (sage.manifolds.structure.RealDifferentialStructure attribute), 42  
 scalar\_field\_algebra (sage.manifolds.structure.RealTopologicalStructure attribute), 43  
 scalar\_field\_algebra (sage.manifolds.structure.TopologicalStructure attribute), 43  
 scalar\_field\_algebra() (sage.manifolds.manifold.TopologicalManifold method), 29  
 ScalarField (class in sage.manifolds.scalarfield), 122  
 ScalarFieldAlgebra (class in sage.manifolds.scalarfield\_algebra), 116  
 schouten() (sage.manifolds.differentiable.metric.PseudoRiemannianMetric method), 471  
 set() (sage.manifolds.differentiable.metric.PseudoRiemannianMetric method), 472  
 set() (sage.manifolds.differentiable.metric.PseudoRiemannianMetricParal method), 479  
 set\_axes\_labels() (in module sage.manifolds.utilities), 489  
 set\_change\_of\_frame() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 200  
 set\_coef() (sage.manifolds.differentiable.affine\_connection.AffineConnection method), 450  
 set\_comp() (sage.manifolds.differentiable.tensorfield.TensorField method), 387  
 set\_comp() (sage.manifolds.differentiable.tensorfield\_paral.TensorFieldParal method), 413  
 set\_coord() (sage.manifolds.point.ManifoldPoint method), 56  
 set\_coordinates() (sage.manifolds.point.ManifoldPoint method), 57  
 set\_default\_chart() (sage.manifolds.manifold.TopologicalManifold method), 29  
 set\_default\_frame() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 201  
 set\_expr() (sage.manifolds.continuous\_map.ContinuousMap method), 170  
 set\_expr() (sage.manifolds.scalarfield.ScalarField method), 144

`set_expression()` (sage.manifolds.continuous\_map.ContinuousMap method), 171  
`set_inverse()` (sage.manifolds.chart.CoordChange method), 70  
`set_name()` (sage.manifolds.differentiable.tensorfield.TensorField method), 388  
`set_name()` (sage.manifolds.scalarfield.ScalarField method), 145  
`set_restriction()` (sage.manifolds.differentiable.tensorfield.TensorField method), 388  
`signature()` (sage.manifolds.differentiable.metric.PseudoRiemannianMetric method), 472  
`simplify()` (sage.manifolds.coord\_func\_symb.CoordFunctionSymb method), 111  
`simplify_abs_trig()` (in module sage.manifolds.utilities), 490  
`simplify_chain_generic()` (in module sage.manifolds.utilities), 490  
`simplify_chain_real()` (in module sage.manifolds.utilities), 491  
`simplify_sqrt_real()` (in module sage.manifolds.utilities), 493  
`sin()` (sage.manifolds.coord\_func.CoordFunction method), 95  
`sin()` (sage.manifolds.coord\_func\_symb.CoordFunctionSymb method), 112  
`sin()` (sage.manifolds.scalarfield.ScalarField method), 145  
`sinh()` (sage.manifolds.coord\_func.CoordFunction method), 95  
`sinh()` (sage.manifolds.coord\_func\_symb.CoordFunctionSymb method), 113  
`sinh()` (sage.manifolds.scalarfield.ScalarField method), 146  
`sqrt()` (sage.manifolds.coord\_func.CoordFunction method), 95  
`sqrt()` (sage.manifolds.coord\_func\_symb.CoordFunctionSymb method), 113  
`sqrt()` (sage.manifolds.scalarfield.ScalarField method), 146  
`sqrt_abs_det()` (sage.manifolds.differentiable.metric.PseudoRiemannianMetric method), 473  
`start_index()` (sage.manifolds.manifold.TopologicalManifold method), 29  
`structure_coeff()` (sage.manifolds.differentiable.vectorframe.CoordFrame method), 336  
`structure_coeff()` (sage.manifolds.differentiable.vectorframe.VectorFrame method), 344  
`subcategory()` (sage.manifolds.structure.DifferentialStructure method), 42  
`subcategory()` (sage.manifolds.structure.RealDifferentialStructure method), 42  
`subcategory()` (sage.manifolds.structure.RealTopologicalStructure method), 43  
`subcategory()` (sage.manifolds.structure.TopologicalStructure method), 43  
`subset()` (sage.manifolds.subset.ManifoldSubset method), 39  
`subsets()` (sage.manifolds.subset.ManifoldSubset method), 39  
`sup()` (sage.manifolds.differentiable.real\_line.OpenInterval method), 230  
`superset()` (sage.manifolds.subset.ManifoldSubset method), 40  
`sym_bilin_form_field()` (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 201  
`sym_bilinear_form()` (sage.manifolds.differentiable.vectorfield\_module.VectorFieldFreeModule method), 301  
`symmetries()` (sage.manifolds.differentiable.tensorfield.TensorField method), 388  
`symmetrize()` (sage.manifolds.differentiable.tensorfield.TensorField method), 389

## T

`tan()` (sage.manifolds.coord\_func.CoordFunction method), 95  
`tan()` (sage.manifolds.coord\_func\_symb.CoordFunctionSymb method), 113  
`tan()` (sage.manifolds.scalarfield.ScalarField method), 147  
`tangent_identity_field()` (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 204  
`tangent_space()` (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 204  
`tangent_vector_field()` (sage.manifolds.differentiable.curve.DifferentiableCurve method), 280  
`TangentSpace` (class in sage.manifolds.differentiable.tangent\_space), 282  
`TangentVector` (class in sage.manifolds.differentiable.tangent\_vector), 285  
`tanh()` (sage.manifolds.coord\_func.CoordFunction method), 95  
`tanh()` (sage.manifolds.coord\_func\_symb.CoordFunctionSymb method), 114  
`tanh()` (sage.manifolds.scalarfield.ScalarField method), 147  
`tensor()` (sage.manifolds.differentiable.vectorfield\_module.VectorFieldFreeModule method), 302

tensor() (sage.manifolds.differentiable.vectorfield\_module.VectorFieldModule method), 311  
 tensor\_field() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 205  
 tensor\_field\_module() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 206  
 tensor\_from\_comp() (sage.manifolds.differentiable.vectorfield\_module.VectorFieldFreeModule method), 303  
 tensor\_module() (sage.manifolds.differentiable.vectorfield\_module.VectorFieldFreeModule method), 303  
 tensor\_module() (sage.manifolds.differentiable.vectorfield\_module.VectorFieldModule method), 311  
 tensor\_rank() (sage.manifolds.differentiable.tensorfield.TensorField method), 390  
 tensor\_type() (sage.manifolds.differentiable.scalarfield.DiffScalarField method), 253  
 tensor\_type() (sage.manifolds.differentiable.tensorfield.TensorField method), 390  
 tensor\_type() (sage.manifolds.differentiable.tensorfield\_module.TensorFieldModule method), 365  
 TensorField (class in sage.manifolds.differentiable.tensorfield), 365  
 TensorFieldFreeModule (class in sage.manifolds.differentiable.tensorfield\_module), 359  
 TensorFieldModule (class in sage.manifolds.differentiable.tensorfield\_module), 362  
 TensorFieldParal (class in sage.manifolds.differentiable.tensorfield\_paral), 398  
 top\_charts() (sage.manifolds.manifold.TopologicalManifold method), 30  
 TopologicalManifold (class in sage.manifolds.manifold), 11  
 TopologicalManifold.options() (in module sage.manifolds.manifold), 26  
 TopologicalManifoldHomset (class in sage.manifolds.manifold\_homset), 148  
 TopologicalStructure (class in sage.manifolds.structure), 43  
 torsion() (sage.manifolds.differentiable.affine\_connection.AffineConnection method), 451  
 torsion() (sage.manifolds.differentiable.levi\_civita\_connection.LeviCivitaConnection method), 486  
 torsion\_form() (sage.manifolds.differentiable.affine\_connection.AffineConnection method), 453  
 trace() (sage.manifolds.differentiable.tensorfield.TensorField method), 390  
 transition\_map() (sage.manifolds.chart.Chart method), 65  
 transition\_map() (sage.manifolds.differentiable.chart.DiffChart method), 215

## U

union() (sage.manifolds.subset.ManifoldSubset method), 41  
 up() (sage.manifolds.differentiable.tensorfield.TensorField method), 392  
 upper\_bound() (sage.manifolds.differentiable.real\_line.OpenInterval method), 230

## V

valid\_coordinates() (sage.manifolds.chart.Chart method), 66  
 valid\_coordinates() (sage.manifolds.chart.RealChart method), 91  
 vector\_field() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 207  
 vector\_field\_module() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 208  
 vector\_frame() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 209  
 VectorField (class in sage.manifolds.differentiable.vectorfield), 313  
 VectorFieldFreeModule (class in sage.manifolds.differentiable.vectorfield\_module), 294  
 VectorFieldModule (class in sage.manifolds.differentiable.vectorfield\_module), 304  
 VectorFieldParal (class in sage.manifolds.differentiable.vectorfield), 327  
 VectorFrame (class in sage.manifolds.differentiable.vectorframe), 336  
 volume\_form() (sage.manifolds.differentiable.metric.PseudoRiemannianMetric method), 474

## W

wedge() (sage.manifolds.differentiable.diff\_form.DiffForm method), 427  
 wedge() (sage.manifolds.differentiable.diff\_form.DiffFormParal method), 434  
 weyl() (sage.manifolds.differentiable.metric.PseudoRiemannianMetric method), 475

## X

xder() (in module sage.manifolds.utilities), [494](#)

## Z

zero() (sage.manifolds.coord\_func\_symb.CoordFunctionSymbRing method), [115](#)

zero() (sage.manifolds.differentiable.diff\_form\_module.DiffFormModule method), [421](#)

zero() (sage.manifolds.differentiable.tensorfield\_module.TensorFieldModule method), [365](#)

zero() (sage.manifolds.differentiable.vectorfield\_module.VectorFieldModule method), [312](#)

zero() (sage.manifolds.scalarfield\_algebra.ScalarFieldAlgebra method), [121](#)

zero\_function() (sage.manifolds.chart.Chart method), [67](#)

zero\_scalar\_field() (sage.manifolds.manifold.TopologicalManifold method), [30](#)