# CONTENTS

## 1 Topological Manifolds

1.1 Topological Manifolds .................................................. 3
1.2 Subsets of Topological Manifolds .................................. 37
1.3 Manifold Structures .................................................... 66
1.4 Points of Topological Manifolds .................................... 70
1.5 Coordinate Charts ....................................................... 85
  1.5.1 Coordinate Charts ............................................... 85
  1.5.2 Chart Functions .................................................. 130
  1.5.3 Coordinate calculus methods .................................. 162
1.6 Scalar Fields ........................................................... 168
  1.6.1 Algebra of Scalar Fields ....................................... 168
  1.6.2 Scalar Fields .................................................... 172
1.7 Continuous Maps ....................................................... 213
  1.7.1 SetsofMorphismsbetweenTopologicalManifolds ............. 213
  1.7.2 Continuous Maps Between Topological Manifolds .......... 216
  1.7.3 Images of Manifold Subsets under Continuous Maps as Subsets of the Codomain 243
1.8 Submanifolds of topological manifolds ............................ 243
1.9 Topological Vector Bundles .......................................... 252
  1.9.1 Topological Vector Bundle ..................................... 252
  1.9.2 Vector Bundle Fibers .......................................... 268
  1.9.3 Vector Bundle Fiber Elements ................................ 271
  1.9.4 Trivializations .................................................. 272
  1.9.5 Local Frames .................................................... 279
  1.9.6 Section Modules ................................................ 294
  1.9.7 Sections .......................................................... 302
1.10 Families of Manifold Objects ........................................ 326
1.11 Topological Closures of Manifold Subsets .......................... 328
1.12 Manifold Subsets Defined as Pullbacks of Subsets under Continuous Maps 329

## 2 Differentiable Manifolds

2.1 Differentiable Manifolds .............................................. 335
2.2 Coordinate Charts on Differentiable Manifolds .................... 388
2.3 The Real Line and Open Intervals ................................... 403
2.4 Scalar Fields .......................................................... 414
  2.4.1 Algebra of Differentiable Scalar Fields ..................... 414
  2.4.2 Differentiable Scalar Fields ................................... 419
2.5 Differentiable Maps and Curves ..................................... 440
  2.5.1 Sets of Morphisms between Differentiable Manifolds .......... 440
  2.5.2 Differentiable Maps between Differentiable Manifolds .... 453
  2.5.3 Curves in Manifolds ............................................ 465
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5.4</td>
<td>Integrated Curves and Geodesics in Manifolds</td>
<td>481</td>
</tr>
<tr>
<td>2.6</td>
<td>Tangent Spaces</td>
<td>514</td>
</tr>
<tr>
<td>2.6.1</td>
<td>Tangent Spaces</td>
<td>514</td>
</tr>
<tr>
<td>2.6.2</td>
<td>Tangent Vectors</td>
<td>518</td>
</tr>
<tr>
<td>2.7</td>
<td>Vector Fields</td>
<td>530</td>
</tr>
<tr>
<td>2.7.1</td>
<td>Vector Field Modules</td>
<td>530</td>
</tr>
<tr>
<td>2.7.2</td>
<td>Vector Fields</td>
<td>552</td>
</tr>
<tr>
<td>2.7.3</td>
<td>Vector Frames</td>
<td>579</td>
</tr>
<tr>
<td>2.7.4</td>
<td>Group of Tangent-Space Automorphism Fields</td>
<td>598</td>
</tr>
<tr>
<td>2.7.5</td>
<td>Tangent-Space Automorphism Fields</td>
<td>604</td>
</tr>
<tr>
<td>2.8</td>
<td>Tensor Fields</td>
<td>615</td>
</tr>
<tr>
<td>2.8.1</td>
<td>Tensor Field Modules</td>
<td>615</td>
</tr>
<tr>
<td>2.8.2</td>
<td>Tensor Fields</td>
<td>621</td>
</tr>
<tr>
<td>2.8.3</td>
<td>Tensor Fields with Values on a Parallelizable Manifold</td>
<td>667</td>
</tr>
<tr>
<td>2.9</td>
<td>Differential Forms</td>
<td>690</td>
</tr>
<tr>
<td>2.9.1</td>
<td>Differential Form Modules</td>
<td>690</td>
</tr>
<tr>
<td>2.9.2</td>
<td>Differential Forms</td>
<td>702</td>
</tr>
<tr>
<td>2.10</td>
<td>Mixed Differential Forms</td>
<td>721</td>
</tr>
<tr>
<td>2.10.1</td>
<td>Graded Algebra of Mixed Differential Forms</td>
<td>721</td>
</tr>
<tr>
<td>2.10.2</td>
<td>Mixed Differential Forms</td>
<td>726</td>
</tr>
<tr>
<td>2.11</td>
<td>De Rham Cohomology</td>
<td>740</td>
</tr>
<tr>
<td>2.12</td>
<td>Alternating Multivector Fields</td>
<td>744</td>
</tr>
<tr>
<td>2.12.1</td>
<td>Multivector Field Modules</td>
<td>744</td>
</tr>
<tr>
<td>2.12.2</td>
<td>Multivector Fields</td>
<td>750</td>
</tr>
<tr>
<td>2.13</td>
<td>Affine Connections</td>
<td>765</td>
</tr>
<tr>
<td>2.14</td>
<td>Submanifolds of differentiable manifolds</td>
<td>788</td>
</tr>
<tr>
<td>2.15</td>
<td>Differentiable Vector Bundles</td>
<td>792</td>
</tr>
<tr>
<td>2.15.1</td>
<td>Differentiable Vector Bundles</td>
<td>792</td>
</tr>
<tr>
<td>2.15.2</td>
<td>Bundle Connections</td>
<td>814</td>
</tr>
<tr>
<td>2.15.3</td>
<td>Characteristic cohomology classes</td>
<td>825</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Pseudo-Riemannian Manifolds</td>
<td>845</td>
</tr>
<tr>
<td>3.1</td>
<td>Pseudo-Riemannian Manifolds</td>
<td>845</td>
</tr>
<tr>
<td>3.2</td>
<td>Euclidean Spaces and Vector Calculus</td>
<td>855</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Euclidean Spaces</td>
<td>855</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Spheres smoothly embedded in Euclidean Space</td>
<td>881</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Operators for vector calculus</td>
<td>890</td>
</tr>
<tr>
<td>3.3</td>
<td>Pseudo-Riemannian Metrics and Degenerate Metrics</td>
<td>894</td>
</tr>
<tr>
<td>3.4</td>
<td>Levi-Civita Connections</td>
<td>928</td>
</tr>
<tr>
<td>3.5</td>
<td>Pseudo-Riemannian submanifolds</td>
<td>935</td>
</tr>
<tr>
<td>3.6</td>
<td>Degenerate Metric Manifolds</td>
<td>959</td>
</tr>
<tr>
<td>3.6.1</td>
<td>Degenerate manifolds</td>
<td>959</td>
</tr>
<tr>
<td>3.6.2</td>
<td>Degenerate submanifolds</td>
<td>965</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Poisson Manifolds</td>
<td>983</td>
</tr>
<tr>
<td>4.1</td>
<td>Poisson tensors</td>
<td>983</td>
</tr>
<tr>
<td>4.2</td>
<td>Symplectic structures</td>
<td>987</td>
</tr>
<tr>
<td>4.3</td>
<td>Symplectic vector spaces</td>
<td>994</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Utilities for Calculus</td>
<td>997</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Manifolds Catalog</td>
<td>1011</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Indices and Tables</td>
<td>1017</td>
</tr>
<tr>
<td>Python Module Index</td>
<td>1019</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>Index</td>
<td>1021</td>
<td></td>
</tr>
</tbody>
</table>
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.
1.1 Topological Manifolds

Given a topological field $K$ (in most applications, $K = \mathbb{R}$ or $K = \mathbb{C}$) and a non-negative integer $n$, a \textit{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 \texttt{TopologicalManifold}. Open subsets of topological manifolds are also implemented via \texttt{TopologicalManifold}, since they are topological manifolds by themselves.

In the current setting, topological manifolds are mostly described by means of charts (see \texttt{Chart}). \texttt{TopologicalManifold} serves as a base class for more specific manifold classes.

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

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

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

\begin{verbatim}
  sage: M = Manifold(2, 'S^2', structure='topological')  
  sage: M
  2-dimensional topological manifold S^2
\end{verbatim}

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

\begin{verbatim}
  sage: M.base_field()  
  Real Field with 53 bits of precision  
  sage: dim(M)  
  2
\end{verbatim}

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

\begin{verbatim}
  sage: U = M.open_subset('U'); U  
  Open subset U of the 2-dimensional topological manifold S^2
\end{verbatim}

A standard chart on $U$ is provided by the stereographic projection from the North pole to the equatorial plane:
Thanks to the operator $<x, y>$ 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)
<class '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 $S^2$
```

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)
```

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 $\text{sage.manifolds.chart.CordChange.inverse()}$:

```
sage: stereoN_to_S.inverse()
Change of coordinates from Chart (W, (u, v)) to Chart (W, (x, y))
sage: stereoN_to_S.inverse().display()
```
At this stage, we have four open subsets on $S^2$:

\[
x = u/(u^2 + v^2)  \\
y = v/(u^2 + v^2)
\]

\[\text{sage: M.subset_family()}
\]

\[
\text{Set \{S^2, U, V, W\} of open subsets of the 2-dimensional topological manifold S^2}
\]

\[\text{W is the open subset that is the complement of the two poles:}
\]

\[\text{sage: N in W or S in W}
\]

\[
\text{False}
\]

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

\[\text{sage: N in V, N in U}
\]

\[
\text{(True, False)}
\]

\[\text{sage: S in U, S in V}
\]

\[
\text{(True, False)}
\]

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

\[\text{sage: M.atlas()}
\]

\[
\text{[Chart (U, (x, y)), Chart (V, (u, v)),
Chart (W, (x, y)), Chart (W, (u, v))]}
\]

Let us consider the point of coordinates $(1, 2)$ in the chart $\text{stereoN}$:

\[\text{sage: p = M.point((1,2), chart=\text{stereoN}, name='p'); p}
\]

\[
\text{Point p on the 2-dimensional topological manifold S^2}
\]

\[\text{sage: p.parent()}
\]

\[
\text{2-dimensional topological manifold S^2}
\]

\[\text{sage: p in W}
\]

\[
\text{True}
\]

The coordinates of $p$ in the chart $\text{stereoS}$ are computed by letting the chart act on the point:

\[\text{sage: stereoS(p)}
\]

\[
\text{(1/5, 2/5)}
\]

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

\[\text{sage: stereoN(p)}
\]

\[
\text{(1, 2)}
\]

Similarly:

\[\text{sage: stereoS(N)}
\]

\[
\text{(0, 0)}
\]

\[\text{sage: stereoN(S)}
\]

\[
\text{(0, 0)}
\]

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

\[\text{sage: f = M.scalar_field({\text{stereoN: \text{atan}(x^2+y^2)}, \text{stereoS: pi/2-\text{atan}(u^2+v^2))},}
\]

\[
\text{....: name='f'}}
\]

(continues on next page)
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 \( \mathbb{C} \):

```plaintext
sage: M = Manifold(1, 'C*', structure='topological', field='complex'); M
Complex 1-dimensional topological manifold \( \mathbb{C}^* \)
```

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 \( \mathbb{C} \):

```plaintext
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 \( z \):

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

The origin of the complex plane is the point of coordinate \( z = 0 \):

```plaintext
sage: O = U.point((0,), chart=Z, name='O', latex_name=r'O'); O
Point \( O \) on the Complex 1-dimensional topological manifold \( \mathbb{C}^* \)
```

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

```plaintext
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:

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

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

```plaintext
sage: M.declare_union(U,V)
```
and we provide the transition map between the two charts as \( w = 1/z \) on \( A = U \cap V \):

```python
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 \((A, (w,))\)

```python
sage: Z_to_W.display()
w = 1/z
```

Change of coordinates from Chart \((A, (w,))\) to Chart \((A, (z,))\)

```python
sage: Z_to_W.inverse().display()
z = 1/w
```

Let consider the complex number \( i \) as a point of the Riemann sphere:

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

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

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

and we have:

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

The following subsets and charts have been defined:

```python
sage: M.subset_family()
Set \{A, U, V, \mathbb{C}^*\} of open subsets of the Complex 1-dimensional topological manifold \( \mathbb{C}^* \)
```

```python
sage: M.atlas()
[Chart \((U, (z,))\), Chart \((V, (w,))\), Chart \((A, (z,))\), Chart \((A, (w,))\)]
```

A constant map \( \mathbb{C}^* \to \mathbb{C} \):

```python
sage: f = M.constant_scalar_field(3+2*I, name='f'); f
Scalar field f on the Complex 1-dimensional topological manifold \( \mathbb{C}^* \)
```

```python
sage: f.display()
f: \mathbb{C}^* \to \mathbb{C}
on U: z \mapsto 2*I + 3
on V: w \mapsto 2*I + 3
```

```python
sage: f(0)
2*I + 3
sage: f(i)
2*I + 3
sage: f(inf)
2*I + 3
```

```python
sage: f.parent()
Algebra of scalar fields on the Complex 1-dimensional topological manifold \( \mathbb{C}^* \)
```
AUTHORS:
- Eric Gourgoulhon (2015): initial version
- Travis Scrimshaw (2015): structure described via `TopologicalStructure` or `RealTopologicalStructure`
- Michael Jung (2020): topological vector bundles and orientability

REFERENCES:
- [Lee2011]
- [Lee2013]
- [KN1963]
- [Huy2005]

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

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

Given a topological field $K$ (in most applications, $K = \mathbb{R}$ or $K = \mathbb{C}$) 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 $\mathbb{R}$. A differentiable (resp. smooth, resp. analytic) manifold is a manifold such that all transition maps are differentiable (resp. smooth, resp. analytic). A pseudo-Riemannian manifold is a real differentiable manifold equipped with a metric tensor $g$ (i.e. a field of non-degenerate symmetric bilinear forms), with the two subcases of Riemannian manifold ($g$ positive-definite) and Lorentzian manifold ($g$ has signature $n - 2$ or $2 - n$).

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 $\mathbb{R}$
  - 'complex' or an object of type `ComplexField` (e.g. CC) for a manifold over $\mathbb{C}$
  - 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
- 'pseudo-Riemannian' for a real differentiable manifold equipped with a pseudo-Riemannian metric; the signature is specified via the keyword argument signature (see below)
- 'Riemannian' for a real differentiable manifold equipped with a Riemannian (i.e. positive definite) metric
- 'Lorentzian' for a real differentiable manifold equipped with a Lorentzian metric; the signature convention is specified by the keyword argument signature = 'positive' (default) or 'negative'

- 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:
  - diff_degree – (only for differentiable manifolds; default: infinity): the degree of differentiability
  - ambient – (only to construct a submanifold): the ambient manifold
  - metric_name – (only for pseudo-Riemannian manifolds; default: 'g') string; name (symbol) given to the metric
  - metric_latex_name – (only for pseudo-Riemannian manifolds; default: None) string; LaTeX symbol to denote the metric; if none is provided, the symbol is set to metric_name
  - signature – (only for pseudo-Riemannian manifolds; 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. 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); for Lorentzian manifolds the values signature = 'positive' (default) or signature = 'negative' are allowed to indicate the chosen signature convention.

OUTPUT:
- a manifold of the specified type, as an instance of TopologicalManifold or one of its subclasses DifferentiableManifold or PseudoRiemannianManifold, or, if the keyword ambient is used, one of the subclasses TopologicalSubmanifold, DifferentiableSubmanifold, or PseudoRiemannianSubmanifold.

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
```

1.1. Topological Manifolds
A 3-dimensional real differentiable manifold of class $C^4$:

```python
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:

```python
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:

```python
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:

```python
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:

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

Other parameters can change the default of the parameter `structure`:

```python
sage: M = Manifold(3, 'M', diff_degree=0); M
3-dimensional topological manifold M
sage: M = Manifold(3, 'M', diff_degree=2); M
3-dimensional differentiable manifold M
sage: M = Manifold(3, 'M', metric_name='g'); M
3-dimensional Riemannian manifold M
```

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

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

Submanifolds are constructed by means of the keyword `ambient`:

```python
sage: N = Manifold(2, 'N', field='complex', ambient=M); N
2-dimensional differentiable submanifold N immersed in the
3-dimensional complex manifold M
```
The immersion $N \rightarrow M$ has to be specified in a second stage, via the method `set_immersion()` or `set_embedding()`.

For more detailed examples, see the documentation of `TopologicalManifold`, `DifferentiableManifold` and `PseudoRiemannianManifold`, or the documentation of `TopologicalSubmanifold`, `DifferentiableSubmanifold` and `PseudoRiemannianSubmanifold` for submanifolds.

### 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 $M_{\text{old}}$ 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 $M_{\text{old}}$:

```
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 \( M_{\text{old}} \) are still considered distinct:

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

This reflects the fact that the equality of manifold objects holds only for identical objects, i.e., one has \( M_1 == M_2 \) if, and only if, \( M_1 \) is \( M_2 \). Actually, the manifold classes inherit from `WithEqualityById`:

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

**class** `sage.manifolds.manifold.TopologicalManifold`

Topological manifold over a topological field \( K \).

Given a topological field \( K \) (in most applications, \( K = \mathbb{R} \) or \( K = \mathbb{C} \)) 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 \( \mathbb{R} \)
  - `'complex'` or an object of type `ComplexField` (e.g., `CC`) for a manifold over \( \mathbb{C} \)
  - an object in the category of topological fields (see `Fields` and `TopologicalSpaces`) for other types of manifolds
- `structure` – manifold structure (see `TopologicalStructure` or `RealTopologicalStructure`)
- `base_manifold` – (default: `None`) if not `None`, must be a topological manifold; the created object is then an open subset of `base_manifold`
- `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 $\mathbb{R}$):

```
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 $\mathbb{Q}$:

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

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

```
sage: N = Manifold(2, 'N', structure='topological', field=Qp(5)); N
#...
˓→ needs sage.rings.padics
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
```

(continues on next page)
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)
```

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)
```

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 $\mathbb{R}^2$ 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:

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

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

```python
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 \( U \):

```python
sage: c_spher.<r, ph> = U.chart(rr:(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:

```python
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 \( \mathbb{R} \)
  - 'complex' for the complex field \( \mathbb{C} \)
  - 'neither_real_nor_complex' for a field different from \( \mathbb{R} \) and \( \mathbb{C} \)

EXAMPLES:
**chart** 

```python
chart(coordinates='', names=None, calc_method=None, coord_restrictions=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 \((x^1, \ldots, x^n)\) of \(\varphi\), defined by \(\varphi(p) = (x^1(p), \ldots, x^n(p)) \in K^n\) 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, ranges and possible periodicities, 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)
- **calc_method** — (default: None) string defining the calculus method to be used on this chart; must be one of
  - 'SR': Sage's default symbolic engine (Symbolic Ring)
  - 'sympy': SymPy
- **coord_restrictions**: Additional restrictions on the coordinates. See below.

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

1. The coordinate symbol (a letter or a few letters).
2. (optional, only for manifolds over \(\mathbb{R}\)) The interval \(I\) defining the coordinate range: if not provided, the coordinate is assumed to span all \(\mathbb{R}\); 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 space character.
3. (optional) Indicator of the periodic character of the coordinate, either as period=T, where \(T\) is the period, or, for manifolds over \(\mathbb{R}\) only, as the keyword periodic (the value of the period is then deduced from the interval \(I\) declared in field 2; see the example below)
4. (optional) The LaTeX spelling of the coordinate; if not provided the coordinate symbol given in the first field will be used.

The order of fields 2 to 4 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, no period and no LaTeX spelling is to be
set for any coordinate, the argument coordinates can be omitted when the shortcut operator <>, is used to declare the chart (see examples below).

Additional restrictions on the coordinates can be set using the argument coord_restrictions. A restriction can be any symbolic equality or inequality involving the coordinates, such as x > y or x^2 + y^2 != 0. The items of the list (or set or frozenset) coord_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 (or set or frozenset) coord_restrictions. For example:

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

means (x > y) and ((x != 0) or (y != 0)) and (z^2 < x). If the list coord_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]. If the chart variables have not been declared as variables yet, coord_restrictions must be lambda-quoted.

OUTPUT:

- the created chart, as an instance of Chart or one of its subclasses, like RealDiffChart for differentiable manifolds over R.

EXAMPLES:

Chart on a 2-dimensional manifold:

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

Indeed, the declared coordinates are then known at the global level:
Actually the instruction `X.<x,y> = M.chart()` is equivalent to the combination of the two instructions `X = M.chart('x y')` and `(x,y) = X[:].`

As an example of coordinate ranges and LaTeX symbols passed via the string `coordinates` to `chart()`, let us introduce polar coordinates:

```
sage: U = M.open_subset(U, coord_def={X: x^2+y^2 != 0})
sage: P.<r,ph> = U.chart(rr:(0,+oo) ph:(0,2*pi):periodic:\phi); P
Chart (U, (r, \phi))
sage: latex(P)
\left(U,(r, \phi)\right)
```

Using `coord_restrictions`:

```
sage: D = Manifold(2, 'D', structure='topological')
sage: X.<x,y> = D.chart(coord_restrictions=lambda x,y: [x^2+y^2<1, x>0]); X
Chart (D, (x, y))
sage: X.valid_coordinates(0, 0)
False
sage: X.valid_coordinates(1/2, 0)
True
```

See the documentation of classes `Chart` and `RealChart` 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
```

(continues on next page)
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()

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

Define a continuous map from self to codomain.

\textbf{INPUT}:

- \texttt{codomain} – \texttt{TopologicalManifold}; the map's codomain
- \texttt{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

- \texttt{chart1} – (default: None; used only in case (ii) above) chart on self defining the start coordinates involved in \texttt{coord\_functions} for case (ii); if None, the coordinates are assumed to refer to the default chart of self
- \texttt{chart2} – (default: None; used only in case (ii) above) chart on codomain defining the target coordinates involved in \texttt{coord\_functions} for case (ii); if None, the coordinates are assumed to refer to the default chart of codomain
- \texttt{name} – (default: None) name given to the continuous map
- \texttt{latex\_name} – (default: None) LaTeX symbol to denote the continuous map; if None, the LaTeX symbol is set to name

\textbf{OUTPUT}:

- the continuous map as an instance of \texttt{ContinuousMap}
EXAMPLES:

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

```python
sage: M = Manifold(2, 'S^2', structure='topological')
sage: U = M.open_subset('U')
sage: _spher.<th,ph> = U.chart(r'\theta:(0,\pi):	heta \ phi:(0,2\pi):\phi')
sage: N = Manifold(3, 'R^3', latex_name=r'\mathbb{R}^3', structure='topological')
sage: _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):

```python
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:

```python
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:
```python
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:

```python
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, (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))}
```

(continues on next page)
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')
M.chart('x y')
Chart (M, (x, y))
M.chart('u v')
Chart (M, (u, v))
M.default_chart()
Chart (M, (x, y))
A = M.open_subset('A')
A.chart('t z')
Chart (A, (t, z))
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')
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')
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(rr:(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
```

**has_orientation()**

Check whether `self` admits an obvious or by user set orientation.

**See also:**

Consult `orientation()` for details about orientations.

**Note:** Notice that if `has_orientation()` returns `False` this does not necessarily mean that the manifold admits no orientation. It just means that the user has to set an orientation manually in that case, see
set_orientation().

EXAMPLES:
The trivial case:

```
sage: M = Manifold(3, 'M', structure='top')
sage: c.<x,y,z> = M.chart()
sage: M.has_orientation()
True
```

The non-trivial case:

```
sage: M = Manifold(2, 'M', structure='top')
sage: U = M.open_subset('U'); V = M.open_subset('V')
sage: M.declare_union(U, V)
sage: c_xy.<x,y> = U.chart(); c_uv.<u,v> = V.chart()
sage: M.has_orientation()
False
sage: M.set_orientation([c_xy, c_uv])
sage: M.has_orientation()
True
```

`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 \( \mathbb{R}^2 \) and \( \mathbb{R}^2 \):

```
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)', coord_restrictions=lambda x, y: x^2+y^2<1)
.....:  # Cartesian coord on M
sage: N = Manifold(2, 'N', structure='topological')  # \mathbb{R}^2
sage: c_XY.<X,Y> = N.chart()  # canonical coordinates on \mathbb{R}^2
sage: Phi = M.homeomorphism(N, [x /sqrt(1-x^2-y^2), y /sqrt(1-x^2-y^2)],
.....:  # Canonical homeomorphism M \to 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:

\[
\text{Id}_M : M \to M
\]

\[
p \mapsto p
\]

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:

```python
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:

```python
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, end=None*)

Single index generator.

**INPUT:**

- `start` – (default: None) initial value \(i_0\) of the index; if None, the value returned by `start_index()` is assumed
- `end` – (default: None) final value \(i_n\) of the index; if None, the value returned by `start_index()` plus \(n - 1\), where \(n\) is the manifold dimension, is assumed

**OUTPUT:**

- an iterable index, starting from \(i_0\) and ending at \(i_0 + i_n\)

**EXAMPLES:**

Index range on a 4-dimensional manifold:

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

(continues on next page)
[2, 3]
sage: list(M.irange(end=2))
[0, 1, 2]
sage: list(M.irange(start=1, end=2))
[1, 2]

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(start=2))
[2, 3, 4]
sage: list(M.irange(end=2))
[1, 2]
sage: list(M.irange(start=2, end=3))
[2, 3]

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={}, supersets=None)

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

• supersets – (default: only self) list of sets that the new open subset is a subset of

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

\[
\text{sage: } M = \text{Manifold(2, 'R^2', structure='topological')}
\]
\[
\text{sage: } \_\text{cart.<}_x,y> = M.\text{chart()} \quad \# \text{Cartesian coordinates on } \mathbb{R}^2
\]
\[
\text{sage: } U = M.\text{open_subset('U', coord_def={c\_cart: x^2+y^2<1})}; U
\]
Open subset \( U \) of the 2-dimensional topological manifold \( \mathbb{R}^2 \)

Since the argument \( \text{coord\_def} \) has been set, \( U \) is automatically provided with a chart, which is the restriction of the Cartesian one to \( U \):

\[
\text{sage: } U.\text{atlas()}
\]

[Chart \((U, (x, y))\)]

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

\[
\text{sage: } M.\text{point}((0,0)) \text{ in } U
\]
True
\[
\text{sage: } M.\text{point}((1/2,1/3)) \text{ in } U
\]
True
\[
\text{sage: } M.\text{point}((1,2)) \text{ in } U
\]
False

\[
\text{options = Current options for manifolds – omit\_function\_arguments: False – textbook\_output: True}
\]

\textbf{orientation()}

Get the preferred orientation of \textit{self} if available.

An \textit{orientation} of an \( n \)-dimensional topological manifold is an atlas of charts whose transition maps are orientation preserving. A homeomorphism \( f: U \rightarrow V \) for open subsets \( U, V \subset \mathbb{R}^n \) is called \textit{orientation preserving} if for each \( x \in U \) the following map between singular homologies is the identity:

\[
H_n(\mathbb{R}^n, \mathbb{R}^n - 0; \mathbb{Z}) \cong H_n(U, U - x; \mathbb{Z}) \xrightarrow{f_*} H_n(V, V - f(x)) \cong H_n(\mathbb{R}^n, \mathbb{R}^n - 0; \mathbb{Z})
\]

See this link for details.

\textbf{Note:} Notice that for differentiable manifolds, the notion of orientability does not need homology theory at all. See \textit{orientation()} for details

The trivial case corresponds to the manifold being covered by one chart. In that case, if no preferred orientation has been manually set before, one of those charts (usually the default chart) is set to the preferred orientation and returned here.

\textbf{EXAMPLES:}

If the manifold is covered by only one chart, it certainly admits an orientation:
sage: M = Manifold(3, 'M', structure='top')
sage: c.<x,y,z> = M.chart()
sage: M.orientation()
[Chart (M, (x, y, z))]

Usually, an orientation cannot be obtained so easily:

sage: M = Manifold(2, 'M', structure='top')
sage: U = M.open_subset('U'); V = M.open_subset('V')
sage: M.declare_union(U, V)
sage: c_xy.<x,y> = U.chart(); c_uv.<u,v> = V.chart()
sage: M.orientation()
[]

In that case, the orientation can be set by the user manually:

sage: M.set_orientation([c_xy, c_uv])
sage: M.orientation()
[Chart (U, (x, y)), Chart (V, (u, v))]

The orientation on submanifolds are inherited from the ambient manifold:

sage: W = U.intersection(V, name='W')
sage: W.orientation()
[Chart (W, (x, y))]

```
scalar_field(coord_expression=None, chart=None, name=None, latex_name=None)
```

Define a scalar field on the manifold.

See :class:`ScalarField` (or :class:`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 :meth:`add_expr()`, :meth:`add_expr_by_continuation()`, or :meth:`set_expr()`.

**OUTPUT:**

- instance of :class:`ScalarField` (or of the subclass :class:`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()`, `ScalarField_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()
Join of Category of commutative algebras over Symbolic Ring and Category of homsets of topological spaces
```
sage: CU.zero()
Scalar field zero on the Open subset U of the 3-dimensional topological ...

The output is cached:

sage: U.scalar_field_algebra() is CU
True

**set_calculus_method**(method)

Set the calculus method to be used for coordinate computations on this manifold.
The provided method is transmitted to all coordinate charts defined on the manifold.

**INPUT:**

- method – string specifying the method to be used for coordinate computations on this manifold; one of
  - 'SR': Sage's default symbolic engine (Symbolic Ring)
  - 'sympy': SymPy

**EXAMPLES:**

Let us consider a scalar field $f$ on a 2-dimensional manifold:

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field(x^2 + cos(y)*sin(x), name='F')

By default, the coordinate expression of $f$ returned by `expr()` is a Sage's symbolic expression:

sage: f.expr()
x^2 + cos(y)*sin(x)
sage: type(f.expr())
<class 'sage.symbolic.expression.Expression'>
sage: parent(f.expr())
Symbolic Ring
sage: f.display()
F: $M \to \mathbb{R}$
$(x, y) \mapsto x^2 + \cos(y) \cdot \sin(x)$

If we change the calculus method to SymPy, it becomes a SymPy object instead:

sage: M.set_calculus_method('sympy')
sage: f.expr()
x**2 + sin(x)*cos(y)
sage: type(f.expr())
<class 'sympy.core.add.Add'>
sage: parent(f.expr())
<class 'sympy.core.add.Add'>
sage: f.display()
F: $M \to \mathbb{R}$
$(x, y) \mapsto x^2 + \sin(x) \cdot \cos(y)$

Back to the Symbolic Ring:
sage: M.set_calculus_method('SR')
sage: f.display()
F: M → ℝ
  (x, y) ↦ x^2 + \cos(y)\cdot \sin(x)

The calculus method chosen via set_calculus_method() applies to any chart defined subsequently on the manifold:

sage: M.set_calculus_method('sympy')
sage: Y.<u,v> = M.chart()  # a new chart
sage: Y.calculus_method()
Available calculus methods (* = current):
- SR (default)
- sympy (*)

See also:

- calculus_method() for a control of the calculus method chart by chart
- 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))

set_orientation(orientation)

Set the preferred orientation of self.

INPUT:

- orientation – a chart or a list of charts

**Warning:** It is the user's responsibility that the orientation set here is indeed an orientation. There is no check going on in the background. See orientation() for the definition of an orientation.

EXAMPLES:

Set an orientation on a manifold:

sage: M = Manifold(2, 'M', structure='top')
sage: c_xy.<x,y> = M.chart(); c_uv.<u,v> = M.chart()
sage: M.set_orientation(c_uv)
sage: M.orientation()
[Chart (M, (u, v))]
Set an orientation in the non-trivial case:

```python
sage: M = Manifold(2, 'M', structure='top')
sage: U = M.open_subset('U'); V = M.open_subset('V')
sage: M.declare_union(U, V)
sage: c_xy.<x,y> = U.chart(); c_uv.<u,v> = V.chart()
sage: M.set_orientation([c_xy, c_uv])
sage: M.orientation()
[Chart (U, (x, y)), Chart (V, (u, v))]
```

`set_simplify_function(simplifying_func, method=None)`

Set the simplifying function associated to a given coordinate calculus method in all the charts defined on `self`.

**INPUT:**
- `simplifying_func` – either the string 'default' for restoring the default simplifying function or a function `f` of a single argument `expr` such that `f(expr)` returns an object of the same type as `expr` (hopefully the simplified version of `expr`), this type being
  - `Expression` if `method = 'SR'`
  - a SymPy type if `method = 'sympy'`
- `method` – (default: None) string defining the calculus method for which `simplifying_func` is provided; must be one of
  - 'SR': Sage's default symbolic engine (Symbolic Ring)
  - 'sympy': SymPy
  - None: the currently active calculus method on each chart is assumed

See also:
- `calculus_method()` and `sage.manifolds.calculus_method.CalculusMethod.simplify()` for a control of the calculus method chart by chart

**EXAMPLES:**

Let us add two scalar fields on a 2-dimensional manifold:

```python
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field((x+y)^2 + cos(x)^2)
sage: g = M.scalar_field(-x^2-2*x*y-y^2 + sin(x)^2)
sage: f.expr()
(x + y)^2 + cos(x)^2
sage: g.expr()
-x^2 - 2*x*y - y^2 + sin(x)^2
sage: s = f + g
```

The outcome is automatically simplified:

```python
sage: s.expr()
1
```

The simplification is performed thanks to the default simplifying function on chart `X`, which is `simplify_chain_real()` in the present case (real manifold and SR calculus):
sage: X.calculus_method().simplify_function() is \
.....: sage.manifolds.utilities.simplify_chain_real
True

Let us change it to the generic Sage function \texttt{simplify}():

\begin{verbatim}
sage: M.set_simplify_function(simplify)
sage: X.calculus_method().simplify_function() is simplify
True
\end{verbatim}

\texttt{simplify}() is faster, but it does not do much:

\begin{verbatim}
sage: s = f + g
sage: s.simplify()
(x + y)^2 - x^2 - 2*x*y - y^2 + \cos(x)^2 + \sin(x)^2
\end{verbatim}

We can replaced it by any user defined function, for instance:

\begin{verbatim}
sage: def simpl_trig(a):
.....: return a.simplify_trig()
sage: M.set_simplify_function(simpl_trig)
sage: s = f + g
sage: s.simplify()
1
\end{verbatim}

The default simplifying function is restored via:

\begin{verbatim}
sage: M.set_simplify_function('default')
\end{verbatim}

Then we are back to:

\begin{verbatim}
sage: X.calculus_method().simplify_function() is \
.....: sage.manifolds.utilities.simplify_chain_real
True
\end{verbatim}

Thanks to the argument \texttt{method}, one can specify a simplifying function for a calculus method distinct from the current one. For instance, let us define a simplifying function for SymPy (note that \texttt{trigsimp}() is a SymPy method only):

\begin{verbatim}
sage: def simpl_trig_sympy(a):
.....: return a.trigsimp()
sage: M.set_simplify_function(simpl_trig_sympy, method='sympy')
\end{verbatim}

Then, it becomes active as soon as we change the calculus engine to SymPy:

\begin{verbatim}
sage: M.set_calculus_method('sympy')
sage: X.calculus_method().simplify_function() is simpl_trig_sympy
True
\end{verbatim}

We have then:

\begin{verbatim}
sage: s = f + g
sage: s.simplify()
1
sage: type(s.simplify())
<class 'sympy.core.numbers.One'>
\end{verbatim}
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.

vector_bundle(rank, name, field='real', latex_name=None)

Return a topological vector bundle over the given field with given rank over this topological manifold.

INPUT:

• rank – rank of the vector bundle
• name – name given to the total space
• field – (default: 'real') topological field giving the vector space structure to the fibers
• latex_name – optional LaTeX name for the total space

OUTPUT:

• a topological vector bundle as an instance of TopologicalVectorBundle
EXAMPLES:

```python
sage: M = Manifold(2, 'M', structure='top')
sage: M.vector_bundle(2, 'E')
Topological real vector bundle E -> M of rank 2 over the base space 2-dimensional topological manifold M
```

`zero_scalar_field()`

Return the zero scalar field defined on `self`.

OUTPUT:

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

EXAMPLES:

```python
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`. Besides, subsets that are images of a manifold subset under a continuous map are implemented by the subclass `ImageManifoldSubset`.

AUTHORS:

- Travis Scrimshaw (2015): review tweaks; removal of facade parents
- Matthias Koeppe (2021): Families and posets of subsets

REFERENCES:

- [Lee2011]

EXAMPLES:

Two subsets on a manifold:

```python
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.subset_family()
Set {A, B, M} of subsets of the 2-dimensional topological manifold M
```
The intersection of the two subsets:

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

Their union:

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

Families of subsets after the above operations:

```python
sage: M.subset_family()
Set {A, A_inter_B, A_union_B, B, M} of subsets of the 2-dimensional topological manifold M
sage: a.subset_family()
Set {A, A_inter_B} of subsets of the 2-dimensional topological manifold M
sage: c.subset_family()
Set {A_inter_B} of subsets of the 2-dimensional topological manifold M
sage: d.subset_family()
Set {A, A_inter_B, A_union_B, B} of subsets of the 2-dimensional topological manifold M
```

```python
class sage.manifolds.subset.ManifoldSubset (manifold, name: str, latex_name=None, category=None)

    Bases: UniqueRepresentation, 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:

    ```python
    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:

    ```python
    sage: M.subset()
```

Chapter 1. Topological Manifolds
Manifolds, Release 10.3

sage: B = M.subset('B', latex_name=r'\mathcal{B}'); B
Subset B of the 2-dimensional topological manifold M
sage: M.subset_family()
Set {A, B, M} of subsets of the 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

closure(name=None, latex_name=None)

Return the topological closure of `self` as a subset of the manifold.

**INPUT:**

- `name` – (default: None) name given to the difference in the case the latter has to be created; the default prepends cl_ to `self._name`

1.2. Subsets of Topological Manifolds 39
• \texttt{latex\_name} – (default: None) LaTeX symbol to denote the difference in the case the latter has to be created; the default is built upon the operator \texttt{cl}

**OUTPUT:**

• if \texttt{self} is already known to be closed (see \texttt{is\_closed()}), \texttt{self}; otherwise, an instance of \texttt{ManifoldSubsetClosure}

**EXAMPLES:**

```python
sage: M = Manifold(2, 'M', structure='topological')
sage: A = M.subset('A')
sage: B1 = A.subset('B1')
sage: B2 = A.subset('B2')
sage: B1.complement()
Subset M_minus_B1 of the 2-dimensional topological manifold M
```

\textbf{complement} (\texttt{superset=None, name=None, latex\_name=None, is\_open=False})

Return the complement of \texttt{self} in the manifold or in \texttt{superset}.

**INPUT:**

• \texttt{superset} – (default: \texttt{self.manifold()}) a superset of \texttt{self}

• \texttt{name} – (default: None) name given to the complement in the case the latter has to be created; the default is \texttt{superset._name} minus \texttt{self._name}

• \texttt{latex\_name} – (default: None) LaTeX symbol to denote the complement in the case the latter has to be created; the default is built upon the symbol $\setminus$

• \texttt{is\_open} – (default: False) if \texttt{True}, the created subset is assumed to be open with respect to the manifold's topology

**OUTPUT:**

• instance of \texttt{ManifoldSubset} representing the subset that is \texttt{superset} minus \texttt{self}

**EXAMPLES:**

```python
sage: M = Manifold(2, 'M', structure='topological')
sage: A = M.subset('A')
sage: B1 = A.subset('B1')
sage: B2 = A.subset('B2')
sage: B1.complement()
Subset M_minus_B1 of the 2-dimensional topological manifold M
```

(continues on next page)
Demanding that the complement is open makes `self` a closed subset:

```python
default
sage: A.is_closed()  # False a priori
False
sage: A.complement(is_open=True)
Open subset M_minus_A of the 2-dimensional topological manifold M
sage: A.is_closed()
True
```

**declare_closed()**

Declare `self` to be a closed subset of the manifold.

**EXAMPLES:**

```python
sage: M = Manifold(2, 'M', structure='topological')
sage: A = M.subset('A')
sage: B1 = A.subset('B1')
sage: B1.is_closed()
False
sage: B1.declare_closed()
True
sage: B2 = A.subset('B2')
sage: c1_B2 = B2.closure()
sage: A.declare_closed()
sage: c1_B2.is_subset(A)
True
```

**declare_empty()**

Declare that `self` is the empty set.

**EXAMPLES:**

```python
sage: M = Manifold(2, 'M', structure='topological')
sage: A = M.subset('A', is_open=True)
sage: AA = A.subset('AA')
sage: A
Open subset A of the 2-dimensional topological manifold M
sage: A.declare_empty()
sage: A.is_empty()
True
```

Empty sets do not allow to define points on them:

```python
sage: A.point()
Traceback (most recent call last):
  ...
TypeError: cannot define a point on the
  Open subset A of the 2-dimensional topological manifold M
  because it has been declared empty
```

Emptiness transfers to subsets:
Manifolds, Release 10.3

```python
sage: AA.is_empty()
True

sage: AA.point()
Traceback (most recent call last):
...
TypeError: cannot define a point on the
Subset AA of the 2-dimensional topological manifold M
because it has been declared empty

sage: AD = A.subset('AD')
sage: AD.is_empty()
True

If points have already been defined on self (or its subsets), it is an error to declare it to be empty:

```python
sage: B = M.subset('B')
sage: b = B.point(name='b'); b
Point b on the 2-dimensional topological manifold M
sage: B.declare_empty()
Traceback (most recent call last):
...
TypeError: cannot be empty because it has defined points
```

Emptiness is recorded as empty open covers:

```python
sage: P = M.subset_poset(open_covers=True, points=[b])       
# needs sage.graphs
sage: def label(element):
.....:     if isinstance(element, str):
.....:         return element
.....:     try:
.....:         return element._name
.....:     except AttributeError:
.....:         return '!' + ', '.join(sorted(x._name for x in element)) + ']

sage: P.plot(element_labels={element: label(element) for element in P})  
# needs sage.graphs sage.plot
```

```python
define_equal(*others)
Declare that self and others are the same sets.

INPUT:

- others – finitely many subsets or iterables of subsets of the same manifold as self.

EXAMPLES:

```python
sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U')
sage: V = M.open_subset('V')
sage: Vs = [M.open_subset(f'V{i}') for i in range(2)]
sage: UV = U.intersection(V)
sage: W = UV.open_subset('W')
sage: P = M.subset_poset()  
# needs sage.graphs
sage: def label(element):
.....:     return element._name
sage: P.plot(element_labels={element: label(element) for element in P})  
# needs sage.graphs sage.plot
```

(continues on next page)
Graphics object consisting of 15 graphics primitives

```python
sage: V.declare_equal(Vs)
```

```python
sage: P = M.subset_poset()
#needs sage.graphs
```

```python
sage: P.plot(element_labels={element: label(element) for element in P})
#needs sage.graphs sage.plot
```

Graphics object consisting of 11 graphics primitives

```python
sage: W.declare_equal(U)
```

```python
sage: P = M.subset_poset()
#needs sage.graphs
```

```python
sage: P.plot(element_labels={element: label(element) for element in P})
#needs sage.graphs sage.plot
```

Graphics object consisting of 6 graphics primitives

### `declare_nonempty()`

Declare that `self` is nonempty.

Once declared nonempty, `self` (or any of its supersets) cannot be declared empty.

This is equivalent to defining a point on `self` using `point()` but is cheaper than actually creating a `ManifoldPoint` instance.

#### EXAMPLES:

```python
sage: M = Manifold(2, 'M', structure='topological')
sage: A = M.subset('A', is_open=True)
sage: AA = A.subset('AA')
sage: AA.declare_nonempty()
sage: A.has_defined_points()
```

(continues on next page)
True

```
sage: A.declare_empty()
Traceback (most recent call last):
...
TypeError: cannot be empty because it has defined points
```

```python
sage: declare_subset(*supersets)

Declare self to be a subset of each of the given supersets.

INPUT:

- supersets – other subsets of the same manifold

EXAMPLES:
```
sage: M = Manifold(2, 'M')
sage: U1 = M.open_subset('U1')
sage: U2 = M.open_subset('U2')
sage: V = M.open_subset('V')
sage: V.superset_family()
Set {M, V} of open subsets of the 2-dimensional differentiable manifold M
sage: U1.superset_family()
Set {U1} of open subsets of the 2-dimensional differentiable manifold M
sage: P = M.subset_poset()  # needs sage.graphs
sage: P.plot(element_labels={element: label(element) for element in P})  # needs sage.graphs sage.plot
Graphics object consisting of 8 graphics primitives
sage: V.declare_subset(U1, U2)
sage: V.superset_family()
Set {M, U1, U2, V} of open subsets of the 2-dimensional differentiable manifold M
sage: P = M.subset_poset()  # needs sage.graphs
sage: P.plot(element_labels={element: label(element) for element in P})  # needs sage.graphs sage.plot
Graphics object consisting of 9 graphics primitives
```
Subsets in a directed cycle of inclusions are equal:

```python
sage: M.declare_subset(V)
sage: M.superset_family()
Set {M, U1, U2, V} of open subsets of the 2-dimensional differentiable manifold M
sage: M.equal_subset_family()
Set {M, U1, U2, V} of open subsets of the 2-dimensional differentiable manifold M
sage: P = M.subset_poset()  # needs sage.graphs
sage: P.plot(element_labels={element: label(element) for element in P})  # needs sage.graphs sage.plot
Graphics object consisting of 2 graphics primitives
```

```
@declare_superset(*subsets)
Declares self to be a superset of each of the given subsets.

INPUT:
- subsets – other subsets of the same manifold

EXAMPLES:
```
```
Subsets in a directed cycle of inclusions are equal:

\[
\text{sage: } W.\text{declare_superset}(U) \\
\text{sage: } W.\text{subset_family}() \\
\text{Set } \{U, V_1, V_1_{\text{inter}} V_2, V_2\} \text{ of open subsets of the 2-dimensional differentiable manifold } M \\
\text{sage: } W.\text{equal_subset_family}() \\
\text{Set } \{U, V_1, V_1_{\text{inter}} V_2, V_2\} \text{ of open subsets of the 2-dimensional differentiable manifold } M \\
\text{sage: } P = M.\text{subset_poset()} \\
\text{# needs sage.graphs} \\
\text{sage: } P.\text{plot(element_labels=\{element: label(element) for element in P\})} \\
\text{# needs sage.graphs sage.plot} \\
\text{Graphics object consisting of 4 graphics primitives}
\]

\[\text{declare_union}(\text{disjoint, } *\text{subsets_or_families})\]

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

Suppose \(U\) is the current subset, then this method declares that \(U = \bigcup_{S \in F} S\).

**INPUT:**

- `subsets_or_families` – finitely many subsets or iterables of subsets
- `disjoint` – (default: False) whether to declare the subsets pairwise disjoint

**EXAMPLES:**

\[
\text{sage: } M = \text{Manifold}(2, 'M', \text{structure='topological'}) \\
\text{sage: } AB = M.\text{subset('AB')} \\
\text{sage: } A = AB.\text{subset('A')} \\
\text{sage: } B = AB.\text{subset('B')} \\
\text{sage: } \text{def } \text{label}(\text{element}): \\
\text{.....: } \text{try:}
\]

(continues on next page)
Manifolds, Release 10.3

(continued from previous page)

```python
....:     return element._name
....: except AttributeError:
....:     return '[ ' + ', '.join(x._name for x in element)) + ' ]'
sage: P = M.subset_poset(open_covers=True); P
# needs sage.graphs
Finite poset containing 4 elements
sage: P.plot(element_labels={element: label(element) for element in P})  # needs sage.graphs sage.plot
Graphics object consisting of 8 graphics primitives

sage: AB.declare_union(A, B)
sage: A.union(B)
Subset AB of the 2-dimensional topological manifold M
sage: P = M.subset_poset(open_covers=True); P  # needs sage.graphs
Finite poset containing 4 elements
sage: P.plot(element_labels={element: label(element) for element in P})  # needs sage.graphs sage.plot
Graphics object consisting of 8 graphics primitives

sage: B1 = B.subset('B1', is_open=True)
sage: B2 = B.subset('B2', is_open=True)
sage: B.declare_union(B1, B2, disjoint=True)
sage: P = M.subset_poset(open_covers=True); P  # needs sage.graphs
Finite poset containing 9 elements
sage: P.plot(element_labels={element: label(element) for element in P})  # needs sage.graphs sage.plot
Graphics object consisting of 19 graphics primitives
```

difference (other, name=None, latex_name=None, is_open=False)

Return the set difference of self minus other.

INPUT:

- other – another subset of the same manifold
- name – (default: None) name given to the difference in the case the latter has to be created; the default is self._name minus other._name
- latex_name – (default: None) LaTeX symbol to denote the difference in the case the latter has to be

1.2. Subsets of Topological Manifolds
created; the default is built upon the symbol \( \setminus \)

- `is_open` (default: `False`) if `True`, the created subset is assumed to be open with respect to the manifold’s topology

**OUTPUT:**

- instance of `ManifoldSubset` representing the subset that is `self` minus `other`

**EXAMPLES:**

```python
sage: M = Manifold(2, 'M', structure='topological')
sage: A = M.subset('A')
sage: CA = M.difference(A)  # CA is an open subset of M
sage: latex(CA)  # LaTeX representation of CA
\setminus A
sage: A.intersection(CA).is_empty()  # True
sage: A.union(CA)  # A union CA is the whole manifold M
sage: O = M.open_subset('O')
sage: CO = M.difference(O)  # CO is an open subset of M
sage: M.difference(O) == CO  # True
sage: CO2 = M.difference(O, is_open=True, name='CO2')  # CO2 is an open subset of M
sage: CO.is_subset(CO2) and CO2.is_subset(CO)  # True
```

Since \( O \) is open and we have asked \( M \setminus O \) to be open, \( O \) is a clopen set (if \( O \neq M \) and \( O \neq \emptyset \), this implies that \( M \) is not connected):

```python
sage: O.is_closed() and O.is_open()  # True
```

**equal_subset_family()**

Generate the declared manifold subsets that are equal to `self`.

**Note:** If you only need to iterate over the equal sets in arbitrary order, you can use the generator method `equal_subsets()` instead.

**EXAMPLES:**

```python
sage: M = Manifold(2, 'M', structure='topological')
sage: U = M.open_subset('U')
sage: V = U.subset('V')
sage: V.declare_equal(M)
sage: V.equal_subset_family()
Set {M, U, V} of subsets of the 2-dimensional topological manifold M
```
equal_subsets()
Generate the declared manifold subsets that are equal to self.

Note: To get the equal subsets as a family, sorted by name, use the method equal_subset_family() instead.

EXAMINES:

sage: M = Manifold(2, 'M', structure='topological')
sage: U = M.open_subset('U')
sage: V = U.subset('V')
sage: V.declare_equal(M)
sage: sorted(V.equal_subsets(), key=lambda v: v._name)
[2-dimensional topological manifold M, Open subset U of the 2-dimensional topological manifold M, Subset V of the 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 sub-
set) representing the subset whose name is name

EXAMINES:

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.subset_family()
Set (A, B, M, U) of subsets 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

has_defined_points(subsets=True)
Return whether any points have been defined on self or any of its subsets.

INPUT:
• subsets – (default: True) if False, only consider points that have been defined directly on self;
  if True, also consider points on all subsets.

1.2. Subsets of Topological Manifolds
EXAMPLES:

```python
sage: M = Manifold(2, 'M', structure='topological')
sage: A = M.subset('A', is_open=True)
sage: AA = A.subset('AA')
sage: AA.point()
Point on the 2-dimensional topological manifold M
sage: AA.has_defined_points()
True
sage: A.has_defined_points(subsets=False)
False
sage: A.has_defined_points()
True
```

**intersection**(name, latex_name, *others)

Return the intersection of the current subset with other subsets.

This method may return a previously constructed intersection instead of creating a new subset. In this case, name and latex_name are not used.

**INPUT:**

- `others` – other subsets 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 inters 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 others

**EXAMPLES:**

Intersection of two subsets:

```python
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.subset_family()
Set {A, A_inter_B} of subsets of the 2-dimensional topological manifold M
sage: b.subset_family()
Set {A_inter_B, B} of subsets of the 2-dimensional topological manifold M
sage: c.superset_family()
Set {A, A_inter_B, B, M} of subsets of the 2-dimensional topological manifold M
```

Intersection of six subsets:

```python
sage: T = Manifold(2, 'T', structure='topological')
sage: S = [T.subset(f'S{i}' for i in range(6)) for i in range(3)]
sage: [S[i].intersection(S[i+3]) for i in range(3)]
[Subset S0_inter_S3 of the 2-dimensional topological manifold T,
 Subset S1_inter_S4 of the 2-dimensional topological manifold T,
 Subset S2_inter_S5 of the 2-dimensional topological manifold T]
```

(continues on next page)
sage: inter_S_i = T.intersection(*S, name='inter_S_i'); inter_S_i
Subset inter_S_i of the 2-dimensional topological manifold T
sage: inter_S_i.superset_family()
Set {S0, S0_inter_S3, S0_inter_S3_inter_S1_inter_S4, S1, S1_inter_S4,
     S2, S2_inter_S5, S3, S4, S5, T, inter_S_i} of
     subsets of the 2-dimensional topological manifold T

is_closed()

Return if self is a closed set.

EXAMPLES:

sage: M = Manifold(2, 'M', structure='topological')
sage: M.is_closed()
True
sage: also_M = M.subset('also_M')
sage: M.declare_subset(also_M)
sage: also_M.is_closed()
True
sage: A = M.subset('A')
`````
However, using `declare_empty()`, a subset can be declared empty, and emptiness transfers to all of its subsets.

**EXAMPLES:**

```sage
M = Manifold(2, 'M', structure='topological')
A = M.subset('A', is_open=True)
AA = A.subset('AA')
sage: A.is_empty()
False
sage: A.declare_empty()
sage: A.is_empty()
True
sage: AA.is_empty()
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')
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')
a = M.subset('A')
b = a.subset('B')
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:

```python
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:** This method is deprecated.

To get the subsets as a `ManifoldSubsetFiniteFamily` instance (which sorts its elements alphabetically by name), use `subset_family()` instead.

To loop over the subsets in an arbitrary order, use the generator method `subsets()` instead.

**EXAMPLES:**

List of subsets of a 2-dimensional manifold (deprecated):

```python
sage: M = Manifold(2, 'M', structure='topological')
sage: U = M.open_subset('U')
sage: V = M.subset('V')
sage: M.list_of_subsets()
```

Using `subset_family()` instead (recommended when order matters):

```python
sage: M.subset_family()
```

The method `subsets()` generates the subsets in an unspecified order. To create a set:

```python
sage: frozenset(M.subsets())  # random (set output)
```
**manifold()**

Return the ambient manifold of self.

**EXAMPLES:**

```python
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`:

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

**open_cover_family** *(trivial=True, supersets=False)*

Return the family 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 a `ManifoldSubsetFiniteFamily` $F$ of open subsets $U \in F$ of $M$ such that

$$A \subseteq \bigcup_{U \in F} U.$$  

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

$$A = \bigcup_{U \in F} U.$$  

The family is sorted lexicographically by the names of the subsets forming the open covers.

**Note:** If you only need to iterate over the open covers in arbitrary order, you can use the generator method `open_covers()` instead.

**INPUT:**

- `trivial` – (default: True) if `self` is open, include the trivial open cover of `self` by itself
- `supersets` – (default: False) if True, include open covers of all the supersets; it can also be an iterable of supersets to include

**EXAMPLES:**

```python
sage: M = Manifold(2, 'M', structure='topological')
sage: M.open_cover_family()
Set {{M}} of objects of the 2-dimensional topological manifold M
sage: U = M.open_subset('U')
sage: U.open_cover_family()
Set {{U}} of objects 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_cover_family()
```

(continues on next page)
Set \{\{A, B\}, \{U\}\} of objects of the 2-dimensional topological manifold M

sage: U.open_cover_family(trivial=False)

Set \{\{A, B\}\} of objects of the 2-dimensional topological manifold M

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

sage: M.declare_union(U,V)

sage: M.open_cover_family()

Set \{\{A, B, V\}, \{M\}, \{U, V\}\} of objects of the 2-dimensional topological manifold M

open_covers (trivial=True, supersets=False)

Generate the 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 a \textit{ManifoldSubsetFiniteFamily} \(F\) of open subsets \(U \in F\) of \(M\) such that

\[ A \subset \bigcup_{U \in F} U. \]

If \(A\) is open, we ask that the above inclusion is actually an identity:

\[ A = \bigcup_{U \in F} U. \]

\textbf{Note:} To get the open covers as a family, sorted lexicographically by the names of the subsets forming the open covers, use the method \textit{open_cover_family()} instead.

\textbf{INPUT:}

- \texttt{trivial} – (default: True) if \texttt{self} is open, include the trivial open cover of \texttt{self} by itself
- \texttt{supersets} – (default: False) if True, include open covers of all the supersets; it can also be an iterable of supersets to include

\textbf{EXAMPLES:}

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

sage: M.open_covers()
<generator ...>

sage: list(M.open_covers())

[Set {M} of open subsets of the 2-dimensional topological manifold M]

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

sage: list(U.open_covers())

[Set {U} of open subsets 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: list(U.open_covers())

[Set {U} of open subsets of the 2-dimensional topological manifold M, Set {A, B} of open subsets of the 2-dimensional topological manifold M]

sage: list(U.open_covers(trivial=False))

[Set {A, B} of open subsets of the 2-dimensional topological manifold M]

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

sage: M.declare_union(U,V)

sage: list(M.open_covers())

[Set {M} of open subsets of the 2-dimensional topological manifold M, Set {U, V} of open subsets of the 2-dimensional topological manifold M, Set {A, B, V} of open subsets of the 2-dimensional topological manifold M]

1.2. Subsets of Topological Manifolds 55
**open_subset** *(name, latex_name=None, coord_def={}, supersets=None)*

Create an open subset of the manifold that is a subset of *self*.

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
- **supersets** – (default: only *self*) list of sets that the new open subset is a subset of

**OUTPUT:**

- the open subset, as an instance of *TopologicalManifold* or one of its subclasses

**EXAMPLES:**

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

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

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

**open_superset_family**()

Return the family of open supersets of *self*. 
The family is sorted by the alphabetical names of the subsets.

OUTPUT:
• a *ManifoldSubsetFiniteFamily* instance containing all the open supersets that have been defined on the current subset

**Note:** If you only need to iterate over the open supersets in arbitrary order, you can use the generator method *open_supersets()* instead.

**EXAMPLES:**
```sage
M = Manifold(2, 'M', structure='topological')
U = M.open_subset('U')
V = U.subset('V')
W = V.subset('W')
W.open_superset_family()
Set (M, U) of open subsets of the 2-dimensional topological manifold M
```

**open_supersets()**
Generate the open supersets of *self*.

**Note:** To get the open supersets as a family, sorted by name, use the method *open_superset_family()* instead.

**EXAMPLES:**
```sage
M = Manifold(2, 'M', structure='topological')
U = M.open_subset('U')
V = U.subset('V')
W = V.subset('W')
V.open_superset_family()
Set (M, U) of open subsets 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
```

subset_digraph (loops=False, quotient=False, open_covers=False, points=False, lower_bound=None)

Return the digraph whose arcs represent subset relations among the subsets of self.

INPUT:
• loops – (default: False) whether to include the trivial containment of each subset in itself as loops of the digraph
• quotient – (default: False) whether to contract directed cycles in the graph, replacing equivalence classes of equal subsets by a single vertex. In this case, each vertex of the digraph is a set of ManifoldSubset instances.
• open_covers – (default: False) whether to include vertices for open covers
• points – (default: False) whether to include vertices for declared points; this can also be an iterable for the points to include
• lower_bound – (default: None) only include supersets of this

OUTPUT:
A digraph. Each vertex of the digraph is either:
• a ManifoldSubsetFiniteFamily containing one instance of ManifoldSubset.
• (if open_covers is True) a tuple of ManifoldSubsetFiniteFamily instances, representing an open cover.

EXAMPLES:
If open_covers is True, the digraph includes a special vertex for each nontrivial open cover of a subset:

```
sage: D = M.subset_digraph(open_covers=True)  # needs sage.graphs
sage: D.relabel(label, inplace=False).plot(layout='acyclic')  # needs sage.graphs sage.plot
Graphics object consisting of 14 graphics primitives
```

```subset_family()`

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

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

**OUTPUT:**

- a `ManifoldSubsetFiniteFamily` instance containing all the subsets that have been defined on the current subset

**Note:** If you only need to iterate over the subsets in arbitrary order, you can use the generator method `subsets()` instead.

**EXAMPLES:**

Subsets of a 2-dimensional manifold:
```python
sage: M = Manifold(2, 'M', structure='topological')
sage: U = M.open_subset('U')
sage: V = M.subset('V')
sage: M.subset_family()
Set {M, U, V} of subsets of the 2-dimensional topological manifold M
```

**subset_poset** *(open_covers=False, points=False, lower_bound=None)*

Return the poset of equivalence classes of the subsets of self.

Each element of the poset is a set of *ManifoldSubset* instances, which are known to be equal.

**INPUT:**

- **open_covers** – (default: False) whether to include vertices for open covers
- **points** – (default: False) whether to include vertices for declared points; this can also be an iterable for the points to include
- **lower_bound** – (default: None) only include supersets of this

**EXAMPLES:**

```python
sage: # needs sage.graphs
sage: M = Manifold(3, 'M')
sage: U = M.open_subset('U'); V = M.open_subset('V'); W = M.open_subset('W')
sage: P = M.subset_poset(); P
Finite poset containing 4 elements
```

```
# needs sage.plot
Graphics object consisting of 8 graphics primitives
```

```python
sage: VW = V.union(W)
sage: P = M.subset_poset(); P
Finite poset containing 5 elements
```

```
Set (M) of open subsets of the 3-dimensional differentiable manifold M
```

```python
sage: from sage.manifolds.subset import ManifoldSubsetFiniteFamily
sage: sorted(P.lower_covers(ManifoldSubsetFiniteFamily([M])), key=str)
```

(continues on next page)
[Set \{U\} of open subsets of the 3-dimensional differentiable manifold \(M\),
Set \{V_{\text{union}}_W\} of open subsets of the 3-dimensional differentiable manifold \(M\)]

sage: P.plot(element_labels={element: element._name for element in P})  # needs sage.plot
Graphics object consisting of 10 graphics primitives

If `open_covers` is `True`, the poset includes a special vertex for each nontrivial open cover of a subset:

sage: # needs sage.graphs
sage: P = M.subset_poset(open_covers=True); P
Finite poset containing 6 elements
sage: from sage.manifolds.subset import ManifoldSubsetFiniteFamily
sage: sorted(P.upper_covers(ManifoldSubsetFiniteFamily([VW])), key=str)
[(Set \{V\} of open subsets of the 3-dimensional differentiable manifold \(M\),
  Set \{W\} of open subsets of the 3-dimensional differentiable manifold \(M\)),
Set \{M\} of open subsets of the 3-dimensional differentiable manifold \(M\)]

sage: def label(element):
....:     try:
....:         return element._name
....:     except AttributeError:
....:         return '+,'.join(sorted(x._name for x in element)) + ']

sage: P.plot(element_labels={element: label(element) for element in P})  # needs sage.plot
Graphics object consisting of 12 graphics primitives

`subsets()`

Generate the subsets that have been defined on the current subset.

**Note:** To get the subsets as a family, sorted by name, use the method `subset_family()` 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')
The method `subset_family()` returns a family (sorted alphabetically by the subset names):

```sage```
M.subset_family()
```
Set (M, U, V) of subsets 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.subset_family()
Set (A, B) of subsets of the 2-dimensional topological manifold M
sage: a.superset_family()
Set (A, B, M) of subsets of the 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

**superset_digraph** *(loops=False, quotient=False, open_covers=False, points=False, upper_bound=None)*

Return the digraph whose arcs represent subset relations among the supersets of `self`.

**INPUT:**
- **loops** – (default: False) whether to include the trivial containment of each subset in itself as loops of the digraph

- **quotient** – (default: False) whether to contract directed cycles in the graph, replacing equivalence classes of equal subsets by a single vertex. In this case, each vertex of the digraph is a set of `ManifoldSubset` instances.

- **open_covers** – (default: False) whether to include vertices for open covers

- **points** – (default: False) whether to include vertices for declared points; this can also be an iterable for the points to include

- **upper_bound** – (default: None) only include subsets of this

**EXAMPLES:**

```
sage: M = Manifold(3, 'M')
sage: U = M.open_subset('U'); V = M.open_subset('V'); W = M.open_subset('W')
sage: VW = V.union(W)
sage: P = V.superset_digraph(loops=False, upper_bound=W); P
Digraph on 2 vertices
```

**superset_family** ()

Return the family of declared supersets of the current subset.

The family is sorted by the alphabetical names of the supersets.

**OUTPUT:**

- a `ManifoldSubsetFiniteFamily` instance containing all the supersets

**Note:** If you only need to iterate over the supersets in arbitrary order, you can use the generator method `supersets()` instead.

**EXAMPLES:**

```
sage: M = Manifold(2, 'M', structure='topological')
sage: U = M.open_subset('U')
sage: V = M.subset('V')
sage: V.superset_family()
Set {M, V} of subsets of the 2-dimensional topological manifold M
```

**superset_poset** *(open_covers=False, points=False, upper_bound=None)*

Return the poset of the supersets of self.

**INPUT:**

- **open_covers** – (default: False) whether to include vertices for open covers

- **points** – (default: False) whether to include vertices for declared points; this can also be an iterable for the points to include

- **upper_bound** – (default: None) only include subsets of this

**EXAMPLES:**

```
sage: M = Manifold(3, 'M')
sage: U = M.open_subset('U'); V = M.open_subset('V'); W = M.open_subset('W')
sage: VW = V.union(W)
```

(continues on next page)
supersets()  

Generate the declared supersets of the current subset.

Note: To get the supersets as a family, sorted by name, use the method \texttt{superset\_family()} instead.

EXAMPLES:

sage: M = Manifold(2, 'M', structure='topological')  
sage: U = M.open_subset('U')  
sage: V = M.subset('V')  
sage: sorted(V.supersets(), key = lambda v: v._name)  
[2-dimensional topological manifold M,  
Subset V of the 2-dimensional topological manifold M]

union\,(name, latex\_name, *others)

Return the union of the current subset with other subsets.

This method may return a previously constructed union instead of creating a new subset. In this case, name and latex\_name are not used.

INPUT:

- others – other subsets 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 \texttt{ManifoldSubset} representing the subset that is the union of the current subset with others

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.superset\_family()  
Set \{(A, A_union_B, M) of subsets of the 2-dimensional topological manifold M\}  
sage: b.superset\_family()  
Set \{(A_union_B, B, M) of subsets of the 2-dimensional topological manifold M\}  
sage: c.superset\_family()  
Set \{(A_union_B, M) of subsets of the 2-dimensional topological manifold M\}
Union of six subsets:

```python
sage: T = Manifold(2, 'T', structure='topological')
sage: S = [T.subset(f'S{i}')] for i in range(6)]
sage: [S[i].union(S[i+3]) for i in range(3)]
[Subset S0_union_S3 of the 2-dimensional topological manifold T,
 Subset S1_union_S4 of the 2-dimensional topological manifold T,
 Subset S2_union_S5 of the 2-dimensional topological manifold T]
sage: union_S_i = S[0].union(S[1:], name='union_S_i'); union_S_i
Subset union_S_i of the 2-dimensional topological manifold T
sage: T.subset_family()
Set {S0, S0_union_S3, S0_union_S3_union_S1_union_S4, S1,
     S1_union_S4, S2, S2_union_S5, S3, S4, S5, T, union_S_i}
```

1.3 Manifold Structures

These classes encode the structure of a manifold.

AUTHORS:


class sage.manifolds.structure.DegenerateStructure
    Bases: Singleton

The structure of a degenerate manifold.

chart
    alias of `RealDiffChart`

homset
    alias of `DifferentiableManifoldHomset`

name = 'degenerate_metric'
**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 DegenerateStructure
sage: from sage.categories.manifolds import Manifolds
sage: DegenerateStructure().subcategory(Manifolds(RR))
Category of manifolds over Real Field with 53 bits of precision
```

---

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

**Bases:** `Singleton`

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

**chart**

alias of `DiffChart`

**homset**

alias of `DifferentiableManifoldHomset`

**name** = `'differentiable'`

**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.LorentzianStructure`

**Bases:** `Singleton`

The structure of a Lorentzian manifold.

**chart**

alias of `RealDiffChart`

**homset**

alias of `DifferentiableManifoldHomset`

**name** = `'Lorentzian'`

**scalar_field_algebra**

alias of `DiffScalarFieldAlgebra`

**subcategory** *(cat)*

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

**EXAMPLES:**

---

1.3. Manifold Structures 67
class sage.manifolds.structure.PseudoRiemannianStructure

Bases: Singleton

The structure of a pseudo-Riemannian manifold.

chart
    alias of RealDiffChart

homset
    alias of DifferentiableManifoldHomset

name = 'pseudo-Riemannian'

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

class sage.manifolds.structure.RealDifferentialStructure

Bases: Singleton

The structure of a differentiable manifold over \( \mathbb{R} \).

chart
    alias of RealDiffChart

homset
    alias of DifferentiableManifoldHomset

name = 'differentiable'

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 RealDifferentialStructure
sage: from sage.categories.manifolds import Manifolds
sage: RealDifferentialStructure().subcategory(Manifolds(RR))
Category of manifolds over Real Field with 53 bits of precision
class sage.manifolds.structure.RealTopologicalStructure
    Bases: Singleton
    
The structure of a topological manifold over \( \mathbb{R} \).
    
chart
        alias of RealChart
    
homset
        alias of TopologicalManifoldHomset
    
name = 'topological'
    
scalar_field_algebra
        alias of ScalarFieldAlgebra
    
subcategory (cat)
        Return the subcategory of \( \text{cat} \) corresponding to the structure of \( \text{self} \).
        
EXAMPLES:

```sage
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.RiemannianStructure
    Bases: Singleton
    
The structure of a Riemannian manifold.
    
chart
        alias of RealDiffChart
    
homset
        alias of DifferentiableManifoldHomset
    
name = 'Riemannian'
    
scalar_field_algebra
        alias of DiffScalarFieldAlgebra
    
subcategory (cat)
        Return the subcategory of \( \text{cat} \) corresponding to the structure of \( \text{self} \).
        
EXAMPLES:

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

class sage.manifolds.structure.TopologicalStructure
    Bases: Singleton
    
The structure of a topological manifold over a general topological field.
    
chart
        alias of Chart

1.3. Manifold Structures
homset
    alias of TopologicalManifoldHomset
name = 'topological'

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 domain of spherical coordinates
sage: c_spher.<r,th,ph> = U.chart(rr:(0,+oo) th:(0,pi): θ ph:(0,2*pi):periodic:
˓ →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:

```python
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:

```python
sage: p1 = U((1, pi /2, pi))
sage: p1 == p
True
sage: q = U((2, pi /2, pi))
sage: q == p
False
```

Even if they were initially not defined within the same coordinate chart:

```python
sage: p2 = U((-1,0,0), chart=c_cart)
sage: p2 == p
True
```

The \( 2\pi \)-periodicity of the \( \phi \) coordinate is also taken into account for the comparison:

```python
sage: p3 = U((1, pi /2, 5*pi))
sage: p3 == p
True
sage: p4 = U((1, pi /2, -pi))
sage: p4 == p
True
```

class `sage.manifolds.point.ManifoldPoint` (``parent``, ``coords=None``, ``chart=None``, ``name=None``, 
``latex_name=None``, ``check_coords=True``)

**Bases:** `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:
Manifolds, Release 10.3

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

But it can be set to any value:

```sage
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()`.

```sage
add_coord (coords, chart=None)
```

Add 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:

```python
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:

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

A shortcut for `add_coordinates` is `add_coord`:

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

Let us introduce a second chart on the manifold:

```python
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:

```python
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:

```python
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:
Manifolds, Release 10.3

```python
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:

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

A shortcut for `add_coordinates` is `add_coord`:

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

Let us introduce a second chart on the manifold:

```python
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:

```python
sage: p.add_coordinates((-1,5), chart=Y)
sage: p._coordinates
{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:

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

```python
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 $\mathbb{R}^3$: ```
sage: M = Manifold(3, 'M', structure='topological')
sage: c_spher.<r,th,ph> = M.chart(r'r:(0,+oo) th:(0,pi):
θ 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:

1.4. Points of Topological Manifolds
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),
 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 $\mathbb{R}^3$:

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))

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:
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)])
```

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)}
```

```
plot(plot=None, ambient_coords=None, mapping=None, label=None, parameters=None, size=10, 
color='black', label_color=None, fontsize=10, label_offset=0.1, **kwds)
```
For real manifolds, plot $\text{self}$ 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:**

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

**OUTPUT:**

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

**EXAMPLES:**

Drawing a point on a 2-dimensional manifold:

```
sage: # needs sage.plot
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 $\text{plot}$:

```
sage: # needs sage.plot
sage: g = p.plot()
sage: g + gX
Graphics object consisting of 20 graphics primitives
```
1.4. Points of Topological Manifolds
Call with some options:

```
sage: # needs sage.plot
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
```

```
\begin{center}
\begin{tikzpicture}
\begin{axis}[
    view={0}{90},
    axis equal image=true,
    axis lines=middle,
    grid=both,
    grid style={line width=.1pt, draw=gray!10},
    major grid style={line width=.2pt,draw=gray!50},
    xmin=-4, xmax=4,
    ymin=-4, ymax=4,
    xtick={-4,-3,...,4},
    ytick={-4,-3,...,4},
    xlabel=$x$,
    ylabel=$y$,
]
\fill [green] (1,2) circle (2pt) node [right] (P) {$P$};
\end{axis}
\end{tikzpicture}
\end{center}
```

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')
# needs sage.plot
sage: gq = q.plot(parameters={a:-2}, label_offset=0.2)
# needs sage.plot
sage: g + gX + gq
# needs sage.plot
```

```
Graphics object consisting of 22 graphics primitives
```

The numerical value is used only for the plot:

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

Drawing a point on a 3-dimensional manifold:
1.4. Points of Topological Manifolds
sage: # needs sage.plot
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()
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)

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'\theta:(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: # needs sage.plot
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

Chapter 1. Topological Manifolds
1.4. Points of Topological Manifolds
**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 \( K = \mathbb{R} \), 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:
- Travis Scrimshaw (2015): review tweaks
- Eric Gourgoulhon (2019): periodic coordinates, add `calculus_method()`

REFERENCES:
- Chap. 2 of [Lee2011]
- Chap. 1 of [Lee2013]
class sage.manifolds.chart.Chart(domain, coordinates, calc_method=None, periods=None, coord_restrictions=None)

Bases: UniqueRepresentation, 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 $(x^1, \ldots, x^n)$ of $\varphi$, defined by $\varphi(p) = (x^1(p), \ldots, x^n(p)) \in K^n$ 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: "") single string defining the coordinate symbols, with " " (whitespace) as a separator; each item has at most three fields, separated by a colon (:):
  1. the coordinate symbol (a letter or a few letters)
  2. (optional) the period of the coordinate if the coordinate is periodic; the period field must be written as `period=T`, where $T$ is the period (see examples below)
  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 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's backslash character (see examples below). If no period and no LaTeX spelling are to be set for any coordinate, the argument `coordinates` can be omitted when the shortcut operator `<,>` is used to declare the chart (see examples below).

- **calc_method** – (default: None) string defining the calculus method for computations involving coordinates of the chart; must be one of
  - 'SR': Sage’s default symbolic engine (Symbolic Ring)
  - 'sympy': SymPy
  - None: the default of `CalculusMethod` will be used

- **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)

- **coord_restrictions**: Additional restrictions on the coordinates. 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 (or set or frozenset) `coord_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 (or set or frozenset) `coord_restrictions`. For example:

\[
\text{coord_restrictions}=\{x > y, (x != 0, y != 0), (z^2 < x)\}
\]

means $(x > y)$ and $((x != 0) \text{ or } (y != 0))$ and $(z^2 < x)$. If the list `coord_restrictions` contains only one item, this item can be passed as such, i.e. writing $x > y$ instead of the single element list $[x > y]$. If the chart variables have not been declared as variables yet, `coord_restrictions` must be lambda-quoted.

EXAMPLES:

A chart on a complex 2-dimensional topological manifold:
To manipulate the coordinates \((x, y)\) as global variables, one has to set:

```python
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()):

```python
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:

```python
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:

```python
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 \(y1\):

```python
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:

```python
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 \(M\):

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

Of course, one may declare a chart only on an open subset of \(M\):
In the above declaration, we have also specified some LaTeX writing of the coordinates different from the text one:

```python
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.

Periodic coordinates are declared through the keyword `period` in the coordinate field:

```python
sage: N = Manifold(2, 'N', field='complex', structure='topological')
sage: XN.<Z1,Z2> = N.chart('Z1:period=1+2*I Z2')
sage: XN.periods()
(2*I + 1, None)
```

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

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

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

```python
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`:

```python
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 `[:]`:

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

Some partial sets of coordinates:

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

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

```python
sage: M.atlas()
[Chart (M, (x, y)), Chart (U, (z1, z2))]
```
Manifolds, Release 10.3

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 a 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
```

Setting additional coordinate restrictions:

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

See also:

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

**add_restrictions** *(restrictions)*

Add some restrictions on the coordinates.

This is deprecated; provide the restrictions at the time of creating the chart.

**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)
doctest:warning...
DeprecationWarning: Chart.add_restrictions is deprecated; provide the
restrictions at the time of creating the chart
See https://github.com/sagemath/sage/issues/32102 for details.
sage: X.valid_coordinates(2+i, 1)
True
sage: X.valid_coordinates(i, 1)
False

calculus_method()

Return the interface governing the calculus engine for expressions involving coordinates of this chart.

The calculus engine can be one of the following:

- Sage’s symbolic engine (Pynac + Maxima), implemented via the Symbolic Ring SR
- SymPy

See also:

CalculusMethod for a complete documentation.

OUTPUT:

- an instance of CalculusMethod

EXAMPLES:

The default calculus method relies on Sage’s Symbolic Ring:

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: X.calculus_method()
Available calculus methods (* = current):
- SR (default) (*)
- sympy

Accordingly the method expr() of a function $f$ defined on the chart $X$ returns a Sage symbolic expression:

sage: f = X.function(x^2 + cos(y)*sin(x))
sage: f.expr()
x^2 + cos(y)*sin(x)
sage: type(f.expr())
<class 'sage.symbolic.expression.Expression'>
sage: parent(f.expr())
Symbolic Ring
sage: f.display()
(x, y) ↦ x^2 + cos(y)*sin(x)

Changing to SymPy:

sage: X.calculus_method().set('sympy')
sage: f.expr()
x**2 + sin(x)*cos(y)
sage: type(f.expr())
<class 'sympy.core.add.Add'>
sage: parent(f.expr())

(continues on next page)
```python
<class 'sympy.core.add.Add'>
sage: f.display()
(x, y) ↦ x**2 + sin(x)*cos(y)
```

Back to the Symbolic Ring:

```python
sage: X.calculus_method().set('SR')
sage: f.display()
(x, y) ↦ x^2 + cos(y)*sin(x)
```

codomain()

Return the codomain of self as a set.

EXAMPLES:

```python
sage: M = Manifold(2, 'M', field='complex', structure='topological')
sage: X.<x,y> = M.chart()
sage: X.codomain()
Vector space of dimension 2 over Complex Field with 53 bits of precision
```

domain()

Return the open subset on which the chart is defined.

EXAMPLES:

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

function(expression, calc_method=None, expansion_symbol=None, order=None)

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

$$f : V \subset K^n \rightarrow K$$

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

INPUT:

- expression – a symbolic expression involving the chart coordinates, to represent $f(x^1, \ldots, x^n)$
- calc_method – string (default: None): the calculus method with respect to which the internal expression of the function must be initialized from expression; one of
  - 'SR': Sage's default symbolic engine (Symbolic Ring)
  - 'sympy': SymPy
  - None: the chart current calculus method is assumed
- expansion_symbol – (default: None) symbolic variable (the "small parameter") with respect to which the coordinate expression is expanded in power series (around the zero value of this variable)
• order – integer (default: None); the order of the expansion if `expansion_symbol` is not None; the order is defined as the degree of the polynomial representing the truncated power series in `expansion_symbol`.

**Warning:** The value of `order` is \( n - 1 \), where \( n \) is the order of the big \( O \) in the power series expansion

**OUTPUT:**

• instance of `ChartFunction` representing the coordinate function \( f \)

**EXAMPLES:**

A symbolic coordinate function:

```sage
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.chart_func.ChartFunctionRing_with_category.element_class'>
sage: f.display()
(\( x, y \) \( \mapsto \) \( \sin(x*y) \))
sage: f(2,3)
sin(6)
```

Using SymPy for the internal representation of the function (dictionary `_express`):

```sage
sage: g = X.function(x**2 + x*cos(y), calc_method='sympy')
sage: g._express
{'sympy': x**2 + x*cos(y)}
```

On the contrary, for \( f \), only the SR part has been initialized:

```sage
sage: f._express
{'SR': sin(x*y)}
```

See `ChartFunction` for more examples.

**function_ring()**

Return the ring of coordinate functions on `self`.

**EXAMPLES:**

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

**manifold()**

Return the manifold on which the chart is defined.

**EXAMPLES:**
```python
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

\[
f : V \subset K^n \longrightarrow K^m
(x^1, \ldots, x^n) \longmapsto (f_1(x^1, \ldots, x^n), \ldots, f_m(x^1, \ldots, x^n)),
\]

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 \( \mathbb{R}^3 \):

```python
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

\[
f : V \subset K^n \longrightarrow K
(x^1, \ldots, x^n) \longmapsto 1,
\]

where \( V \) is the chart codomain.

See class `ChartFunction` for a complete documentation.

**OUTPUT:**

- a `ChartFunction` 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()
\(x, y) \mapsto 1\)
sage: type(X.one_function())
<class 'sage.manifolds.chart_func.ChartFunctionRing_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: # needs sage.rings.padics
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) \mapsto 1 + O(5^{20})

periods()
Return the coordinate periods.

OUTPUT:

- a tuple containing the period of each coordinate, with the value None if the coordinate is not periodic

EXAMPLES:

A chart without any periodic coordinate:

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: X.periods()
(None, None)

Charts with a periodic coordinate:

sage: Y.<u,v> = M.chart("u v:0,2*pi:periodic")
sage: Y.periods()
(0, 2*pi)

Complex manifold with a periodic coordinate:

sage: M = Manifold(2, 'M', field='complex', structure='topological', ...
.....:    start_index=1)
sage: X.<x,y> = M.chart("x y:period=1+I")
sage: X.periods()
(None, I + 1)
**preimage** \( (\text{codomain_subset, name=None, latex_name=None}) \)

Return the preimage (pullback) of \( \text{codomain_subset} \) under \( \text{self} \).

It is the subset of the domain of \( \text{self} \) formed by the points whose coordinate vectors lie in \( \text{codomain_subset} \).

**INPUT:**

- \( \text{codomain_subset} \) - an instance of `ConvexSet_base` or another object with a `__contains__` method that accepts coordinate vectors
- \( \text{name} \) – string; name (symbol) given to the subset
- \( \text{latex_name} \) – (default: None) string; LaTeX symbol to denote the subset; if none are provided, it is set to name

**OUTPUT:**

- either a `TopologicalManifold` or a `ManifoldSubsetPullback`

**EXAMPLES:**

```
sage: M = Manifold(2, 'R^2', structure='topological')
sage: c_cart.<x,y> = M.chart() # Cartesian coordinates on R^2

Pulling back a polytope under a chart:

```sage```
# needs sage.geometry.polyhedron
sage: P = Polyhedron(vertices=[[0, 0], [1, 2], [2, 1]]); P
A 2-dimensional polyhedron in ZZ^2 defined as the convex hull of 3 vertices
sage: McP = c_cart.preimage(P); McP
Subset x_y_inv_P of the 2-dimensional topological manifold R^2
sage: M((1, 2)) in McP
True
sage: M((2, 0)) in McP
False
```

Pulling back the interior of a polytope under a chart:

```sage```
# needs sage.geometry.polyhedron
sage: int_P = P.interior(); int_P
Relative interior of
a 2-dimensional polyhedron in ZZ^2 defined as the convex hull of 3 vertices
sage: McInt_P = c_cart.preimage(int_P, name='McInt_P'); McInt_P
Open subset McInt_P of the 2-dimensional topological manifold R^2
sage: M((0, 0)) in McInt_P
False
sage: M((1, 1)) in McInt_P
True
```

Pulling back a point lattice:

```sage```
W = span([[1, 0], [3, 5]], ZZ); W
Free module of degree 2 and rank 2 over Integer Ring
Echelon basis matrix:
\[
\begin{pmatrix}
1 & 0 \\
0 & 5 \\
\end{pmatrix}
\]
```
sage```
McW = c_cart.pullback(W, name='McW'); McW
Subset McW of the 2-dimensional topological manifold R^2
sage: M((4, 5)) in McW
```

(continues on next page)
True
\texttt{sage: M((4, 4)) in McW}
False

Pulling back a real vector subspaces:

\texttt{sage: V = span([[1, 2]], RR); V}
Vector space of degree 2 and dimension 1 over Real Field with 53 bits of...
\[ \rightarrow \]
Basis matrix:
\[
\begin{bmatrix}
1.00000000000000 & 2.00000000000000
\end{bmatrix}
\]
\texttt{sage: McV = c_cart.pullback(V, name =\texttt{McV}); McV}
Subset \texttt{McV} of the 2-dimensional topological manifold \(\mathbb{R}^2\)
\texttt{sage: M((2, 4)) in McV}
True
\texttt{sage: M((1, 0)) in McV}
False

Pulling back a finite set of points:

\texttt{sage: F = Family([vector(QQ, [1, 2], immutable=True),
....: vector(QQ, [2, 3], immutable=True)])}
\texttt{sage: McF = c_cart.pullback(F, name =\texttt{McF}); McF}
Subset \texttt{McF} of the 2-dimensional topological manifold \(\mathbb{R}^2\)
\texttt{sage: M((2, 3)) in McF}
True
\texttt{sage: M((0, 0)) in McF}
False

Pulling back the integers:

\texttt{sage: R = manifolds.RealLine(); R}
Real number line \(\mathbb{R}\)
\texttt{sage: McZ = R.canonical_chart().pullback(ZZ, name =\texttt{Z}); McZ}
Subset \texttt{Z} of the Real number line \(\mathbb{R}\)
\texttt{sage: R((3/2,)) in McZ}
False
\texttt{sage: R((-2,)) in McZ}
True

\texttt{pullback (codomain_subset, name=None, latex_name=None)}

Return the preimage (pullback) of \texttt{codomain_subset} under \texttt{self}.

It is the subset of the domain of \texttt{self} formed by the points whose coordinate vectors lie in \texttt{codomain_subset}.

INPUT:

\begin{itemize}
  \item \texttt{codomain_subset} - an instance of \texttt{ConvexSet_base} or another object with a \texttt{__contains__} method that accepts coordinate vectors
  \item \texttt{name} - string; name (symbol) given to the subset
  \item \texttt{latex_name} - (default: None) string; \LaTeX{} symbol to denote the subset; if none are provided, it is set to \texttt{name}
\end{itemize}

OUTPUT:

\begin{itemize}
  \item either a \texttt{TopologicalManifold} or a \texttt{ManifoldSubsetPullback}
EXAMPLES:

```
sage: M = Manifold(2, 'R^2', structure='topological')
sage: c_cart.<x,y> = M.chart() # Cartesian coordinates on R^2
```

Pulling back a polytope under a chart:

```
sage: # needs sage.geometry.polyhedron
sage: P = Polyhedron(vertices=[[0, 0], [1, 2], [2, 1]]); P
A 2-dimensional polyhedron in ZZ^2 defined as the convex hull of 3 vertices
sage: McP = c_cart.preimage(P); McP
Subset x_y_inv_P of the 2-dimensional topological manifold R^2
sage: M((1, 2)) in McP
True
sage: M((2, 0)) in McP
False
```

Pulling back the interior of a polytope under a chart:

```
sage: # needs sage.geometry.polyhedron
sage: int_P = P.interior(); int_P
Relative interior of a 2-dimensional polyhedron in ZZ^2 defined as the convex hull of 3 vertices
sage: McInt_P = c_cart.preimage(int_P, name='McInt_P'); McInt_P
Open subset McInt_P of the 2-dimensional topological manifold R^2
sage: M((0, 0)) in McInt_P
False
sage: M((1, 1)) in McInt_P
True
```

Pulling back a point lattice:

```
sage: W = span([[1, 0], [3, 5]], ZZ); W
Free module of degree 2 and rank 2 over Integer Ring
Echelon basis matrix:
[1 0]
[0 5]
sage: McW = c_cart.pullback(W, name='McW'); McW
Subset McW of the 2-dimensional topological manifold R^2
sage: M((4, 5)) in McW
True
sage: M((4, 4)) in McW
False
```

Pulling back a real vector subspaces:

```
sage: V = span([[1, 2]], RR); V
Vector space of degree 2 and dimension 1 over Real Field with 53 bits of
˓→precision
Basis matrix:
[1.00000000000000 2.00000000000000]
sage: McV = c_cart.pullback(V, name='McV'); McV
Subset McV of the 2-dimensional topological manifold R^2
sage: M((2, 4)) in McV
True
sage: M((1, 0)) in McV
False
```

Pulling back a finite set of points:

```
```

1.5. Coordinate Charts

97
Pulling back the integers:

```
sage: R = manifolds.RealLine(); R
Real number line ℝ
sage: McZ = R.canonical_chart().pullback(ZZ, name='ℤ'); McZ
Subset ℤ of the Real number line ℝ
sage: R((3/2,)) in McZ
False
sage: R((-2,)) in McZ
True
```

\textbf{restrict (subset, restrictions=None)}

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

If the current chart is \((U, \varphi)\), a \textit{restriction} (or \textit{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:

- \texttt{subset} – open subset \(V\) of the chart domain \(U\) (must be an instance of \texttt{TopologicalManifold})
- \texttt{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 \texttt{restrictions} are combined with the \texttt{and} operator; if some restrictions are to be combined with the \texttt{or} operator instead, they have to be passed as a tuple in some single item of the list \texttt{restrictions}. For example:

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

means \((x > y) \land ((x \neq 0) \lor (y \neq 0))\) and \((z^2 < x)\). If the list \texttt{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 \texttt{Chart}

EXAMPLES:

Coordinates on the unit open ball of \(C^2\) as a subchart of the global coordinates of \(C^2\):

```
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

\[
\psi \circ \varphi^{-1} : \varphi(U \cap V) \subset K^n \to \psi(U \cap V) \subset K^n,
\]

where \(K\) is the manifold's base field. In other words, the transition map expresses the coordinates \((y^1, \ldots, y^n)\) of \((V, \psi)\) in terms of the coordinates \((x^1, \ldots, x^n)\) 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) \((Y_1, \ldots, Y_n)\), where \(Y_i\) is the symbolic expression of the coordinate \(y^i\) in terms of the coordinates \((x^1, \ldots, x^n)\)
* 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 \texttt{and} operator; if some restrictions are to be combined with the \texttt{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 \((x > y) \text{ and } ((x != 0) \text{ or } (y != 0)) \text{ and } (z^2 < x)\). 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 \texttt{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():

1.5. Coordinate Charts 99
sage: F = M.subset_family(); F
Set {S^1, U, V, W} of open subsets of the 1-dimensional topological manifold...
˓→S^1
sage: W = F[W]
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 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(rr:(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 ∩ M = U:

sage: M.subset_family()
Set {R^2, U} of open subsets 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(coord_restrictions=\lambda x,y: [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)
(continues on next page)
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(coord_restrictions=lambda u,v: abs(v)<a)
sage: Y.valid_coordinates(0, i, parameters={a: 2})  # setting a=2
True
sage: Y.valid_coordinates(0, 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

\[
f : V \subset K^n \rightarrow K \quad (x^1, \ldots, x^n) \mapsto 0,
\]

where \(V\) is the chart codomain.

See class \texttt{ChartFunction} for a complete documentation.

**OUTPUT:**

- a \texttt{ChartFunction} 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.chart_func.ChartFunctionRing_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: # needs sage.rings.padics
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
```

(continues on next page)
class sage.manifolds.chart.CoordChange(chart1, chart2, *transformations)

Bases: 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 \((y^1, \ldots, y^n)\) of \((V, \psi)\) in terms of the coordinates \((x^1, \ldots, x^n)\) 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) \(\{Y_1, \ldots, Y_2\}\), where \(Y_i\) is the symbolic expression of the coordinate \(y^i\) in terms of the coordinates \((x^1, \ldots, x^n)\)

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(rr:(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.
```

```
\left\{\begin{array}{lcl}
x & = & r \cos\left({\phi}\right) \\
y & = & r \sin\left({\phi}\right) \\
\end{array}\right.
```
A shortcut is \texttt{disp()}: 

\begin{verbatim}
sage: spher_to_cart.disp()
x = r*cos(ph)
y = r*sin(ph)
\end{verbatim}

\texttt{disp()}

Display of the coordinate transformation.

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

\textbf{EXAMPLES:}

From spherical coordinates to Cartesian ones in the plane:

\begin{verbatim}
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(rr:(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.
\end{verbatim}

A shortcut is \texttt{disp()}: 

\begin{verbatim}
sage: spher_to_cart.disp()
x = r*cos(ph)
y = r*sin(ph)
\end{verbatim}

\texttt{inverse()}

Return the inverse coordinate transformation.

If the inverse is not already known, it is computed here. If the computation fails, the inverse can be set by hand via the method \texttt{set_inverse()}. 

\textbf{OUTPUT:}

\begin{itemize}
  \item an instance of \texttt{CoordChange} representing the inverse of the current coordinate transformation
\end{itemize}

\textbf{EXAMPLES:}

Inverse of a coordinate transformation corresponding to a rotation in the Cartesian plane:

\begin{verbatim}
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: c_uv.<u,v> = M.chart()
sage: phi = var('phi', domain='real')
sage: xy_to_uv = c_xy.transition_map(c_uv, 
    ....:    [cos(phi)*x + sin(phi)*y, 
    ....:     -sin(phi)*x + cos(phi)*y])
sage: M.coord_changes()
{(Chart (M, (x, y)), 
  Chart (M, (u, v))): Change of coordinates from Chart (M, (x, y)) to Chart ...
  \rightarrow (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))
\end{verbatim}

(continues on next page)
sage: uv_to_xy.display()
x = u*cos(phi) - v*sin(phi)
y = v*cos(phi) + u*sin(phi)
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...
    \rightarrow (M, (x, y)),
   (Chart (M, (x, y)),
   Chart (M, (u, v))): Change of coordinates from Chart (M, (x, y)) to Chart...
    \rightarrow (M, (u, v))}

The result is cached:

sage: xy_to_uv.inverse() is uv_to_xy
True

We have as well:

sage: uv_to_xy.inverse() is xy_to_uv
True

**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, **kwds)*

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

• **kwds** – optional arguments; valid keywords are
  
  – **check** (default: True) – boolean determining whether the provided transformations are checked to be indeed the inverse coordinate transformations
  
  – **verbose** (default: False) – boolean determining whether some details of the check are printed out; if False, no output is printed if the check is passed (see example below)

**EXAMPLES:**

From spherical coordinates to Cartesian ones in the plane:

```
sage: M = Manifold(2, 'R^2', structure='topological')
sage: U = M.open_subset('U') # complement of the half line {y=0, x>= 0}
sage: c_cart.<x,y> = U.chart()
sage: c_spher.<r,ph> = U.chart(rr:(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))
```

Check of the inverse coordinate transformation:

- \( r = r \) *passed*
- \( \text{ph} = \text{arctan2}(r*\text{sin}(\text{ph}), r*\text{cos}(\text{ph})) \) **failed**
- \( x = x \) *passed*
- \( y = y \) *passed*

NB: a failed report can reflect a mere lack of simplification.

As indicated, the failure for \( \text{ph} \) is due to a lack of simplification of the \text{arctan2} term, not to any error in the provided inverse formulas.

We have now:

```
sage: spher_to_cart.inverse()  
```

Change of coordinates from Chart \((U, (x, y))\) to Chart \((U, (r, \text{ph}))\)

```
sage: spher_to_cart.inverse().display()  
```

\( r = \sqrt{x^2 + y^2} \)

\( \text{ph} = \text{arctan2}(y, x) \)

```
sage: M.coord_changes()  
```

# random (dictionary output)

\{
  (Chart \((U, (r, ph))\)),
  Change of coordinates from Chart \((U, (r, ph))\) to Chart \((U, (x, y))\),
  (Chart \((U, (x, y))\)),
  Change of coordinates from Chart \((U, (x, y))\) to Chart \((U, (r, ph))\))
\}

One can suppress the check of the provided formulas by means of the optional argument \texttt{check=False}:

```
sage: spher_to_cart.set_inverse(sqrt(x^2+y^2), atan2(y,x), 
  ....: check=False)
```

However, it is not recommended to do so, the check being (obviously) useful to avoid some mistake. For instance, if the term \( \sqrt{x^2+y^2} \) contains a typo \((x^3\) instead of \(x^2\)), we get:

```
sage: spher_to_cart.set_inverse(sqrt(x^3+y^2), atan2(y,x))
```

Check of the inverse coordinate transformation:

- \( r = \sqrt{r*\text{cos}(\text{ph})^3 + \text{sin}(\text{ph})^2}*r \) **failed**
- \( \text{ph} = \text{arctan2}(r*\text{sin}(\text{ph}), r*\text{cos}(\text{ph})) \) **failed**
- \( x = \sqrt{x^3 + y^2})*x/\sqrt{x^2 + y^2} \) **failed**

(continues on next page)
If the check is passed, no output is printed out:

```python
sage: M = Manifold(2, 'M')
sage: X1.<x,y> = M.chart()
sage: X2.<u,v> = M.chart()
sage: X1_to_X2 = X1.transition_map(X2, [x+y, x-y])
sage: X1_to_X2.set_inverse((u+v)/2, (u-v)/2)
```

unless the option `verbose` is set to `True`:

```python
sage: X1_to_X2.set_inverse((u+v)/2, (u-v)/2, verbose=True)
```

Check of the inverse coordinate transformation:

- x == x  *passed*
- y == y  *passed*
- u == u  *passed*
- v == v  *passed*

---

class sage.manifolds.chart.RealChart (domain, coordinates, calc_method=None, bounds=None, periods=None, coord_restrictions=None)

Bases: Chart

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

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

The components $(x^1, \ldots, x^n)$ of $\varphi$, defined by $\varphi(p) = (x^1(p), \ldots, x^n(p)) \in \mathbb{R}^n$ 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, with ' ' (whitespace) as a separator; each item has at most four 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 $\mathbb{R}$; otherwise $I$ must be provided in the form $[a, b]$ (or equivalently $]a, b[$); the bounds $a$ and $b$ can be $+/-\text{Infinity}, \text{Inf}, \text{infinity}, \text{inf}$ or $\text{oo}$; 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) indicator of the periodic character of the coordinate, either as `period=T`, where $T$ is the period, or as the keyword `periodic` (the value of the period is then deduced from the interval $I$ declared in field 2; see examples below)
  4. (optional) the LaTeX spelling of the coordinate; if not provided the coordinate symbol given in the first field will be used

The order of fields 2 to 4 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’s backslash character (see examples below). If interval range, no period and no LaTeX spelling are to be set for any coordinate, the argument `coordinates` can be omitted when the shortcut operator `<,>` is used to declare the chart (see examples below).
• **calc_method** – (default: None) string defining the calculus method for computations involving coordinates of the chart; must be one of
  - `'SR'`: Sage's default symbolic engine (Symbolic Ring)
  - `'sympy'`: SymPy
  - `None`: the default of *CalculusMethod* will be used

• **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)

• **coord_restrictions**: Additional restrictions on the coordinates. A restriction can be any symbolic equality or inequality involving the coordinates, such as \( x > y \) or \( x^2 + y^2 
eq 0 \). The items of the list (or set or frozenset) *coord_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 (or set or frozenset) *coord_restrictions*. For example:

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

means \( (x > y) \) and \( ((x != 0) \text{ or } (y != 0)) \) and \( (z^2 < x) \). If the list `coord_restrictions` contains only one item, this item can be passed as such, i.e. writing \( x > y \) instead of the single element list \( [x > y] \). If the chart variables have not been declared as variables yet, *coord_restrictions* must be lambda-quoted.

**EXAMPLES:**

**Cartesian coordinates on \( \mathbb{R}^3 \):**

```python
sage: M = Manifold(3, 'R^3', r'\mathbb{R}^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:

```python
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()*):

```python
sage: M = Manifold(3, 'R^3', r'\mathbb{R}^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:

```python
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 \(\mathbb{R}^3\) 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(rr:(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)
<class '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: (-\infty, +\infty); y: (-\infty, +\infty); z: (-\infty, +\infty)
sage: c_spher.coord_range()
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)
```

A coordinate can be declared periodic by adding the keyword `periodic` to its range:

```
sage: V = M.open_subset('V')
sage: c_spher1.<r,th,ph1> = 
    ....: V.chart(r'(r:(0,+oo) th:(0,pi): \theta ph1:(0,2*pi):periodic: \phi_1')
sage: c_spher1.periods()
    (None, None, 2*pi)
sage: c_spher1.coord_range()
    r: (0, +oo); th: (0, pi); ph1: [0, 2*pi] (periodic)
```

It is equivalent to give the period as `period=2*pi`, skipping the coordinate range:

```
sage: c_spher2.<r,th,ph2> = 
    ....: V.chart(r'(r:(0,+oo) th:(0,pi): \theta ph2:period=2*pi: \phi_2')
sage: c_spher2.periods()
    (None, None, 2*pi)
sage: c_spher2.coord_range()
    r: (0, +oo); th: (0, pi); ph2: [0, 2*pi] (periodic)
```

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)),
     Chart (V, (r, th, ph1)), Chart (V, (r, th, ph2))]
```

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

---

1.5. Coordinate Charts
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(coord_restrictions=lambda x,y,z: (y != 0, x < 0))
sage: c_cartU.valid_coordinates(-1,0,2)
True
sage: c_cartU.valid_coordinates(1,0,2)
False
```

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

```
c_cartU.<x,y,z> = U.chart(coord_restrictions=lambda x,y,z: [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.

This is deprecated; provide the restrictions at the time of creating the chart.

**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 \((x \neq 0)\) or \((y \neq 0)\) and \((z^2 < x)\). If the list \(\text{restrictions}\) 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 \(\mathbb{R}^2\):

```python
sage: M = Manifold(2, 'M', structure='topological') # the open unit disc
sage: X.<x,y> = M.chart()  
```

```python
doctest:warning...
DeprecationWarning: Chart.add_restrictions is deprecated; provide the
restrictions at the time of creating the chart
See https://github.com/sagemath/sage/issues/32102 for details.
```

```python
sage: X.valid_coordinates(0,2)  
False
sage: X.valid_coordinates(0,1/3)  
True
```

The restrictions are transmitted to subcharts:

```python
sage: A = M.open_subset('A') # annulus 1/2 < r < 1
sage: X_A = X.restrict(A, x^2+y^2 > 1/4)
```

```python
sage: X_A._restrictions  
[x^2 + y^2 < 1, x^2 + y^2 > (1/4)]
```

```python
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:

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

```python
sage: X_U.coord_range()  
x: (-oo, +oo); y: (-oo, +oo)
```

```python
sage: X_U.add_restrictions([x<0, y >1/2])
```

```python
sage: X_U.coord_range()  
x: (-oo, 0); y: (1/2, +oo)
```

**codomain()**

Return the codomain of \(\text{self}\) as a set.

**EXAMPLES:**

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

```python
sage: U = M.open_subset('U') # the complement of the half line \{y=0, x >= 0\}
```

```python
sage: _spher.<r,phi> = U.chart(r'r:(0,+oo) phi:(0,2*pi):\phi')
```

```python
sage: _spher.codomain()  
The Cartesian product of ((0, +oo), (0, 2*pi))
```

```python
sage: M = Manifold(3, 'R^3', r'\RR^3', structure='topological', start_index=1)
```

```python
sage: _cart.<x,y,z> = M.chart()
```

```python
sage: _cart.codomain()  
Vector space of dimension 3 over Real Field with 53 bits of precision
```

In the current implementation, the codomain of periodic coordinates are represented by a fundamental do-

1.5. Coordinate Charts
Manifolds, Release 10.3

sage: V = M.open_subset('V')
sage: c_spher1.<r,th,ph1> = 
.....: V.chart(r'r:(0,+oo) th:(0,pi)\theta ph1:(0,2*pi):periodic:\phi_1')
sage: c_spher1.codomain()
The Cartesian product of ((0, +oo), (0, pi), [0, 2*pi))

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
  – xmin 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()  # all 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:

```python
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: XU = X.restrict(U)  # restriction of chart X to U
  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:

```python
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, color='red', style='-', thickness=1, plot_points=75, label_axes=True, **kwds)*

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 value 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(); g  # equivalent to c_cart.plot(c_cart)  # needs sage.plot
Graphics object consisting of 18 graphics primitives
```

Grid of polar coordinates in terms of Cartesian coordinates in the Euclidean plane:
1.5. Coordinate Charts
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(rr:(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); g  # needs sage.plot
Graphics object consisting of 18 graphics primitives

Call with non-default values:

```python
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:

```python
sage: g = c_pol.plot(c_cart, fixed_coords={r: 2})
```
1.5. Coordinate Charts
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
U = S2.open_subset('U'); V = S2.open_subset('V')  # complement of the North and South pole, respectively
S2.declare_union(U,V)

c_xy.<x,y> = U.chart()  # stereographic coordinates from the North pole
c_uv.<u,v> = V.chart()  # stereographic coordinates from the South pole
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)
uv_to_xy = xy_to_uv.inverse()

R3 = Manifold(3, 'R^3', structure='topological')  # the Euclidean space

c_cart.<X,Y,Z> = R3.chart()  # Cartesian coordinates on R^3
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)]},
```

(continues on next page)
### Manifolds, Release 10.3

(continued from previous page)

```python
....:

```sage`
\rightarrow S^2 in \mathbb{R}^3
```

```python
sage:
g = c_xy.plot(c_cart, mapping = Phi); g
```

*needs sage.plot* 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:

```python
sage:
g = c_xy.plot(c_cart, mapping=Phi, label_axes=False)
```

*needs sage.plot* Graphics3d Object

The North and South stereographic charts on the same plot:

```python
sage:
g2 = c_uv.plot(c_cart, mapping=Phi, color='green')
```

*needs sage.plot* Graphics3d Object

```python
sage:
g + g2
```

*needs sage.plot* Graphics3d Object

South stereographic chart drawn in terms of the North one (we split the plot in four parts to avoid the singularity at \((u,v) = (0,0)\)):
1.5. Coordinate Charts
The coordinate line $u = 1$ (red) and the coordinate line $v = 1$ (green) on the same plot:

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:
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: # needs sage.plot
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: # needs sage.plot
sage: g = X.plot(ambient_coords=(t,y))  # the coordinates x and z are not depicted
sage: g
-># long time
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:
Chapter 1. Topological Manifolds
so that they can be adjusted by the user:

```python
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:

```python
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 \((U, \phi)\), a restriction (or subchart) is a chart \((V, \psi)\) such that \(V \subset U\) and \(\psi = \phi|_V\). 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:

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

means \((x > y)\) and \((x \neq 0)\) 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 \(\mathbb{R}^2\) as a subchart of the global Cartesian coordinates:

```python
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

(continues on next page)
valid_coordinates_numerical(*coordinates)

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

This version is optimized for float numbers, and cannot accept parameters nor tolerance. The chart restriction must also be specified in CNF (i.e. a list of tuples).

**INPUT:**
- *coordinates* – coordinate values

**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_numerical(0,1)
True
sage: X.valid_coordinates_numerical(-3/2,5/4)
True
sage: X.valid_coordinates_numerical(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_numerical(0,1)
False
sage: XD.valid_coordinates_numerical(-3/2,5/4)
False
sage: XD.valid_coordinates_numerical(-1/2,1/2)
True
sage: XD.valid_coordinates_numerical(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),
.....: (x^2+y^2<1, abs(y)<1)]})
```

```
sage: XB = X.restrict(B)
sage: XB.valid_coordinates_numerical(-1/2, 0)
True
sage: XB.valid_coordinates_numerical(-1/2, 3/2)
False
sage: XB.valid_coordinates_numerical(3/2, 1/2)
True
```
1.5.2 Chart Functions

In the context of a topological manifold $M$ over a topological field $K$, a *chart function* is a function from a chart codomain to $K$. In other words, a chart function is a $K$-valued function of the coordinates associated to some chart. The internal coordinate expressions of chart functions and calculus on them are taken in charge by different calculus methods, at the choice of the user:

- Sage’s default symbolic engine (Pynac + Maxima), implemented via the Symbolic Ring (SR)
- SymPy engine, denoted sympy hereafter

See *CalculusMethod* for details.

AUTHORS:
- Marco Mancini (2017): initial version
- Eric Gourgoulhon (2015): for a previous class implementing only SR calculus (CoordFunctionSymb)
- Florentin Jaffredo (2018): series expansion with respect to a given parameter

```python
class sage.manifolds.chart_func.ChartFunction (parent, expression=None, calc_method=None, expansion_symbol=None, order=None)
```

Bases: `AlgebraElement`, `ModuleElementWithMutability`

Function of coordinates of a given chart.

If $(U, \varphi)$ is a chart on a topological manifold $M$ of dimension $n$ over a topological field $K$, a *chart function* associated to $(U, \varphi)$ is a map

$$f : \quad V \subset K^n \quad \mapsto \quad K$$

$$\quad (x^1, \ldots, x^n) \quad \mapsto \quad f(x^1, \ldots, x^n),$$

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 chart function $f$ can be represented by expressions pertaining to different calculus methods; the currently implemented ones are

- SR (Sage’s Symbolic Ring)
- SymPy

See *expr()* for details.

INPUT:

- `parent` – the algebra of chart functions on the chart $(U, \varphi)$
- `expression` – (default: None) a symbolic expression representing $f(x^1, \ldots, x^n)$, where $(x^1, \ldots, x^n)$ are the coordinates of the chart $(U, \varphi)$
- `calc_method` – string (default: None): the calculus method with respect to which the internal expression of self must be initialized from expression; one of
  - 'SR': Sage’s default symbolic engine (Symbolic Ring)
  - 'sympy': SymPy
  - None: the chart current calculus method is assumed
- `expansion_symbol` – (default: None) symbolic variable (the “small parameter”) with respect to which the coordinate expression is expanded in power series (around the zero value of this variable)
- `order` - integer (default: None); the order of the expansion if `expansion_symbol` is not None; the order is defined as the degree of the polynomial representing the truncated power series in `expansion_symbol`.

**Warning:** The value of `order` is $n - 1$, where $n$ is the order of the big $O$ in the power series expansion.

**EXAMPLES:**

A symbolic chart function on a 2-dimensional manifold:

```sage
M = Manifold(2, 'M', structure='topological')
x, y = M.chart()
f = x^2 + 3*y + 1
f.display()
```

The symbolic expression is returned when asking for the direct display of the function:

```sage
f
```

```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 expression returned by `expr()` is by default a Sage symbolic expression:

```sage
type(f.expr())
<class 'sage.symbolic.expression.Expression'>
```

A SymPy expression can also be asked for:

```sage
f.expr('sympy')
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)
```

An unspecified chart function:
Coordinate functions can be compared to other values:

```python
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
```

A coercion by means of the restriction is implemented:

```python
sage: D = M.open_subset('D')
sage: X_D = X.restrict(D, x^2+y^2<1)  # open disk
sage: c = X_D.function(x^2)
sage: c + f
2*x^2 + 3*y + 1
```

Expansion to a given order with respect to a small parameter:

```python
sage: t = var('t')  # the small parameter
sage: f = X.function(cos(t)*x*y, expansion_symbol=t, order=2)
```

The expansion is triggered by the call to `simplify()`:

```python
sage: f
x*y*cos(t)
sage: f.simplify()
-1/2*t^2*x*y + x*y
```

**Differences between ChartFunction and callable symbolic expressions**

Callable symbolic expressions are defined directly from symbolic expressions of the coordinates:

```python
sage: f0(x,y) = x^2 + 3*y + 1
sage: type(f0)
<class '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 \( f_0 \) for a chart function, we must use the method \texttt{display()}:

\begin{verbatim}
\texttt{sage: f = X.function(x^2+3*y+1)}
\texttt{sage: f}
x^2 + 3*y + 1
\texttt{sage: f.display()}
(x, y) \mapsto x^2 + 3*y + 1
\texttt{sage: f(x,y)}
x^2 + 3*y + 1
\end{verbatim}

More importantly, instances of \texttt{ChartFunction} differ from callable symbolic expression by the automatic simplifications in all operations. For instance, adding the two callable symbolic expressions:

\begin{verbatim}
\texttt{sage: f0(x,y,z) = cos(x)^2 ; g0(x,y,z) = sin(x)^2}
\end{verbatim}

results in:

\begin{verbatim}
\texttt{sage: f0 + g0}
(x, y, z) \mapsto \cos(x)^2 + \sin(x)^2
\end{verbatim}

To get 1, one has to call \texttt{simplify_trig()}:

\begin{verbatim}
\texttt{sage: (f0 + g0).simplify_trig()}
(x, y, z) \mapsto 1
\end{verbatim}

On the contrary, the sum of the corresponding \texttt{ChartFunction} instances is automatically simplified (see \texttt{simplify_chain_real()} and \texttt{simplify_chain_generic()} for details):

\begin{verbatim}
\texttt{sage: f = X.function(cos(x)^2) \ ; g = X.function(sin(x)^2)}
\texttt{sage: f + g}
1
\end{verbatim}

Another difference regards the display of partial derivatives: for callable symbolic functions, it involves \texttt{diff}:

\begin{verbatim}
\texttt{sage: g = function('g')(x, y)}
\texttt{sage: f0(x,y) = diff(g, x) + diff(g, y)}
\texttt{sage: f0}
(x, y) \mapsto \frac{\partial}{\partial x}g(x, y) + \frac{\partial}{\partial y}g(x, y)
\end{verbatim}

while for chart functions, the display is more “textbook” like:

\begin{verbatim}
\texttt{sage: f = X.function(diff(g, x) + diff(g, y))}
\texttt{sage: f}
d(g)/dx + d(g)/dy
\end{verbatim}

The difference is even more dramatic on LATEX outputs:

\begin{verbatim}
\texttt{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)
\texttt{sage: latex(f)}
\frac{\partial}{\partial x}g + \frac{\partial}{\partial y}g
\end{verbatim}

Note that this regards only the display of coordinate functions: internally, the \texttt{diff} notation is still used, as we can check by asking for the symbolic expression stored in \( f \):

\begin{verbatim}
\texttt{sage: f.expr()}
diff(g(x, y), x) + diff(g(x, y), y)
\end{verbatim}
One can switch to Pynac notation by changing the options:

```python
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)
```

Another difference between `ChartFunction` and callable symbolic expression is the possibility to switch off the display of the arguments of unspecified functions. Consider for instance:

```python
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:

```python
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:

```python
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:

```python
sage: Manifold.options._reset()
sage: f
u(x, y)*v(x, y)

__call__(*coords, **options)

Compute the value of the function at specified coordinates.

INPUT:

- *coords – list of coordinates \((x^1, \ldots, x^n)\), where the function \(f\) is to be evaluated
- **options – allows to pass simplify=False to disable the call of the simplification chain on the result

OUTPUT:

- the value \(f(x^1, \ldots, x^n)\), where \(f\) is the current chart function

EXAMPLES:

```python
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
```
sage: f = X.function(sin(x*y))
sage: f._call_(-2, 3)
-sin(6)
sage: f(-2, 3)
-sin(6)
sage: var('a b')
(a, b)
sage: f._call_(a, b)
sin(a*b)
sage: f(a, b)
sin(a*b)
sage: f._call_(pi, 1)
0
sage: f._call_(pi, 1/2)
1

With SymPy:

sage: X.calculus_method().set('sympy')
sage: f(-2, 3)
-sin(6)
sage: type(f(-2, 3))
<class 'sympy.core.mul.Mul'>
sage: f(a, b)
sin(a*b)
sage: type(f(a, b))
sin
sage: type(f(pi, 1))
<class 'sympy.core.numbers.Zero'>
sage: f(pi, 1/2)
1
sage: type(f(pi, 1/2))
<class 'sympy.core.numbers.One'>

arccos()

Arc cosine of self.

OUTPUT:

• chart function arccos(f), where f is the current chart 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

The same test with SymPy:
Manifolds, Release 10.3

sage: M.set_calculus_method('sympy')
sage: f = X.function(x*y)
sage: f.arccos()
acos(x*y)
sage: arccos(f)   # equivalent to f.arccos()
acos(x*y)
sage: acos(f)    # equivalent to f.arccos()
acos(x*y)
sage: arccos(f).display()
(x, y) ↦ acos(x*y)

arccosh()  
Inverse hyperbolic cosine of self.

OUTPUT:
  • chart function arccosh(f), where f is the current chart 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: acosh(f)    # equivalent to f.arccosh()
arccosh(x*y)
sage: acosh(f)    # equivalent to f.arccosh()
arccosh(x*y)
sage: acosh(f).display()
(x, y) ↦ acosh(x*y)
sage: acosh(X.function(1)) == X.zero_function()
True
```

The same tests with SymPy:

```
sage: X.calculus_method().set('sympy')
sage: f.arccosh()
acosh(x*y)
sage: acosh(f)    # equivalent to f.arccosh()
acosh(x*y)
sage: acosh(f)    # equivalent to f.arccosh()
acosh(x*y)
```

arcsin()  
Arc sine of self.

OUTPUT:
  • chart function arcsin(f), where f is the current chart 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)
```

(continues on next page)
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

The same tests with SymPy:

sage: X.calculus_method().set('sympy')
sage: f.arcsin()
asin(x*y)
sage: arcsin(f)  # equivalent to f.arcsin()
asin(x*y)
sage: asin(f)  # equivalent to f.arcsin()
asin(x*y)
arcsinh()
Inverse hyperbolic sine of self.
OUTPUT:

• chart function arcsinh(f), where f is the current chart 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: arcsinh(f).display()
(x, y) ↦ arcsinh(x*y)
sage: arcsinh(X.zero_function()) == X.zero_function()
True

The same tests with SymPy:

sage: X.calculus_method().set('sympy')
sage: f.arcsinh()
asinh(x*y)
sage: arcsinh(f)  # equivalent to f.arcsinh()
asinh(x*y)
sage: asinh(f)  # equivalent to f.arcsinh()
asinh(x*y)
arctan()
Arc tangent of self.
OUTPUT:

• chart function arctan(f), where f is the current chart 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

The same tests with SymPy:

sage: X.calculus_method().set('sympy')
sage: f.arctan()
atan(x*y)
sage: arctan(f)  # equivalent to f.arctan()
atan(x*y)
sage: atanh(f)  # equivalent to f.arctanh()
atanh(x*y)

arctanh()

Inverse hyperbolic tangent of self.

OUTPUT:

• chart function arctanh(f), where f is the current chart 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()
atanh(x*y)
sage: arctanh(f).display()
(x, y) ↦ arctanh(x*y)
sage: arctanh(X.zero_function()) == X.zero_function()
True

The same tests with SymPy:

sage: X.calculus_method().set('sympy')
sage: f.arctanh()
atanh(x*y)
sage: arctanh(f)  # equivalent to f.arctanh()
atanh(x*y)

chart()

Return the chart with respect to which self is defined.
OUTPUT:

• a Chart

EXAMPLES:

```python
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
```

collect\( (s) \)

Collect the coefficients of \( s \) in the expression of \( \text{self} \) into a group.

INPUT:

• \( s \) – the symbol whose coefficients will be collected

OUTPUT:

• \( \text{self} \) with the coefficients of \( s \) grouped in its expression

EXAMPLES:

Action on a 2-dimensional chart function:

```python
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) \mapsto 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 \( f \):

```python
sage: f.display()
(x, y) \mapsto x^2*y^2 + (x^2 + x)*y
```

The same test with SymPy

```python
sage: X.calculus_method().set('sympy')
sage: f = X.function(x^2*y + x*y + (x*y)^2)
sage: f.display()
(x, y) \mapsto x**2*y**2 + x**2*y + x*y
sage: f.collect(y)
x**2*y**2 + y*(x**2 + x)
```

collect_common_factors\( () \)

Collect common factors in the expression of \( \text{self} \).

This method does not perform a full factorization but only looks for factors which are already explicitly present.

OUTPUT:

• \( \text{self} \) with the common factors collected in its expression
**EXAMPLES:**

**Action on a 2-dimensional chart 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)
```

The same test with SymPy:

```
sage: X.calculus_method().set('sympy')
sage: g = X.function(x/(x^2*y + x*y))
sage: g.display()
(x, y) ↦ x/(x^2*y + x*y)
sage: g.collect_common_factors()
1/(y*(x + 1))
```

**copy()**

Return an exact copy of the object.

**OUTPUT:**

- a `ChartFunctionSymb`

**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:**

- chart function \( \cos(f) \), where \( f \) is the current chart function
EXAMPLES:

```python
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
```

The same tests with SymPy:

```python
sage: X.calculus_method().set('sympy')
sage: f.cos()
\cos(x*y)
sage: cos(f)  # equivalent to f.cos()
\cos(x*y)
```

cosh()

Hyperbolic cosine of self.

OUTPUT:

- chart function $\cosh(f)$, where $f$ is the current chart function

EXAMPLES:

```python
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
```

The same tests with SymPy:

```python
sage: X.calculus_method().set('sympy')
sage: f.cosh()
\cosh(x*y)
sage: cosh(f)  # equivalent to f.cosh()
\cosh(x*y)
```

derivative(coord)

Partial derivative with respect to a coordinate.

INPUT:

- coord – either the coordinate $x^i$ with respect to which the derivative of the chart 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 ChartFunction representing the partial derivative \( \frac{\partial f}{\partial x_i} \)

EXAMPLES:

Partial derivatives of a 2-dimensional chart function:

```sage
M = Manifold(2, 'M', structure='topological')
X.<x,y> = M.chart(calc_method='SR')
f = X.function(x^2+3*y+1); f
x^2 + 3*y + 1
f.derivative(x)
2*x
f.derivative(y)
3
```

An alias is diff:

```sage
f.diff(x)
2*x
```

Each partial derivative is itself a chart function:

```sage
f.diff(x)
2*x
type(f.diff(x))
<class 'sage.manifolds.chart_func.ChartFunctionRing_with_category.element_class'>
```

The same result is returned by the function diff:

```sage
diff(f, x)
2*x
```

An index can be used instead of the coordinate symbol:

```sage
f.diff(0)
2*x
diff(f, 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)
X.<x,y> = M.chart()
f = X.function(x^2+3*y+1)
f.diff(0)
Traceback (most recent call last):
...
ValueError: coordinate index out of range
f.diff(1)
2*x
f.diff(2)
3
```

The same test with SymPy:

```sage
M = Manifold(2, 'M', structure='topological')
X.<x,y> = M.chart(calc_method='sympy')
f = X.function(x^2+3*y+1); f
(continues on next page)```
**diff** (coord)

Partial derivative with respect to a coordinate.

**INPUT:**

- `coord` – either the coordinate $x^i$ with respect to which the derivative of the chart 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 `ChartFunction` representing the partial derivative $\frac{\partial f}{\partial x^i}$

**EXAMPLES:**

Partial derivatives of a 2-dimensional chart function:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart(calc_method='SR')
sage: f = X.function(x^2+3*y+1); f
x^2 + 3*y + 1
sage: f.derivative(x)
2*x
sage: f.derivative(y)
3
```

An alias is `diff`:

```
sage: f.diff(x)
2*x
```

Each partial derivative is itself a chart function:

```
sage: type(f.diff(x))
<class 'sage.manifolds.chart_func.ChartFunctionRing_with_category.element_class'>
```

The same result is returned by the function `diff`:

```
sage: diff(f, x)
2*x
```

An index can be used instead of the coordinate symbol:

```
sage: f.diff(0)
2*x
sage: diff(f, 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
```

The same test with SymPy:
```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart(calc_method='sympy')
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
```

**disp()**

Display self in arrow notation. For display the standard SR representation is used.

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. For display the standard SR representation is used.

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

**EXAMPLES:**

Coordinate function on a 2-dimensional manifold:
Manifolds, Release 10.3

```python
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()`:

```python
sage: f.disp()
(x, y) ↦ cos(1/2*x*y)
```

Display of the zero function:

```python
sage: X.zero_function().display()
(x, y) ↦ 0
```

**exp()**

Exponential of `self`.

**OUTPUT:**

- chart function `exp(f)`, where `f` is the current chart function

**EXAMPLES:**

```python
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
```

The same test with SymPy:

```python
sage: X.calculus_method().set('sympy')
sage: f = X.function(x+y)
sage: f.exp()
exp(x + y)
sage: exp(f)  # equivalent to f.exp()
exp(x + y)
sage: exp(f).display()
(x, y) ↦ exp(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 chart 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
```

The same test with SymPy

```
sage: X.calculus_method().set('sympy')
sage: g = X.function((x - y)^2)
sage: g.expand()
x**2 - 2*x*y + y**2
```

```
expr (method=None)
```

Return the symbolic expression of `self` in terms of the chart coordinates, as an object of a specified calculus method.

INPUT:

- `method` -- string (default: None): the calculus method which the returned expression belongs to; one of
  - 'SR': Sage's default symbolic engine (Symbolic Ring)
  - 'sympy': SymPy
  - None: the chart current calculus method is assumed

OUTPUT:

- a Sage symbolic expression if `method` is 'SR'
- a SymPy object if `method` is 'sympy'

EXAMPLES:

Chart function on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x^2+y)
sage: f.expr()
x^2 + y
sage: type(f.expr())
<class 'sage.symbolic.expression.Expression'>
```

Asking for the SymPy expression:

```
sage: f.expr('sympy')
x**2 + y
```
The default corresponds to the current calculus method, here the one based on the Symbolic Ring SR:

\[
\text{sage: } \text{f.expr('sympy')} \text{ is f.expr('SR')}
\]

\[
\text{True}
\]

If we change the current calculus method on chart \(X\), we change the default:

\[
\text{sage: } X.\text{calculus_method().set('sympy')}
\text{sage: } \text{f.expr()}
\]

\[
x^2 + y
\]

\[
\text{sage: } \text{f.expr().subs(a=2)}
\]

\[
2x*y
\]

Internally, the expressions corresponding to various calculus methods are stored in the dictionary \_express:

\[
\text{sage: for method in sorted(f._express):}
\text{...: print(""{}: {}"".format(method, f._express[method])))}
\text{...:}
\]

\[
'SR': x^2 + y
'sympy': x^2 + y
\]

The method \text{expr()}\ is useful for accessing to all the symbolic expression functionalities in Sage; for instance:

\[
\text{sage: var('a')}
\]

\[
a
\]

\[
\text{sage: } \text{f = X.function(a*x*y); f.display()}
\]

\[
(x, y) \mapsto axy
\]

\[
\text{sage: } \text{f.expr()}
\]

\[
axy
\]

\[
\text{sage: } \text{f.expr().subs(a=2)}
\]

\[
2x*y
\]

Note that for substituting the value of a coordinate, the function call can be used as well:

\[
\text{sage: f(x,3)}
\]

\[
3ax
\]

\[
\text{sage: bool( f(x,3) == f.expr().subs(y=3) )}
\]

\[
\text{True}
\]

**factor()**

Factorize the coordinate expression of \text{self}.

\[
\text{OUTPUT:}
\]

\[
\bullet \text{self with its expression factorized}
\]

**EXAMPLES:**

Factorization of a 2-dimensional chart function:

\[
\text{sage: } M = \text{Manifold(2, 'M', structure='topological')}
\text{sage: } X.<x,y> = M.chart()
\text{sage: } f = X.function(x^2 + 2x*y + y^2)
\]
The method `factor()` has changed the expression of `f`:

```
sage: f.display()
(x, y) ↦ (x + y)^2
```

The same test with SymPy:

```
sage: X.calculus_method().set('sympy')
sage: g = X.function(x^2 + 2*x*y + y^2)
sage: g.display()
(x, y) ↦ x**2 + 2*x*y + y**2
sage: g.factor()
(x + y)**2
```

**is_trivial_one()**

Check if `self` is trivially equal to one without any simplification.

This method is supposed to be fast as compared with `self == 1` and is intended to be used in library code where trying to obtain a mathematically correct result by applying potentially expensive rewrite rules is not desirable.

**EXAMPLES:**

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(1)
sage: f.is_trivial_one()
True
sage: f = X.function(float(1.0))
sage: f.is_trivial_one()
True
sage: f = X.function(x-x+1)
sage: f.is_trivial_one()
True
sage: X.one_function().is_trivial_one()
True
```

No simplification is attempted, so that `False` is returned for non-trivial cases:

```
sage: f = X.function(cos(x)^2 + sin(x)^2)
sage: f.is_trivial_one()
False
```

On the contrary, the method `is_zero()` and the direct comparison to one involve some simplification algorithms and return `True`:

```
sage: (f - 1).is_zero()
True
sage: f == 1
True
```
**is_trivial_zero()**

Check if `self` is trivially equal to zero without any simplification.

This method is supposed to be fast as compared with `self.is_zero()` or `self == 0` and is intended to be used in library code where trying to obtain a mathematically correct result by applying potentially expensive rewrite rules is not desirable.

**EXAMPLES:**

```python
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(0)
sage: f.is_trivial_zero()
True
sage: f = X.function(float(0.0))
sage: f.is_trivial_zero()
True
sage: f = X.function(x-x)
sage: f.is_trivial_zero()
True
sage: X.zero_function().is_trivial_zero()
True
```

No simplification is attempted, so that False is returned for non-trivial cases:

```python
sage: f = X.function(cos(x)^2 + sin(x)^2 - 1)
sage: f.is_trivial_zero()
False
```

On the contrary, the method `is_zero()` and the direct comparison to zero involve some simplification algorithms and return True:

```python
sage: f.is_zero()
True
sage: f == 0
True
```

**is_unit()**

Return True iff `self` is not trivially zero since most chart functions are invertible and an actual computation would take too much time.

**EXAMPLES:**

```python
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x^2+3*y+1)
sage: f.is_unit()
True
sage: zero = X.function(0)
sage: zero.is_unit()
False
```

**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:

- chart function $\log_a(f)$, where $f$ is the current chart 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)
```

The same test with SymPy:

```
sage: X.calculus_method().set('sympy')
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)
```

`ScalarField(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()
```
\( M \rightarrow \mathbb{R} \) 
\((x, y) \mapsto 2y^3 + x\)

\[
\text{sage: } f.\text{coord\_function(c\_xy)} = \text{fc}
\]

\[
\text{True}
\]

**\text{set\_expr}(\text{calc\_method, expression})**

Add an expression in a particular calculus method \text{self}. Some control is done to verify the consistency between the different representations of the same expression.

**INPUT:**

- \text{calc\_method} – calculus method
- \text{expression} – symbolic expression

**EXAMPLES:**

\[
\text{sage: } M = \text{Manifold}(2, 'M', \text{structure}='\text{topological}')
\]
\[
\text{sage: } X.<x,y> = M.\text{chart}()
\]
\[
\text{sage: } f = X.\text{function}(1+x^2)
\]
\[
\text{sage: } f.\text{display}()
\]
\[
(x, y) \mapsto x^2 + 1
\]

\[
\text{sage: } f.\text{set\_expr('sympy','x^2+1')}
\]

\[
\text{sage: } f
\]

# indirect doctest

\[
x^2 + 1
\]

\[
\text{sage: } g = X.\text{function}(1+x^3)
\]
\[
\text{sage: } g.\text{display}()
\]
\[
(x, y) \mapsto x^3 + 1
\]
\[
\text{sage: } g.\text{set\_expr('sympy','x^2+y')}
\]

Traceback (most recent call last):
...

ValueError: Expressions are not equal

**\text{simplify}()**

Simplify the coordinate expression of \text{self}.

For details about the employed chain of simplifications for the SR calculus method, see \text{simplify\_chain\_real()} for chart functions on real manifolds and \text{simplify\_chain\_generic()} for the generic case.

If \text{self} has been defined with the small parameter \text{expansion\_symbol} and some truncation order, the coordinate expression of \text{self} will be expanded in power series of that parameter and truncated to the given order.

**OUTPUT:**

- \text{self} with its coordinate expression simplified

**EXAMPLES:**

Simplification of a chart function on a 2-dimensional manifold:

\[
\text{sage: } M = \text{Manifold}(2, 'M', \text{structure}='\text{topological}')
\]
\[
\text{sage: } X.<x,y> = M.\text{chart}()
\]
\[
\text{sage: } f = X.\text{function}(\cos(x)^2 + \sin(x)^2 + \sqrt{x^2})
\]
\[
\text{sage: } f.\text{display}()
\]
\[
(x, y) \mapsto \cos(x)^2 + \sin(x)^2 + \abs{x}
\]
\[
\text{sage: } f.\text{simplify}()
\]
\[
\abs{x} + 1
\]
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)); 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', structure='topological')
sage: X.<x,y> = M.chart(x:(1,+oo) y)
sage: f = X.function(sqrt(x^2-2*x+1)); f
sqrt(x^2 - 2*x + 1)
sage: f.simplify()
x - 1
```

```
sage: forget()  # to clear the previous assumption on x
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart(x:(-oo,0) y)
sage: f = X.function(sqrt(x^2-2*x+1)); f
sqrt(x^2 - 2*x + 1)
sage: f.simplify()
-x + 1
```

The same tests with SymPy:

```
sage: forget()  # to clear the previous assumption on x
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart(calc_method='sympy')
sage: f = X.function(cos(x)^2 + sin(x)^2 + sqrt(x^2)); f
sin(x)**2 + cos(x)**2 + Abs(x)
sage: f.simplify()
Abs(x) + 1
```

```
sage: f = X.function((x^2-1)/(x+1)); f
(x**2 - 1)/(x + 1)
sage: f.simplify()
x - 1
```

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart(x:(1,+oo) y, calc_method='sympy')
sage: f = X.function(sqrt(x^2-2*x+1)); f
sqrt(x^2 - 2*x + 1)
sage: f.simplify()
x - 1
```

```
sage: forget()  # to clear the previous assumption on x
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart(x:(-oo,0) y, calc_method='sympy')
sage: f = X.function(sqrt(x^2-2*x+1)); f
sqrt(x^2 - 2*x + 1)
```

(continues on next page)
Power series expansion with respect to a small parameter $t$ (at the moment, this is implemented only for the SR calculus backend, hence the first line below):

```
sage: X.calculus_method().set('SR')
sage: t = var('t')
sage: f = X.function(exp(t*x), expansion_symbol=t, order=3)
```

At this stage, $f$ is not expanded in power series:

```
sage: f
e^t*x
```

Invoking `simplify()` triggers the expansion to the given order:

```
sage: f.simplify()
1/6*t^3*x^3 + 1/2*t^2*x^2 + t*x + 1
sage: f.display()
(x, y) ↦ 1/6*t^3*x^3 + 1/2*t^2*x^2 + t*x + 1
```

### `sin()`

Sine of `self`.

**OUTPUT:**

- chart function $\sin(f)$, where $f$ is the current chart 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
```

The same tests with SymPy:

```
sage: X.calculus_method().set('sympy')
sage: f = X.function(x*y)
sage: f.sin()
sin(x*y)
sage: sin(f)  # equivalent to f.sin()
sin(x*y)
```

---

1.5. Coordinate Charts 153
**sinh()**

Hyperbolic sine of `self`.

**OUTPUT:**

• chart function $\sinh(f)$, where $f$ is the current chart 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
```

The same tests with SymPy:

```sage
X.calculus_method().set('sympy')
sage: f.sinh()
sinh(x+y)
sage: sinh(f)  # equivalent to f.sinh()
sinh(x+y)
```

**sqrt()**

Square root of `self`.

**OUTPUT:**

• chart function $\sqrt{f}$, where $f$ is the current chart 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:**

• chart function $\tan(f)$, where $f$ is the current chart function

**EXAMPLES:**
```python
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

The same test with SymPy:
```}

```python
sage: M.set_calculus_method('sympy')
sage: g = X.function(x*y)
sage: g.tan()
tan(x*y)
sage: tan(g)  # equivalent to g.tan()
tan(x*y)
sage: tan(g).display()
(x, y) ↦ tan(x*y)
```

tanh()  
Hyperbolic tangent of self.

OUTPUT:

- chart function $\tanh(f)$, where $f$ is the current chart function

EXAMPLES:
```python
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

The same tests with SymPy:
```python
sage: X.calculus_method().set('sympy')
sage: f.tanh()
tanh(x*y)
sage: tanh(f)  # equivalent to f.tanh()
tanh(x*y)
```

```python
class sage.manifolds.chart_func.ChartFunctionRing(chart)
Bases: Parent, UniqueRepresentation
Ring of all chart functions on a chart.
INPUT:
- chart – a coordinate chart, as an instance of class Chart
```

1.5. Coordinate Charts
EXAMPLES:

The ring of all chart functions w.r.t. to a chart:

```sage
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: FR = X.function_ring(); FR
Ring of chart functions on Chart (M, (x, y))
sage: type(FR)
<class 'sage.manifolds.chart_func.ChartFunctionRing_with_category'>
sage: FR.category()
Category of commutative algebras over Symbolic Ring
```

Coercions by means of restrictions are implemented:

```sage
sage: FR_X = X.function_ring()
sage: D = M.open_subset('D')
sage: X_D = X.restrict(D, x^2+y^2<1)  # open disk
sage: FR_X_D = X_D.function_ring()
sage: FR_X_D.has_coerce_map_from(FR_X)
True
```

But only if the charts are compatible:

```sage
sage: Y.<t,z> = D.chart()
sage: FR_Y = Y.function_ring()
sage: FR_Y.has_coerce_map_from(FR_X)
False
```

Element

alias of `ChartFunction`

`is_field (proof=True)`

Return `False` as `self` is not an integral domain.

EXAMPLES:

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

`is_integral_domain (proof=True)`

Return `False` as `self` is not an integral domain.

EXAMPLES:
one()

Return the constant function 1 in self.

EXAMPLES:

```python
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:

```python
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
```

class sage.manifolds.chart_func.MultiCoordFunction(chart, expressions)

Bases: SageObject, Mutability

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

\[
f : \quad V \subset K^n \quad \longrightarrow \quad K^m
\]

\[
(x^1, \ldots, x^n) \quad \mapsto \quad (f_1(x^1, \ldots, x^n), \ldots, f_m(x^1, \ldots, x^n)),
\]

where \( V \) is the codomain of \( \varphi \). In other words, \( f \) is a \( K^m \)-valued function of the coordinates associated to the chart \((U, \varphi)\). Each component \( f_i \) (\( 1 \leq i \leq m \)) is a coordinate function and is therefore stored as a `ChartFunction`.

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 \( f : V \subset \mathbb{R}^2 \longrightarrow \mathbb{R}^3 \):
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 ChartFunction:

```
sage: isinstance(f[0], sage.manifolds.chart_func.ChartFunction)
True
```

A class MultiCoordFunction can represent a real-valued function (case $m = 1$), although one should rather employ the class ChartFunction 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: f(a,b)
(a - b, a*b, cos(a)*e^b)
sage: g(1,2)
(,)
```

chart() Return the chart with respect to which self is defined.

OUTPUT:

- a Chart

EXAMPLES:
Manifolds, Release 10.3

```python
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 (method=None)**

Return a tuple of data, the item no. \( i \) being 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 \( f \).

**INPUT:**

- `method` – string (default: None): the calculus method which the returned expressions belong to; one of
  - 'SR': Sage's default symbolic engine (Symbolic Ring)
  - 'sympy': SymPy
  - None: the chart current calculus method is assumed

**OUTPUT:**

- a tuple of the symbolic expressions of the chart functions composing \( f \)

**EXAMPLES:**

```python
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])
<class 'sage.symbolic.expression.Expression'>
```

**A SymPy output:**

```python
sage: f.expr('sympy')
(x - y, x*y, exp(y)*cos(x))
sage: type(f.expr('sympy')[0])
<class 'sympy.core.add.Add'>
```

One shall always have:

```python
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:

```python
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 chart function:

```python
sage: type(f.jacobian()[2,0])
<class 'sage.manifolds.chart_func.ChartFunctionRing_with_category.element_parent'>
sage: f.jacobian()[2,0].display()
(x, y) ↦ -y^3*sin(x)
```

Test of the computation:

```python
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:

```python
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]]
```

\textbf{jacobian\_det ()}

Return the Jacobian determinant of the system of functions.

The number $m$ of coordinate functions must equal the number $n$ of coordinates.

\textbf{OUTPUT:}

• a ChartFunction representing the determinant

\textbf{EXAMPLES:}

Jacobian determinant of a set of 2 functions of 2 coordinates:

```python
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 \texttt{jacobian\_det()} is an instance of \texttt{ChartFunction} and can therefore be called on specific values of the coordinates, e.g. \((x, y) = (1, 2)\):

\begin{verbatim}
sage: type(f.jacobian_det())
<class 'sage.manifolds.chart_func.ChartFunctionRing_with_category.element_
    class'>
sage: f.jacobian_det().display()
(x, y) ↦ x + y
sage: f.jacobian_det()(1,2)
3
\end{verbatim}

The result is cached:

\begin{verbatim}
sage: f.jacobian_det() is f.jacobian_det()
True
\end{verbatim}

We verify the determinant of the Jacobian:

\begin{verbatim}
sage: f.jacobian_det() == det(matrix([[f[i].diff(j).expr() for j in range(2)]
    ....: for i in range(2)]))
True
\end{verbatim}

An example using SymPy:

\begin{verbatim}
sage: M.set\_calculus\_method('sympy')
sage: g = X.multifunction(x*y^3, e^x)
sage: g.jacobian\_det()
-3*x*y**2*exp(x)
sage: type(g.jacobian\_det().expr())
<class 'sympy.core.mul.Mul'>
\end{verbatim}

Jacobian determinant of a set of 3 functions of 3 coordinates:

\begin{verbatim}
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
\end{verbatim}

We verify the determinant of the Jacobian:

\begin{verbatim}
sage: f.jacobian\_det() == det(matrix([[f[i].diff(j).expr() for j in range(3)]
    ....: for i in range(3)]))
True
\end{verbatim}

\texttt{set\_immutable()}

Set self and all chart functions of self immutable.

EXAMPLES:

Declare a coordinate function immutable:

\begin{verbatim}
sage: M = Manifold(3, 'M', structure='topological')
sage: X.<x,y,z> = M.chart()
sage: f = X.multifunction(x+y+z, x*y*z)
sage: f.is\_immutable()
False
sage: f.set\_immutable()
\end{verbatim}
The chart functions are now immutable, too:

```python
sage: f[0].parent()
Ring of chart functions on Chart (M, (x, y, z))
sage: f[0].is_immutable()
True
```

1.5.3 Coordinate calculus methods

The class `CalculusMethod` governs the calculus methods (symbolic and numerical) used for coordinate computations on manifolds.

**AUTHORS:**

- Marco Mancini (2017): initial version
- Eric Gourgoulhon (2019): add `set_simplify_function()` and various accessors

```python
class sage.manifolds.calculus_method.CalculusMethod(
current=None,
base_field_type='real')
```

Control of calculus backends used on coordinate charts of manifolds.

This class stores the possible calculus methods and permits to switch between them, as well as to change the simplifying functions associated with them. For the moment, only two calculus backends are implemented:

- Sage’s symbolic engine (Pynac + Maxima), implemented via the Symbolic Ring `SR`
- SymPy engine, denoted `sympy` hereafter

**INPUT:**

- `current` – (default: `None`) string defining the calculus method that will be considered as the active one, until it is changed by `set()`. Must be one of
  - `'SR'`: Sage’s default symbolic engine (Symbolic Ring)
  - `'sympy'`: SymPy
  - `None`: the default calculus method ('SR')
- `base_field_type` – (default: `'real'`) base field type of the manifold (cf. `base_field_type()`)

**EXAMPLES:**

```python
sage: from sage.manifolds.calculus_method import CalculusMethod
sage: cm = CalculusMethod()
```

In the display, the currently active method is pointed out with a star:

```python
sage: cm
Available calculus methods (* = current):
  - SR (default) (*)
  - sympy
```

It can be changed with `set()`:
sage: cm.set('sympy')
sage: cm
Available calculus methods (* = current):
- SR (default)
- sympy (*)

while `reset()` brings back to the default:

sage: cm.reset()
sage: cm
Available calculus methods (* = current):
- SR (default) (*)
- sympy

See `simplify_function()` for the default simplification algorithms associated with each calculus method and `set_simplify_function()` for introducing a new simplification algorithm.

current()

Return the active calculus method as a string.

OUTPUT:

- string defining the calculus method, one of
  - 'SR': Sage’s default symbolic engine (Symbolic Ring)
  - 'sympy': SymPy

EXAMPLES:

sage: from sage.manifolds.calculus_method import CalculusMethod
sage: cm = CalculusMethod(); cm
Available calculus methods (* = current):
- SR (default) (*)
- sympy
sage: cm.current()
'SR'
sage: cm.set('sympy')
sage: cm.current()
'sympy'

is_trivial_zero(expression, method=None)

Check if an expression is trivially equal to zero without any simplification.

INPUT:

- expression – expression
- method – (default: None) string defining the calculus method to use; if None the current calculus method of self is used.

OUTPUT:

- True is expression is trivially zero, False elsewhere.

EXAMPLES:

sage: from sage.manifolds.calculus_method import CalculusMethod
sage: cm = CalculusMethod(base_field_type='real')
sage: f = sin(x) - sin(x)
sage: cm.is_trivial_zero(f)
True

```sage``
``cm.is_trivial_zero(f._sympy_(), method='sympy')``
True
```

```sage``
f = sin(x)^2 + cos(x)^2 - 1
```

```sage```
``cm.is_trivial_zero(f)```
False
```

```sage```
``cm.is_trivial_zero(f._sympy_(), method='sympy')``
False
```

``reset()``

Set the current calculus method to the default one.

```
``` EXAMPLES:

```sage```
``from sage.manifolds.calculus_method import CalculusMethod``
````
``sage``
``cm = CalculusMethod(base_field_type='complex')``
````
``sage``
``cm``
Available calculus methods (* = current):
- SR (default) (*)
- sympy
```
``sage``
``cm.set('sympy')``
````
``sage``
``cm``
Available calculus methods (* = current):
- SR (default) (*)
- sympy
```
``sage``
``cm.set('sympy')``
````
``sage``
``cm``
Available calculus methods (* = current):
- SR (default) (*)
- sympy
```
``sage``
``cm.set(lala)``
````
``Traceback (most recent call last):
  ...
NotImplementedError: method lala not implemented```

``set(method)``

Set the currently active calculus method.

- method – string defining the calculus method

```
``` EXAMPLES:

```sage```
``from sage.manifolds.calculus_method import CalculusMethod``
````
``sage``
``cm = CalculusMethod(base_field_type='complex')``
````
``sage``
``cm``
Available calculus methods (* = current):
- SR (default) (*)
- sympy
```
``sage``
``cm.set('sympy')``
````
``sage``
``cm``
Available calculus methods (* = current):
- SR (default) (*)
- sympy
```
``sage``
``cm.set('lala')``
````
``Traceback (most recent call last):
  ...
NotImplementedError: method lala not implemented```

``set_simplify_function(simplifying_func, method=None)``

Set the simplifying function associated to a given calculus method.
INPUT:

- `simplifying_func` – either the string 'default' for restoring the default simplifying function or a function \( f \) of a single argument `expr` such that \( f(expr) \) returns an object of the same type as `expr` (hopefully the simplified version of `expr`), this type being
  - `Expression` if `method = 'SR'`
  - a SymPy type if `method = 'sympy'`
- `method` – (default: `None`) string defining the calculus method for which `simplifying_func` is provided; must be one of
  - 'SR': Sage's default symbolic engine (Symbolic Ring)
  - 'sympy': SymPy
  - `None`: the currently active calculus method of `self` is assumed

EXAMPLES:

On a real manifold, the default simplifying function is `simplify_chain_real()` when the calculus method is `SR`:

```python
sage: from sage.manifolds.calculus_method import CalculusMethod
sage: cm = CalculusMethod(base_field_type='real'); cm
Available calculus methods (* = current):
- SR (default) (*)
- sympy

sage: cm.simplify_function() is \
....: sage.manifolds.utilities.simplify_chain_real
True
```

Let us change it to `simplify()`:

```python
sage: cm.set_simplify_function(simplify)

sage: cm.simplify_function() is simplify
True
```

Since `SR` is the current calculus method, the above is equivalent to:

```python
sage: cm.set_simplify_function(simplify, method='SR')

sage: cm.simplify_function(method='SR') is simplify
True
```

We revert to the default simplifying function by:

```python
sage: cm.set_simplify_function('default')

Then we are back to:

```python
sage: cm.simplify_function() is \
....: sage.manifolds.utilities.simplify_chain_real
True
```

`simplify(expression, method=None)`

Apply the simplifying function associated with a given calculus method to a symbolic expression.

INPUT:

- `expression` – symbolic expression to simplify
• method – (default: None) string defining the calculus method to use; must be one of
  – 'SR': Sage’s default symbolic engine (Symbolic Ring)
  – 'sympy': SymPy
  – None: the current calculus method of self is used.

OUTPUT:
• the simplified version of expression

EXAMPLES:

```sage```
M = Manifold(2, 'M', structure='topological')
X.<x, y> = M.chart()
f = x^2 + sin(x)^2 + cos(x)^2
from sage.manifolds.calculus_method import CalculusMethod
cm = CalculusMethod(base_field_type='real')
cm.simplify(f)
```
x^2 + 1```

Using a weaker simplifying function, like `simplify()`, does not succeed in this case:

```sage```
cm.set_simplify_function(simplify)
cm.simplify(f)
x^2 + cos(x)^2 + sin(x)^2
```

Back to the default simplifying function (`simplify_chain_real()` in the present case):

```sage```
cm.set_simplify_function('default')
cm.simplify(f)
x^2 + 1
```

A SR expression, such as `f`, cannot be simplified when the current calculus method is sympy:

```sage```
cm.set('sympy')
cm.simplify(f)
Traceback (most recent call last):
...
AttributeError: 'sage.symbolic.expression.Expression' object has no attribute 'combsimp'
```

In the present case, one should either transform `f` to a SymPy object:

```sage```
cm.simplify(f._sympy_())
x**2 + 1
```

or force the calculus method to be 'SR':

```sage```
cm.simplify(f, method='SR')
x^2 + 1
```

`simplify_function` (`method=None`)
Return the simplifying function associated to a given calculus method.

The simplifying function is that used in all computations involved with the calculus method.

INPUT:
• method – (default: None) string defining the calculus method for which `simplify_func` is provided; must be one of
• 'SR': Sage's default symbolic engine (Symbolic Ring)
• 'sympy': SymPy
• None: the currently active calculus method of self is assumed

OUTPUT:

• the simplifying function

EXAMPLES:

```python
sage: from sage.manifolds.calculus_method import CalculusMethod
sage: cm = CalculusMethod()
sage: cm
Available calculus methods (* = current):
- SR (default) (*)
- sympy
sage: cm.simplify_function()  # random (memory address)
<function simplify_chain_real at 0x7f958d5b6758>
```

The output stands for the function `simplify_chain_real()`:

```python
sage: cm.simplify_function() is \n....: sage.manifolds.utilities.simplify_chain_real
True
```

Since SR is the default calculus method, we have:

```python
sage: cm.simplify_function() is cm.simplify_function(method='SR')
True
```

The simplifying function associated with sympy is `simplify_chain_real_sympy()`:

```python
sage: cm.simplify_function(method='sympy')  # random (memory address)
<function simplify_chain_real_sympy at 0x7f0b35a578c0>
sage: cm.simplify_function(method='sympy') is  
....: sage.manifolds.utilities.simplify_chain_real_sympy
True
```

On complex manifolds, the simplifying functions are `simplify_chain_generic()` and `simplify_chain_generic_sympy()` for respectively SR and sympy:

```python
sage:cmc = CalculusMethod(base_field_type='complex')
sage:cmc.simplify_function(method='SR') is  
....: sage.manifolds.utilities.simplify_chain_generic
True
sage:cmc.simplify_function(method='sympy') is  
....: sage.manifolds.utilities.simplify_chain_generic_sympy
True
```

Note that the simplifying functions can be customized via `set_simplify_function()`.
1.6 Scalar Fields

1.6.1 Algebra of Scalar Fields

The class \texttt{ScalarFieldAlgebra} implements the commutative algebra \( C^0(M) \) of scalar fields on a topological manifold \( M \) over a topological field \( K \). By \textit{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:
- Travis Scrimshaw (2016): review tweaks

REFERENCES:
- [Lee2011]
- [KN1963]

\begin{class}
\texttt{sage.manifolds.scalarfield_algebra.ScalarFieldAlgebra}(\texttt{domain})
\end{class}

\texttt{Bases: UniqueRepresentation, 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 \( K = \mathbb{R} \) or \( K = \mathbb{C} \), the field \( K \) over which the algebra \( C^0(M) \) is constructed is represented by the \texttt{Symbolic Ring SR}, since there is no exact representation of \( \mathbb{R} \) nor \( \mathbb{C} \).

INPUT:
- \texttt{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:

\begin{verbatim}
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
\end{verbatim}

\( C^0(M) \) and \( C^0(W) \) belong to the category of commutative algebras over \( \mathbb{R} \) (represented here by \texttt{Symbolic Ring SR}).
The elements of $C^0(M)$ are scalar fields on $M$:

```sage
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
$M \to \mathbb{R}$
on $U$: $(x, y) \mapsto 2$
on $V$: $(u, v) \mapsto 2$
```

Those of $C^0(W)$ are scalar fields on $W$:

```sage
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
$W \to \mathbb{R}$
$(x, y) \mapsto 2$
$(u, v) \mapsto 2$
```

The zero element:

```sage
sage: CM.zero()
Scalar field zero on the 2-dimensional topological manifold $M$
sage: CM.zero().display()
zero: $M \to \mathbb{R}$
on $U$: $(x, y) \mapsto 0$
on $V$: $(u, v) \mapsto 0$
```

```sage
sage: CW.zero()
Scalar field zero on the Open subset $W$ of the 2-dimensional
manifold $M$
sage: CW.zero().display()
zero: $W \to \mathbb{R}$
$(x, y) \mapsto 0$
$(u, v) \mapsto 0$
```

The unit element:

```sage
sage: CM.one()
Scalar field 1 on the 2-dimensional topological manifold $M$
sage: CM.one().display()
1: $M \to \mathbb{R}$
on $U$: $(x, y) \mapsto 1$
on $V$: $(u, v) \mapsto 1$
```
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()

$W \rightarrow \mathbb{R}$

$(x, y) \mapsto \arctan(x^2 + y^2)$

$(u, v) \mapsto \frac{1}{2}\pi - \arctan(u^2 + v^2)$
```

```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()

$W \rightarrow \mathbb{R}$

$(x, y) \mapsto 2\arctan(x^2 + y^2)$

$(u, v) \mapsto \pi - 2\arctan(u^2 + v^2)$
```

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 $s\cdot CM\text{.one()}$, 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

```sage```
h.display()

Scalar field on the 2-dimensional topological manifold $M$

```sage```
h.display()

$M \rightarrow \mathbb{R}$

on $U$: $(x, y) \mapsto \sqrt{2}\pi$

on $V$: $(u, v) \mapsto \sqrt{2}\pi$

```sage```
a = var('a')

```sage```
h = CM(a); h.display()

$M \rightarrow \mathbb{R}$

on $U$: $(x, y) \mapsto a$

on $V$: $(u, v) \mapsto a$
```

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()

$M \rightarrow \mathbb{R}$

on $U$: $(x, y) \mapsto a + x$

on $W$: $(u, v) \mapsto (a*u^2 + a*v^2 + u)/(u^2 + v^2)$

```sage```
h = CM(a+u); h.display()

$M \rightarrow \mathbb{R}$

on $W$: $(x, y) \mapsto (a*x^2 + a*y^2 + x)/(x^2 + y^2)$

on $V$: $(u, v) \mapsto a + u$
```

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()

$M \rightarrow \mathbb{R}$

1.6. Scalar Fields
**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
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
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
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
sage: CM.zero() is z
True
```

### 1.6.2 Scalar Fields

Given a topological manifold \( M \) over a topological field \( K \) (in most applications, \( K = \mathbb{R} \) or \( K = \mathbb{C} \)), a *scalar field* on \( M \) is a continuous map

\[
f : M \longrightarrow K
\]

Scalar fields are implemented by the class *ScalarField*.

**AUTHORS:**

- Travis Scrimshaw (2016): review tweaks
• Marco Mancini (2017): SymPy as an optional symbolic engine, alternative to SR
• Florentin Jaffredo (2018): series expansion with respect to a given parameter
• Michael Jung (2019): improve restrictions; make display() show all distinct expressions

REFERENCES:
• [Lee2011]
• [KN1963]

class sage.manifolds.scalarfield.ScalarField (parent, coord_expression=None, chart=None, name=None, latex_name=None)

Bases: CommutativeAlgebraElement, ModuleElementWithMutability

Scalar field on a topological manifold.

Given a topological manifold $M$ over a topological field $K$ (in most applications, $K = \mathbb{R}$ or $K = \mathbb{C}$), a scalar field on $M$ is a continuous map

$$f : M \rightarrow K.$$  

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)
```
(continues on next page)
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
```

(continues on next page)
Manifolds, Release 10.3

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

The method `coord_function()` returns instead a function of the chart coordinates, i.e. an instance of `ChartFunction`:

```
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.chart_func.ChartFunctionRing_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
```

The constant scalar fields zero and one are immutable, and therefore their expressions cannot be changed:

```
sage: zer.is_immutable()
True
sage: zer.set_expr(x)
Traceback (most recent call last):
  ... ValueError: the expressions of an immutable element cannot be changed
sage: one = M.one_scalar_field()
sage: one.is_immutable()
True
sage: one.set_expr(x)
Traceback (most recent call last):
```

(continues on next page)
... 

ValueError: the expressions of an immutable element cannot be changed

Other scalar fields can be declared immutable, too:

```python
sage: c.isImmutable()
False
sage: c.setImmutable()
sage: c.isImmutable()
True
sage: c.setExpr(y^2)
Traceback (most recent call last):
... 
ValueError: the expressions of an immutable element cannot be changed
```

Immutable elements are hashable and can therefore be used as keys for dictionaries:

```python
sage: {c: 1}[c]
1
```

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):

```python
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: zero(E)
0
```

A scalar field can be compared to another scalar field:

```python
sage: f == g
False
```

…to a symbolic expression:

```python
sage: f == x*y
False
sage: g == x*y
True
```
\begin{verbatim}
from sage.manifolds.manifold import Manifold
from sage.manifolds.scalarfield import ScalarField

# Define a manifold
M = Manifold(2, 'M', structure='topological')
U = M.open_subset('U', coordinates='x y')
V = M.open_subset('V', coordinates='u v')
M.declare_union(U, V)

# Define scalar fields
f = M.scalar_field({(x,y): x^2 + y^2, (u,v): u^2 + v^2})
c = M.scalar_field({(x,y): a, (u,v): b})
zer = M.scalar_field({(x,y): 0, (u,v): 0})
f2 = M.scalar_field({(x,y): 2, (u,v): 2})
M = M.scalar_field({(x,y): x, (u,v): u})

# Arithmetic operations
s = f + c
s.display()

# Standard mathematical functions
sqrt_f = sqrt(f)
sqrt_f.display()
tan_f = tan(f)
tan_f.display()

# Algebraic operations
f.parent() == M.scalar_field_algebra()
g.parent() == U.scalar_field_algebra()

# Consequence of scalar field operations
s = f + c
s.display()

# End of example
\end{verbatim}

---

Standard mathematical functions are implemented:

- `sqrt(f)`: Returns the square root of the scalar field `f`.
- `tan(f)`: Returns the tangent of the scalar field `f`.

Arithmetics of scalar fields:

Scalar fields on `M` (resp. `U`) belong to the algebra `C^0(M)` (resp. `C^0(U)`):
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 f+1 on the 2-dimensional topological manifold M
sage: s.display()
f+1: 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 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)
on W: (u, v) ↦ (u^4 + v^4 + u^3 + (2*u^2 + u)*v^2 + u)/(u^4 + v^4 + (2*u^2 + 1)*v^2 + u^2 + u^2)
sage: s = f + u; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M → R
on W: (x, y) ↦ (x^3 + (x + 1)*y^2 + x^2 + x)/(x^4 + y^4 + (2*x^2 + 1)*y^2 + x^2)
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:
The operation actually performed is \( f|_U + g \):

\[
\begin{align*}
\text{sage: } & s == f \text{.restrict}(U) + g \\
& \text{True}
\end{align*}
\]

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*):

\[
\begin{align*}
\text{sage: } & \text{CM = M.scalar_field_algebra()} \\
\text{sage: } & \text{CU = U.scalar_field_algebra()} \\
\text{sage: } & \text{CU.has_coerce_map_from(CM)} \\
& \text{True}
\end{align*}
\]

The coercion map is nothing but the restriction to domain \( U \):

\[
\begin{align*}
\text{sage: } & \text{CU.coerce(f) == f.restrict(U)} \\
& \text{True}
\end{align*}
\]

Since the algebra \( C^0(M) \) is a vector space over \( \mathbb{R} \), scalar fields can be multiplied by a number, either an explicit one:

\[
\begin{align*}
\text{sage: } & s = 2*f ; s \\
& \text{Scalar field on the 2-dimensional topological manifold M} \\
\text{sage: } & s\text{.display()} \\
& M \to \mathbb{R} \\
& \text{on U: } (x, y) \mapsto 2/(x^2 + y^2 + 1) \\
& \text{on V: } (u, v) \mapsto 2*(u^2 + v^2)/(u^2 + v^2 + 1)
\end{align*}
\]

or a symbolic one:

\[
\begin{align*}
\text{sage: } & s = a*f ; s \\
& \text{Scalar field on the 2-dimensional topological manifold M} \\
\text{sage: } & s\text{.display()} \\
& M \to \mathbb{R} \\
& \text{on U: } (x, y) \mapsto a/(x^2 + y^2 + 1) \\
& \text{on V: } (u, v) \mapsto (u^2 + v^2)*a/(u^2 + v^2 + 1)
\end{align*}
\]

However, if the symbolic variable is a chart coordinate, the multiplication is performed only in the corresponding chart:

\[
\begin{align*}
\text{sage: } & s = x*f; s \\
& \text{Scalar field on the 2-dimensional topological manifold M}
\end{align*}
\]

(continues on next page)
Some tests:

```
sage: 0*f == 0
True
sage: 0*f == zero
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 on the 2-dimensional topological manifold M
sage: s.display()
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 $C^0(M) \to C^0(U)$ 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 f*g on the Open subset U of the 2-dimensional topological
manifold M
sage: s.display()
f*g: 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):
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
```
Examples with SymPy as the symbolic engine

From now on, we ask that all symbolic calculus on manifold $M$ are performed by SymPy:

```python
sage: M.set_calculus_method('sympy')
```

We define $f$ as above:

```python
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
```

The scalar field $g$ defined on $U$:

```python
sage: g = U.scalar_field({c_xy: x*y}, name='g')
```

Definition on a single chart and subsequent completion:

```python
sage: f = M.scalar_field(1/(1+x^2+y^2), chart=c_xy, name='f')
sage: f.add_expr((u^2+v^2)/(1+u^2+v^2), chart=c_uv)
sage: f.display()
```

Definition without any coordinate expression and subsequent completion:

```python
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()
```

Use of `add_expr_by_continuation()`:

```python
sage: f = M.scalar_field(1/(1+x^2+y^2), chart=c_xy, name='f')
sage: f.add_expr_by_continuation(c_uv, U.intersection(V))
sage: f.display()
```

A scalar field defined by some unspecified function of the coordinates:

```python
sage: h = U.scalar_field(function('H')(x, y), name='h') ; h
```

(continues on next page)
The coordinate expression in a given chart is obtained via the method `expr()`, which in the present context, returns a SymPy object:

```
sage: f.expr(c_uv)
(u^2 + v^2)/(u^2 + v^2 + 1)
sage: type(f.expr(c_uv))
<class 'sympy.core.mul.Mul'>
```

The method `coord_function()` returns instead a function of the chart coordinates, i.e. an instance of `ChartFunction`:

```
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.chart_func.ChartFunctionRing_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
```

We may ask for the `SR` representation of the coordinate function:

```
sage: f.coord_function(c_uv).expr('SR')
(u^2 + v^2)/(u^2 + v^2 + 1)
```

A constant scalar field with SymPy representation:

```
sage: c = M.constant_scalar_field(2, name='c')
sage: c.display()
(2)
sage: type(c.expr(c_xy))
<class 'sympy.core.numbers.Integer'>
```

The constant value can be some unspecified parameter:

```
sage: var('a')
a
sage: c = M.constant_scalar_field(a, name='c')
sage: c.display()
(2)
sage: type(c.expr(c_xy))
<class 'sympy.core.symbol.Symbol'>
```

The zero scalar field:
Manifolds, Release 10.3

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
sage: type(zer.expr(c_xy))
<class 'sympy.core.numbers.Zero'>
sage: zer is M.zero_scalar_field()
True

Action of scalar fields on manifold's points:

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:

1.6. Scalar Fields
Scalar field `sqrt(f)` on the 2-dimensional topological manifold `M`

```python
sage: sqrt(f)
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)
```

Scalar field `tan(f)` on the 2-dimensional topological manifold `M`

```python
tan(f): M → ℝ
on U: (x, y) ↦ tan(1/(x**2 + y**2 + 1))
on V: (u, v) ↦ tan((u**2 + v**2)/(u**2 + v**2 + 1))
```

### Arithmetics of scalar fields with SymPy

Scalar fields on `M` (resp. `U`) belong to the algebra `C^0(M)` (resp. `C^0(U)`):

```python
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:

```python
sage: s = f + c ; s
Scalar field f+c on the 2-dimensional topological manifold M
sage: s.display()
s+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*u**2 + a*v**2 + a + u**2 + v**2)/(u**2 + v**2 + 1)
```

and subtracted:

```python
sage: s = f - c ; s
Scalar field f-c on the 2-dimensional topological manifold M
sage: s.display()
s-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*u**2 - a*v**2 - a + u**2 + v**2)/(u**2 + v**2 + 1)
```

Some tests:

```python
sage: f + zer == f
True
sage: f - f == zer
True
sage: f + (-f) == zer
True
sage: (f+c)-f == c
```
We may add a number (interpreted as a constant scalar field) to a scalar field:

```
sage: s = f + 1; s
Scalar field f+1 on the 2-dimensional topological manifold M
sage: s.display()
f+1: 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)
on W: (u, v) ↦ (u**4 + u**3 + 2*u**2*v**2 + u*v**2 + u + v**2)/(u**2 + v**2 + 1)
sage: s = f + u; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M → ℝ
 on W: (x, y) ↦ (x**3 + x**2 + x*y**2 + x + y**2)/(x**4 + 2*x**2*y**2 + x**2 + y**2)
on V: (u, v) ↦ (u**3 + u**2 + u*v**2 + u + v**2)/(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 f+g 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()
f+g: U → ℝ
 (x, y) ↦ (x**3*y + x*y**3 + x*y + 1)/(x**2 + y**2 + 1)
on W: (u, v) ↦ (u**6 + 3*u**4*v**2 + u**3*v + 3*u**2*v**4 + u*v**3 + u*v + v**6)/(u**2 + v**2 + 1)
```
The operation actually performed is $f|_U + g$:

```python
sage: s == f.restrict(U) + g
True
```

Since the algebra $C^0(M)$ is a vector space over $\mathbb{R}$, scalar fields can be multiplied by a number, either an explicit one:

```python
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:

```python
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) ↦ a*(u**2 + v**2)/(u**2 + v**2 + 1)
```

However, if the symbolic variable is a chart coordinate, the multiplication is performed only in the corresponding chart:

```python
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)
on W: (u, v) ↦ u/(u**2 + v**2 + 1)
sage: s = u*f; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M → ℝ
on W: (x, y) ↦ x/(x**4 + 2*x**2*y**2 + x**2 + y**4 + y**2)
on V: (u, v) ↦ u*(u**2 + v**2)/(u**2 + v**2 + 1)
```

Some tests:

```python
sage: 0*f == 0
True
sage: 0*f == zero
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:

```python
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 + 2*x**2*y**2 + 2*x**2 + y**4 + 2*y**2 + 1)
```

(continues on next page)
on \( V: (u, v) \mapsto (u^4 + 2u^2v^2 + v^4)/(u^4 + 2u^2v^2 + 2u^2 + v^4 + v^2 + 1) \)

\[ \text{sage: } s = g*h ; s \]
Scalar field \( g*h \) on the Open subset \( U \) of the 2-dimensional topological manifold \( M \)
\[ \text{sage: } s.display() \]
\( g*h: U \to \mathbb{R} \)
\( (x, y) \mapsto x*y*H(x, y) \)
on \( W: (u, v) \mapsto u*v*H(u/(u^2 + v^2), v/(u^2 + v^2))/(u^4 + 2u^2v^2 + v^4) \)

Thanks to the coercion \( C^0(M) \to C^0(U) \) 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 \( U \):

\[ \text{sage: } f.domain(), g .domain() \]
\((2-dimensional topological manifold M, \ Open subset U of the 2-dimensional topological manifold M) \)
\[ \text{sage: } s = f* g ; s \]
Scalar field \( f* g \) on the Open subset \( U \) of the 2-dimensional topological manifold \( M \)
\[ \text{sage: } s.display() \]
\( f* g: U \to \mathbb{R} \)
\( (x, y) \mapsto x*y/(x^2 + y^2 + 1) \)
on \( W: (u, v) \mapsto u*v/(u^4 + 2u^2v^2 + u^2 + v^4 + v^2) \)

\[ \text{sage: } s == f.restrict(U)*g \]
True

Scalar fields can be divided (pointwise division):

\[ \text{sage: } s = f/c ; s \]
Scalar field \( f/c \) on the 2-dimensional topological manifold \( M \)
\[ \text{sage: } s.display() \]
f/c: \( M \to \mathbb{R} \)
on \( U: (x, y) \mapsto 1/(a*(x^2 + y^2 + 1)) \)
on \( V: (u, v) \mapsto (u^2 + v^2)/(a*(u^2 + v^2 + 1)) \)
\[ \text{sage: } s = g/h ; s \]
Scalar field \( g/h \) on the Open subset \( U \) of the 2-dimensional topological manifold \( M \)
\[ \text{sage: } s.display() \]
g/h: \( U \to \mathbb{R} \)
\( (x, y) \mapsto x*y/H(x, y) \)
on \( W: (u, v) \mapsto u*v/((u^4 + 2u^2v^2 + v^4)*H(u/(u^2 + v^2), v/(u^2 + v^2))/(u^4 + 2u^2v^2 + v^4 + v^2)) \)
\[ \text{sage: } s = f/g ; s \]
Scalar field \( f/g \) on the Open subset \( U \) of the 2-dimensional topological manifold \( M \)
\[ \text{sage: } s.display() \]
f/g: \( U \to \mathbb{R} \)
\( (x, y) \mapsto 1/(x*y*(x^2 + y^2 + 1)) \)
on \( W: (u, v) \mapsto (u^6 + 3u^4v^2 + 3u^2v^4 + v^6)/(u^8v^2*u^2 + v^2 + 1) \)
\[ \text{sage: } s == f.restrict(U)/g \]
True

For scalar fields defined on a single chart domain, we may perform some arithmetics with symbolic expressions involving the chart coordinates:
\begin{verbatim}
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*y - y
on \mathcal{W}: (u, v) ↦ (-u**2*v + u**2 + u*v - v**3)/(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 \mathcal{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 \mathcal{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 \mathcal{W}: (u, v) ↦ u**2/v + v
\end{verbatim}

The test suite is passed:

\begin{verbatim}
sage: TestSuite(f).run()
sage: TestSuite(zer).run()
\end{verbatim}

\texttt{add_expr} (\texttt{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 \texttt{set_expr()} instead.

INPUT:

\begin{itemize}
  \item \texttt{coord_expression} -- coordinate expression of the scalar field
  \item \texttt{chart} -- (default: None) chart in which \texttt{coord_expression} is defined; if None, the default chart of the scalar field's domain is assumed
\end{itemize}

\textbf{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.

\textbf{EXAMPLES:}

Adding scalar field expressions on a 2-dimensional manifold:

\begin{verbatim}
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
\end{verbatim}

(continues on next page)
Since zero and one are special elements, their expressions cannot be changed:

```python
sage: z = M.zero_scalar_field()
sage: z.add_expr(cos(u)-sin(v), c_uv)
Traceback (most recent call last):
  ... ValueError: the expressions of an immutable element cannot be changed
sage: one = M.one_scalar_field()
sage: one.add_expr(cos(u)-sin(v), c_uv)
Traceback (most recent call last):
  ... ValueError: the expressions of an immutable element cannot be changed
```

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 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 \(S^2\):

```python
sage: M = Manifold(2, 'S^2', structure='topological')
sage: U = M.open_subset('U') ; V = M.open_subset('V') # the complement of 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))
```

The scalar field has been defined only on the domain covered by the chart \(c_{xy}\), i.e. \(U\):
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:

We use this fact to extend the definition of $f$ to the open subset $V$, covered by the chart $c_{uv}$:

Then, $f$ is known on the whole sphere:

Arc cosine of the scalar field.

OUTPUT:

- the scalar field $\arccos f$, where $f$ is the current scalar field

EXAMPLES:

The notation $\acos$ can be used as well:

Some tests:

**arccosh()**

Inverse hyperbolic cosine of the scalar field.

**OUTPUT:**

- the scalar field \( \text{arccosh} f \), where \( f \) is the current scalar field

**EXAMPLES:**

```python
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)
\ \arccosh\left(\Phi\right)
sage: g.display()
\arccosh(f): M \to \mathbb{R} \ 
(x, y) \mapsto \arccosh(x*y)
```

The notation \( \text{acosh} \) can be used as well:

```python
sage: acosh(f)
Scalar field acosh(f) on the 2-dimensional topological manifold M
sage: acosh(f) == g
True
```

Some tests:

```python
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 \( \text{arcsin} f \), where \( f \) is the current scalar field

**EXAMPLES:**

```python
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 \to \mathbb{R} 
(x, y) \mapsto \arcsin(x*y)
```

The notation \( \text{asin} \) can be used as well:

```python
sage: asin(f)
Scalar field arcsin(f) on the 2-dimensional topological manifold M
sage: asin(f) == g
True
```
Some tests:

```python
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 arcsinh $f$, where $f$ is the current scalar field

EXAMPLES:

```python
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{arcsinh}\left(\Phi\right)
```

The notation `asinh` can be used as well:

```python
sage: asinh(f)
Scalar field arcsinh(f) on the 2-dimensional topological manifold M
sage: asinh(f) == g
True
```

Some tests:

```python
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:

```python
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)
```

(continues on next page)
The notation \texttt{atan} can be used as well:

```
sage: atan(f)
Scalar field \texttt{atan}(f) on the 2-dimensional topological manifold \texttt{M}
sage: atan(f) == g
True
```

Some tests:

```
sage: tan(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
```

\texttt{arctanh}()

Inverse hyperbolic tangent of the scalar field.

\textbf{OUTPUT:}

- the scalar field \texttt{arctanh} $f$, where $f$ is the current scalar field

\textbf{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 \texttt{arctanh}(f) on the 2-dimensional topological manifold \texttt{M}
sage: latex(g)
\mathrm{arctanh}\left(\Phi\right)
sage: g.display()
\texttt{arctanh}(f): \texttt{M} \rightarrow \mathbb{R}
\quad (x, y) \mapsto \texttt{arctanh}(x*y)
```

The notation \texttt{atanh} can be used as well:

```
sage: atanh(f)
Scalar field \texttt{atanh}(f) on the 2-dimensional topological manifold \texttt{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
```
codomain()  
Return the codomain of the scalar field.

EXAMPLES:

```python
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: f = M.scalar_field(x+2*y)
sage: f.codomain()
Real Field with 53 bits of precision
```

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:

```python
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))]
```

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`:

```python
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))]
```

(continues on next page)
{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: g.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  # random (dictionary output)}  
{Chart (U, (x, y)): x + y, Chart (V, (u, v)): u, Chart (W, (u, v)): u}  

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 ChartFunction 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.chart_func.ChartFunctionRing_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: 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(name=None, latex_name=None)
Return an exact copy of the scalar field.

INPUT:

- name – (default: None) name given to the copy
- latex_name – (default: None) LaTeX symbol to denote the copy; if none is provided, the LaTeX symbol is set to name

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()
```
copy_from(other)

Make self a copy of other.

INPUT:

- other -- other scalar field, in the same module as self

Note: While the derived quantities are not copied, the name is kept.

Warning: All previous defined expressions and restrictions will be deleted!

EXAMPLES:

```python
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: f = M.scalar_field(x*y^2, name='f')
sage: f.display()
f: M → ℝ
  (x, y) ↦ x*y^2
sage: g = M.scalar_field(name='g')
sage: g.copy_from(f)
sage: g.display()
g: M → ℝ
  (x, y) ↦ x*y^2
sage: f == g
True
```

While the original scalar field is modified, the copy is not:

```python
sage: f.set_expr(x-y)
sage: f.display()
f: M → ℝ
  (x, y) ↦ x - y
sage: g.display()
g: M → ℝ
  (x, y) ↦ x*y^2
sage: f == g
False
```

cos()

Cosine of the scalar field.

OUTPUT:
- the scalar field \( \cos f \), where \( f \) is the current scalar field

**EXAMPLES:**

```python
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 \( \cos(\Phi) \) on the 2-dimensional topological manifold \( M \)
sage: latex(g)
\cos(\Phi)
sage: g.display()
cos(f): M \to \mathbb{R}
\( (x, y) \mapsto \cos(x*y) \)
```

Some tests:

```python
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:**

```python
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 \( \cosh(\Phi) \) on the 2-dimensional topological manifold \( M \)
sage: latex(g)
\cosh(\Phi)
sage: g.display()
cosh(f): M \to \mathbb{R}
\( (x, y) \mapsto \cosh(x*y) \)
```

Some test:

```python
sage: cosh(M.zero_scalar_field()) == M.constant_scalar_field(1)
True
```

\( \text{disp} \) (**chart=**`None`)

Display the expression of the scalar field in a given chart.

Without any argument, this function displays all known, distinct expressions.

**INPUT:**

- \texttt{chart} – (default: **None**) chart with respect to which the coordinate expression is to be displayed; if **None**, the display is performed in all the greatest charts in which the coordinate expression is known

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

**EXAMPLES:**

Various displays:
A shortcut of `display()` is `disp()`:

```python
sage: f.disp()
f: M → ℝ
  (x, y) ↦ sqrt(x + 1)
```

In case the scalar field is piecewise-defined, the `display()` command still outputs all expressions. Each expression displayed corresponds to the chart on the greatest domain where this particular expression is known:

```python
sage: U = M.open_subset('U')
sage: f.set_expr(y^2, c_xy.restrict(U))
sage: f.display()
| f: M → ℝ |
| on U : (x, y) ↦ y^2 |

display(chart=None)
Display the expression of the scalar field in a given chart.

Without any argument, this function displays all known, distinct expressions.

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 greatest charts in which the coordinate expression is known

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

EXAMPLES:

Various displays:

```python
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} \\
 & \left(x, y\right) & \longmapsto & \sqrt{x + 1} \end{array}
```

(continues on next page)
A shortcut of \texttt{display()} is \texttt{disp():}

\begin{Verbatim}
\texttt{sage: f.disp()}
\texttt{f: M \rightarrow \mathbb{R}}
\texttt{(x, y) \mapsto \sqrt{x + 1}}
\end{Verbatim}

In case the scalar field is piecewise-defined, the \texttt{display()} command still outputs all expressions. Each expression displayed corresponds to the chart on the greatest domain where this particular expression is known:

\begin{Verbatim}
\texttt{sage: U = M.open_subset('U')}
\texttt{sage: f.set_expr(y^2, c_xy.restrict(U))}
\texttt{sage: f.display()}
\texttt{f: M \rightarrow \mathbb{R}}
\texttt{on \textbf{U}: (x, y) \mapsto y^2}
\end{Verbatim}

\texttt{domain()}

Return the open subset on which the scalar field is defined.

\textbf{OUTPUT:}

\begin{itemize}
\item instance of class \textit{TopologicalManifold} representing the manifold’s open subset on which the scalar field is defined
\end{itemize}

\textbf{EXAMPLES:}

\begin{Verbatim}
\texttt{sage: M = Manifold(2, ‘M’, structure=’topological’)}
\texttt{sage: c_xy.<x,y> = M.chart()}
\texttt{sage: f = M.scalar_field(x+2*y)}
\texttt{sage: f.domain()}
\texttt{2-dimensional topological manifold M}
\texttt{sage: U = M.open_subset(‘U’, coord_def={c_xy: x<0})}
\texttt{sage: g = f.restrict(U)}
\texttt{sage: g.domain()}
\texttt{Open subset U of the 2-dimensional topological manifold M}
\end{Verbatim}

\texttt{exp()}

Exponential of the scalar field.

\textbf{OUTPUT:}

\begin{itemize}
\item the scalar field \texttt{exp} \( f \), where \( f \) is the current scalar field
\end{itemize}

\textbf{EXAMPLES:}

\begin{Verbatim}
\texttt{sage: M = Manifold(2, ‘M’, structure=’topological’)}
\texttt{sage: X.<x,y> = M.chart()}
\end{Verbatim}
Automatic simplifications occur:

```sage
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
sage: log(exp(f)) == f
True
```

Some tests:

```sage
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:**

- the coordinate expression of the scalar field in the given chart, either as a Sage's symbolic expression or as a SymPy object, depending on the symbolic calculus method used on the chart

**EXAMPLES:**

Expression of a scalar field on a 2-dimensional manifold:

```sage
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
```

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}

Note that the object returned by expr() depends on the symbolic backend used for coordinate computations:

sage: type(f.expr())
<class 'sage.symbolic.expression.Expression'>
sage: M.set_calculus_method('sympy')
sage: type(f.expr())
<class 'sympy.core.mul.Mul'>
sage: f.expr() # note the SymPy exponent notation
x*y**2

**is_trivial_one()**

Check if self is trivially equal to one without any simplification.

This method is supposed to be fast as compared with self == 1 and is intended to be used in library code where trying to obtain a mathematically correct result by applying potentially expensive rewrite rules is not desirable.

**EXAMPLES:**

sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: 1})
sage: f.is_trivial_one()
True
sage: f = M.scalar_field(1)

sage: f.is_trivial_one()
True
sage: M.one_scalar_field().is_trivial_one()
True
sage: f = M.scalar_field({X: x+y})
sage: f.is_trivial_one()
False

Scalar field defined by means of two charts:

sage: U1 = M.open_subset('U1'); X1.<x1,y1> = U1.chart()
sage: U2 = M.open_subset('U2'); X2.<x2,y2> = U2.chart()
sage: f = M.scalar_field>({X1: 1, X2: 1})
sage: f.is_trivial_one()
True
sage: f = M.scalar_field([{X1: 0, X2: 1})

sage: f.is_trivial_one()
False
No simplification is attempted, so that `False` is returned for non-trivial cases:

```python
sage: f = M.scalar_field({X: cos(x)^2 + sin(x)^2})
sage: f.is_trivial_one()
False
```

On the contrary, the method `is_zero()` and the direct comparison to one involve some simplification algorithms and return `True`:

```python
sage: (f - 1).is_zero()
True
sage: f == 1
True
```

**is_trivial_zero()**

Check if `self` is trivially equal to zero without any simplification.

This method is supposed to be fast as compared with `self.is_zero()` or `self == 0` and is intended to be used in library code where trying to obtain a mathematically correct result by applying potentially expensive rewrite rules is not desirable.

**EXAMPLES:**

```python
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: 0})
sage: f.is_trivial_zero()
True
sage: f = M.scalar_field(0)
sage: f.is_trivial_zero()
True
sage: M.zero_scalar_field().is_trivial_zero()
True
sage: f = M.scalar_field({X: x+y})
sage: f.is_trivial_zero()
False
```

Scalar field defined by means of two charts:

```python
sage: U1 = M.open_subset('U1'); X1.<x1,y1> = U1.chart()
sage: U2 = M.open_subset('U2'); X2.<x2,y2> = U2.chart()
sage: f = M.scalar_field({X1: 0, X2: 0})
sage: f.is_trivial_zero()
True
sage: f = M.scalar_field({X1: 0, X2: 1})
sage: f.is_trivial_zero()
False
```

No simplification is attempted, so that `False` is returned for non-trivial cases:

```python
sage: f = M.scalar_field({X: cos(x)^2 + sin(x)^2 - 1})
sage: f.is_trivial_zero()
False
```

On the contrary, the method `is_zero()` and the direct comparison to zero involve some simplification algorithms and return `True`:
```python
sage: f.is_zero()
True
sage: f == 0
True
```

**is_unit()**

Return `True` iff `self` is not trivially zero in at least one of the given expressions since most scalar fields are invertible and a complete computation would take too much time.

**EXAMPLES:**

```python
sage: M = Manifold(2, 'M', structure='top')
sage: one = M.scalar_field_algebra().one()  
sage: one.is_unit()
True
sage: zero = M.scalar_field_algebra().zero()  
sage: zero.is_unit()
False
```

**log()**

Natural logarithm of the scalar field.

**OUTPUT:**

- the scalar field \(\ln f\), where \(f\) is the current scalar field

**EXAMPLES:**

```python
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 \rightarrow \mathbb{R}
\(x, y\) ↦ \log(x + y)
sage: latex(g)
\ln\left(\Phi\right)
```

The inverse function is `exp()`:

```python
sage: exp(log(f)) == f
True
```

**preimage** *(codomain_subset, name=None, latex_name=None)*

Return the preimage of `codomain_subset`.

An alias is `pullback()`.

**INPUT:**

- `codomain_subset` – an instance of `RealSet`
- `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`

**OUTPUT:**

- either a `TopologicalManifold` or a `ManifoldSubsetPullback`
**EXAMPLES:**

```python
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x+y}, name='f')
sage: L = f.pullback(RealSet.point(1)); latex(L)
f^{-1}(\{1\})
sage: M((-1, 1)) in L
False
sage: M((0, 1)) in L
True
sage: M.zero_scalar_field().preimage(RealSet.point(0)) is M
True
```

**pullback (codomain_subset, name=None, latex_name=None)**

Return the preimage of `codomain_subset`.

An alias is `pullback()`.

**INPUT:**

- `codomain_subset` – an instance of `RealSet`
- `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`

**OUTPUT:**

- either a `TopologicalManifold` or a `ManifoldSubsetPullback`

**EXAMPLES:**

```python
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x+y}, name='f')
sage: L = f.pullback(RealSet.point(1)); latex(L)
f^{-1}(\{1\})
sage: M((-1, 1)) in L
False
sage: M((0, 1)) in L
True
sage: M.zero_scalar_field().preimage(RealSet.point(0)) is M
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 $\mathbb{R}^2$ 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 \( M \)

\[
\begin{align*}
\text{sage: } f_U.display() \\
&f: U \to \mathbb{R} \\
&(x, y) \mapsto \cos(x y)
\end{align*}
\]

\[
\begin{align*}
\text{sage: } f_U.parent() \\
&\text{Algebra of scalar fields on the Open subset } U \text{ of the 2-dimensional topological manifold } M
\end{align*}
\]

The restriction to the whole domain is the identity:

\[
\begin{align*}
\text{sage: } & f.restrict(M) \text{ is } f \\
&\text{True} \\
\text{sage: } & f_U.restrict(U) \text{ is } f_U \\
&\text{True}
\end{align*}
\]

Restriction of the zero scalar field:

\[
\begin{align*}
\text{sage: } & M.zero_scalar_field().restrict(U) \\
&\text{Scalar field zero on the Open subset } U \text{ of the 2-dimensional topological manifold } M
\end{align*}
\]

\[
\begin{align*}
\text{sage: } & M.zero_scalar_field().restrict(U) \text{ is } U.zero_scalar_field() \\
&\text{True}
\end{align*}
\]

**set_calc_order** *(symbol, order, truncate=False)*

Trigger a powerseries expansion with respect to a small parameter in computations involving the scalar field. This property is propagated by usual operations. The internal representation must be **SR** for this to take effect.

If the small parameter is \( \epsilon \) and \( f \) is **self**, the power series expansion to order \( n \) is

\[
f = f_0 + \epsilon f_1 + \epsilon^2 f_2 + \cdots + \epsilon^n f_n + O(\epsilon^{n+1}),
\]

where \( f_0, f_1, \ldots, f_n \) are \( n+1 \) scalar fields that do not depend upon \( \epsilon \).

**INPUT:**

- **symbol** – symbolic variable (the “small parameter” \( \epsilon \)) with respect to which the coordinate expressions of **self** in various charts are expanded in power series (around the zero value of this variable)
- **order** – integer; the order \( n \) of the expansion, defined as the degree of the polynomial representing the truncated power series in **symbol**

**Warning:** The order of the big \( O \) in the power series expansion is \( n + 1 \), where \( n \) is **order**.

- **truncate** – (default: False) determines whether the coordinate expressions of **self** are replaced by their expansions to the given order

**EXAMPLES:**
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: t = var('t')  # the small parameter
sage: f = M.scalar_field(exp(-t*x))
sage: f.expr()
e^(-t*x)
sage: f.set_calc_order(t, 2, truncate=True)
sage: f.expr()
1/2*t^2*x^2 - t*x + 1

**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:

```python
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}
```

Since zero and one are special elements, their expressions cannot be changed:

```python
sage: z = M.zero_scalar_field()
sage: z.set_expr(3*y)
Traceback (most recent call last):
  ... ValueError: the expressions of an immutable element cannot be changed
sage: one = M.one_scalar_field()
sage: one.set_expr(3*y)
Traceback (most recent call last):
  ... ValueError: the expressions of an immutable element cannot be changed
```

**set_immutable** ()
Set self and all restrictions of self immutable.

EXAMPLES:

```python
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: U = M.open_subset('U', coord_def={X: x^2+y^2<1})  # disk
sage: V = M.open_subset('V', coord_def={X: x>0})  # half plane
sage: f = M.scalar_field(x^2, name='f')
sage: fU = f.restrict(U)
sage: f.set_immutable()
sage: fU.is_immutable()
True
sage: f.restrict(V).is_immutable()
True
```

**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:**

```python
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x+y})
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
```

**set_restriction** *(rst)*

Define a restriction of self to some subdomain.

**INPUT:**

- rst – *ScalarField* defined on a subdomain of the domain of self

**EXAMPLES:**

```python
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: f = M.scalar_field(name='f')
sage: g = U.scalar_field(x^2+y)
```

(continues on next page)
\textbf{sin()}

Sine of the scalar field.

\textbf{OUTPUT:}

\begin{itemize}
  \item the scalar field $\sin f$, where $f$ is the current scalar field
\end{itemize}

\textbf{EXAMPLES:}

\begin{Verbatim}
\begin{split}
\text{sage: } & M = \text{Manifold}(2, 'M', \text{structure}='\text{topological}') \\
\text{sage: } & X.x, y = M.\text{chart}() \\
\text{sage: } & f = M.\text{scalar_field}((X: x' y), \text{name}='f', \text{latex_name}=r'\Phi') \\
\text{sage: } & g = \text{sin}(f); g \\
\end{split}
\end{Verbatim}

Scalar field $\sin(f)$ on the 2-dimensional topological manifold $M$

\begin{Verbatim}
\begin{split}
\text{sage: } & \text{latex}(g) \\
\text{sage: } & g.\text{display()} \\
\text{sage: } & \text{sin}(f): M \rightarrow \mathbb{R} \\
\text{True} \\
\end{split}
\end{Verbatim}

Some tests:

\begin{Verbatim}
\begin{split}
\text{sage: } & \text{sin}(M.\text{zero_scalar_field}()) == M.\text{zero_scalar_field}() \\
\text{True} \\
\text{sage: } & \text{sin}(M.\text{constant_scalar_field}(\pi/2)) == M.\text{constant_scalar_field}(1) \\
\text{True} \\
\end{split}
\end{Verbatim}

\textbf{sinh()}

Hyperbolic sine of the scalar field.

\textbf{OUTPUT:}

\begin{itemize}
  \item the scalar field $\sinh f$, where $f$ is the current scalar field
\end{itemize}

\textbf{EXAMPLES:}

\begin{Verbatim}
\begin{split}
\text{sage: } & M = \text{Manifold}(2, 'M', \text{structure}='\text{topological}') \\
\text{sage: } & X.x, y = M.\text{chart}() \\
\text{sage: } & f = M.\text{scalar_field}((X: x' y), \text{name}='f', \text{latex_name}=r'\Phi') \\
\text{sage: } & g = \text{sinh}(f); g \\
\end{split}
\end{Verbatim}

Scalar field $\sinh(f)$ on the 2-dimensional topological manifold $M$

\begin{Verbatim}
\begin{split}
\text{sage: } & \text{latex}(g) \\
\text{sage: } & g.\text{display()} \\
\text{sage: } & \sinh(f): M \rightarrow \mathbb{R} \\
\text{True} \\
\end{split}
\end{Verbatim}

Some test:

\begin{Verbatim}
\begin{split}
\text{sage: } & \text{sinh}(M.\text{zero_scalar_field}()) == M.\text{zero_scalar_field}() \\
\text{True} \\
\end{split}
\end{Verbatim}
$$\sqrt{}$$

Square root of the scalar field.

**OUTPUT:**

- the scalar field $\sqrt{f}$, where $f$ is the current scalar field

**EXAMPLES:**

```python
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:

```python
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:**

```python
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:

```python
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:

```python
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): \text{\( M \)} \rightarrow \mathbb{R}
\quad (x, y) \mapsto \sinh(x*y)/\cosh(x*y)
```

Some tests:

```python
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:

- Eric Gourgoulhon (2015): initial version
- Travis Scrimshaw (2016): review tweaks

REFERENCES:

- [Lee2011]
- [KN1963]

```python
class sage.manifolds.manifold_homset.TopologicalManifoldHomset(domain, codomain, name=None, latex_name=None)
```

Bases: `UniqueRepresentation, 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:

- \text{domain} = `TopologicalManifold`: the domain topological manifold \( M \) of the morphisms
- \text{codomain} = `TopologicalManifold`: the codomain topological manifold \( N \) of the morphisms
• name – (default: None) string; the name of self; if None, 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:

```python
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 \( N \):  

```python
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:

```python
sage: TestSuite(H).run()
```

When the codomain coincides with the domain, the homset is a set of \emph{endomorphisms} in the category of topological manifolds:

```python
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:
This was of course not the case of $\text{H} = \text{Hom}(M, N)$:

\begin{Verbatim}
sage: H in Monoids()
False
\end{Verbatim}

The identity element of the monoid is of course the identity map of $M$:

\begin{Verbatim}
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 \to M
  (x, y) \mapsto (x, y)
\end{Verbatim}

The test suite is passed by $E$:

\begin{Verbatim}
sage: TestSuite(E).run()
\end{Verbatim}

This test suite includes more tests than in the case of $\text{H}$, since $E$ has some extra structure (monoid).

**Element**

\texttt{one}()

Return the identity element of \texttt{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 \texttt{ContinuousMap}

**EXAMPLES:**

The identity map of a 2-dimensional manifold:

\begin{Verbatim}
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 \to M
  (x, y) \mapsto (x, y)
\end{Verbatim}

The identity map is cached:

\begin{Verbatim}
1.7. Continuous Maps 215
\end{Verbatim}
If the homset is not a set of endomorphisms, the identity element is meaningless:

```python
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:**
- Travis Scrimshaw (2016): review tweaks

**REFERENCES:**
- Chap. 1 of [KN1963]
- [Lee2011]

```python
class sage.manifolds.continuous_map.ContinuousMap (parent, coord_functions=None, name=None, latex_name=None, is_isomorphism=False, is_identity=False)
```

**Bases:** `Morphism`

Continuous map between two topological manifolds.

This class implements continuous maps of the type

$$
\Phi : M \rightarrow N,
$$

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 an 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 $\mathbb{R}^3$:

```python
sage: M = Manifold(2, 'S^2', structure='topological')  # the 2-dimensional sphere
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()
```

```python
sage: N = Manifold(3, 'R^3', latex_name=r'\mathbb{R}^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')
```

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:

```python
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)/(x^2+y^2+1)],
        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:

```python
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)/(x^2+y^2+1)],
        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:

```python
sage: Phi1.display()
Phi: $S^2 \to \mathbb{R}^3$
   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():

```python
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: $S^2 \to \mathbb{R}^3$
   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:

```python
sage: Phi1 == Phi
True
```

The map acts on points:

```python
sage: np = M.point((0,0), chart=c_uv)  # the North pole
sage: Phi(np)
Point on the 3-dimensional topological manifold $\mathbb{R}^3$
  (0, 0, 1)
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:

```python
sage: TestSuite(Phi).run()
sage: TestSuite(Phi1).run()
```

Continuous maps can be composed by means of the operator *. Let us introduce the map $\mathbb{R}^3 \to \mathbb{R}^2$ corresponding to the projection from the point $(X, Y, Z) = (0, 0, 1)$ onto the equatorial plane $Z = 0$: 
Manifolds, Release 10.3

sage: P = Manifold(2, 'R^2', latex_name=r'\RR^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 → 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 R → 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)

(continues on next page)
An example of homeomorphism between the unit open disk and the Euclidean plane $\mathbb{R}^2$:

```
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.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 $\mathbb{R}^2$
to the Open subset D of the 2-dimensional topological manifold $\mathbb{R}^2$
sage: Phi.inverse().display()
$\Phi^{-1}$: $\mathbb{R}^2$ → 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 $\Phi^{-1}$ 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:
A special case of homeomorphism: the identity map of the open unit disk:

```sage
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}
```

The identity map acting on a point:

```sage
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
sage: id.display()
Id_D: D → D
  \( (x, y) \rightarrow (x, y) \)
```

The identity map is its own inverse:

```sage
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:

```python
sage: M = Manifold(2, 'R^2', latex_name=r'\mathbb{R}^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(rr:(0,+oo) ph:(0,2*pi): \phi) # spherical coordinates on U

We construct the links between spherical coordinates and Cartesian ones:

```python
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))
```

Check of the inverse coordinate transformation:

- `x == x` passed
- `y == y` passed
- `r == r` passed
- `ph == arctan2(r*sin(ph), r*cos(ph))` failed

NB: a failed report can reflect a mere lack of simplification.

```python
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:

```python
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:

```python
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:

```python
sage: rot._coord_expression
{...}
```

(continues on next page)
(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:

```python
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()`:

```python
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:

```python
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:

```
add_expression(Chart (U, (r, ph)), Chart (U, (r, ph))):
Coordinate functions (r, 1/3*pi + ph) on the Chart (U, (r, ph))}
```

1.7. Continuous Maps 223
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))
```

Check of the inverse coordinate transformation:
- `x == x` *passed*
- `y == y` *passed*
- `r == r` *passed*
- `ph == arctan2(r*sin(ph), r*cos(ph))` **failed**

NB: a failed report can reflect a mere lack of simplification.

```
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._coord_expression
```

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
```

(continues on next page)
((Chart (U, (r, ph)), Chart (U, (r, ph))): 
Coordinate functions \((r, 1/3*\pi + \text{ph})\) on the Chart \((U, (r, \text{ph}))\)

It is recovered (thanks to the known change of coordinates) by a call to \texttt{display()}:

\begin{verbatim}
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 + \text{ph}) on the Chart \((U, (r, \text{ph}))\)
\end{verbatim}

The rotation can be applied to a point by means of either coordinate system:

\begin{verbatim}
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
\end{verbatim}

\texttt{coord\_functions} \((\text{chart1}=\text{None}, \text{chart2}=\text{None})\)

Return the functions of the coordinates representing \texttt{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.

\textbf{INPUT:}

\begin{itemize}
\item chart1 – (default: \text{None}) chart on the domain of \texttt{self}; if \text{None}, the domain’s default chart is assumed
\item chart2 – (default: \text{None}) chart on the codomain of \texttt{self}; if \text{None}, the codomain’s default chart is assumed
\end{itemize}

\textbf{OUTPUT:}

\begin{itemize}
\item a \texttt{MultiCoordFunction} representing the continuous map in the above two charts
\end{itemize}

\textbf{EXAMPLES:}

Continuous map from a 2-dimensional manifold to a 3-dimensional one:

\begin{verbatim}
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...
\end{verbatim}
→default charts
Coordinate functions \((u*v, u/v, u + v)\) on the Chart \((M, (u, v))\)
\[
sage: \text{type}(\text{Phi coord functions()})
\]
\[
<\text{class 'sage.manifolds.chart_func.MultiCoordFunction'>}
\]

Coordinate representation in other charts:
\[
sage: \text{c_UV.UV} = M.\text{chart()} \quad \# \text{new chart on } M
\]
\[
sage: \text{ch_UV.UV \text{inverse}(U,V)} \quad \begin{pmatrix}1/2*U + 1/2*V,
-1/2*U + 1/2*V\end{pmatrix}
\]
\[
sage: \text{c_XYZ.XYZ} = N.\text{chart()} \quad \# \text{new chart on } N
\]
\[
sage: \text{ch_XYZ.XYZ \text{inverse}(X,Y,Z)} \quad \begin{pmatrix}2*x - 3*y + z, y + z - x, -x + 2*y - z\end{pmatrix}
\]
\[
sage: \text{Phi coord functions(c_UV, c_XYZ)}
\]

Coordinate representation with respect to a subchart in the domain:
\[
sage: A = M.\text{open_subset('A', coord_def={c_UV: u>0})}
\]
\[
sage: \text{Phi coord functions(c_UV restrict(A), c_XYZ)}
\]

Coordinate representation with respect to a superchart in the codomain:
\[
sage: B = N.\text{open_subset('B', coord_def={c_XYZ: x<0})}
\]
\[
sage: c_XYZ_B = c_XYZ.\text{restrict(B)}
\]
\[
sage: \text{Phi1 = M.\text{continuous_map}(B, \{(c_UV, c_XYZ_B): (u*v, u/v, u+v)})}
\]
\[
sage: \text{Phi1 coord functions(c_UV, c_XYZ_B)} \quad \# \text{definition charts}
\]
\[
sage: \text{Phi1 coord functions(c_UV, c_XYZ)} \quad \# \text{c_XYZ = superchart of c_XYZ_B}
\]

Coordinate representation with respect to a pair (subchart, superchart):
\[
sage: \text{Phi1 coord functions(c_UV restrict(A), c_XYZ)}
\]

Same example with SymPy as the symbolic calculus engine:
\[
sage: M.\text{set_calculus_method('sympy')}
\]
\[
sage: N.\text{set_calculus_method('sympy')}
\]
\[
sage: \text{Phi = M.\text{continuous_map}(N, (u*v, u/v, u+v), name='Phi'),}
\]
....: latex_name=r'\Phi')
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(c_UV, c_xyz)
Coordinate functions (-U**2/4 + V**2/4, (-U - V)/(U - V), V) on the Chart (M, (U, V))
sage: Phi.coord_functions(c_UV, c_XYZ)
Coordinate functions ((-U**3 + U**2*V + U*V**2 + 2*U*V + 6*U - V**3 - 2*V**2 + 6*V)/(2*(U - V)), (U**3/4 - U**2*V/4 - U*V**2/4 + U*V - U + V**3/4 - V**2 - V)/(U - V), (U**3 - U**2*V - U*V**2 - 4*U*V - 8*U + V**3 + 4*V**2 - 8*V)/(4*(U - V))) on the Chart (M, (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:
A simple reparametrization:
sage: R.<t> = manifolds.RealLine()
sage: I = R.open_interval(0, 2*pi)
sage: J = R.open_interval(2*pi, 6*pi)
sage: h = J.continuous_map(I, ((t -2*pi)/2,), name = h)
sage: h.display()
h: (2*pi, 6*pi) \rightarrow (0, 2*pi)
t ↦ t = -pi + 1/2*t
sage: latex(h.display())
\begin{array}{llcl} h:& \left(2 \, \pi, 6 \, \pi\right) & \longrightarrow & \left(0, 2 \, \pi\right) \\
& t & \longmapsto & t = -\pi + \frac{1}{2} \, t \end{array}

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'\mathbb{R}^3', structure='topological') # $\mathbb{R}^3$
sage: c_cart.<X,Y,Z> = N.chart() # Cartesian coordinates on $\mathbb{R}^3$
sage: Phi = M.continuous_map(N, ...

(continues on next page)
\[
\{ (u, v) \mapsto (x, y, z) = (2u/(u^2 + v^2 + 1), 2v/(u^2 + v^2 + 1), -(u^2 + v^2 - 1)/(u^2 + v^2 + 1)) \}
\]

The LaTeX output of that embedding is:

\[
\begin{array}{llcl}
\Phi: & S^2 & \longrightarrow & \mathbb{R}^3 \\
on U: & (x, y) & \mapsto & (X, Y, Z) = \left(\frac{2 \, x}{x^2 + y^2 + 1}, \frac{2 \, y}{x^2 + y^2 + 1}, \frac{x^2 + y^2 - 1}{x^2 + y^2 + 1}\right)
\end{array}
\]

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`):

\[
sage: Phi.display(c_xy)
\]

\[
Phi: S^2 \rightarrow \mathbb{R}^3 \\
on U: (x, y) \mapsto (X, Y, Z) = (2x/(x^2 + y^2 + 1), 2y/(x^2 + y^2 + 1), (x^2 + y^2 - 1)/(x^2 + y^2 + 1))
\]

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:

\[
sage: Phi.display(chart2=c_cart)
\]

\[
Phi: S^2 \rightarrow \mathbb{R}^3 \\
on U: (x, y) \mapsto (X, Y, Z) = (2x/(x^2 + y^2 + 1), 2y/(x^2 + y^2 + 1), (x^2 + y^2 - 1)/(x^2 + y^2 + 1))
\]

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:

\[
sage: Phi.display()
\]

\[
Phi: S^2 \rightarrow \mathbb{R}^3 \\
on U: (x, y) \mapsto (X, Y, Z) = (2x/(x^2 + y^2 + 1), 2y/(x^2 + y^2 + 1), (x^2 + y^2 - 1)/(x^2 + y^2 + 1))
\]

If a chart covers entirely the map’s domain, the mention “on …” is omitted:
A shortcut of \texttt{display()} is \texttt{disp()}: 

\begin{verbatim}
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))
\end{verbatim}

Display when SymPy is the symbolic engine:

\begin{verbatim}
sage: M.set_calculus_method('sympy')
sage: N.set_calculus_method('sympy')
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: latex(Phi.display(c_xy, c_cart))\begin{array}{llcl}
\Phi:& S^2 & \longrightarrow & \RR^3 \\
\text{on}
U : & \left(x, y\right) & \longmapsto & \left(X, Y, Z\right) = \\
& & & \left(\frac{2 \, x}{x^{2} + y^{2} + 1}, \\
& & & \frac{2 \, y}{x^{2} + y^{2} + 1}, \\
& & & \frac{x^{2} + y^{2} - 1}{x^{2} + y^{2} + 1}\right)
\end{array}
\end{verbatim}

\texttt{display}(chart1=None, chart2=None)

Display the expression of \texttt{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 \texttt{self}; if \texttt{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 \texttt{self}; if \texttt{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:

A simple reparametrization:

\begin{verbatim}
sage: R.<t> = manifolds.RealLine()
sage: I = R.open_interval(0, 2*pi)
sage: J = R.open_interval(2*pi, 6*pi)
sage: h = J.continuous_map(I, ((t -2*pi)/2,), name =h)
sage: h.display()
h: (2*pi, 6*pi) → (0, 2*pi)
\begin{array}{llcl}
t & \longmapsto & t = -pi + 1/2*t
\end{array}
sage: latex(h.display())\begin{array}{llcl}
\text{h:}& \left(2 \, \pi, 6 \, \pi\right) & \longrightarrow & \left(0, 2 \, \pi\right) \\
& t & \longmapsto & t = -\pi + \frac{1}{2} \, t
\end{array}
\end{verbatim}

Standard embedding of the sphere \(S^2\) in \(R^3\): 

1.7. Continuous Maps 229
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'\mathbb{R}^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')

The LaTeX output of that embedding is:

\begin{align*}
\Phi &: S^2 \rightarrow \mathbb{R}^3 \\
n_1 : & \quad (x, y) \mapsto (X, Y, Z) = \left(\frac{2x}{x^2 + y^2 + 1}, \frac{2y}{x^2 + y^2 + 1}, \frac{x^2 + y^2 - 1}{x^2 + y^2 + 1}\right) \\
n_2 : & \quad (u, v) \mapsto (X, Y, Z) = \left(\frac{2u}{u^2 + v^2 + 1}, \frac{2v}{u^2 + v^2 + 1}, \frac{-u^2 - v^2 - 1}{u^2 + v^2 + 1}\right)
\end{align*}

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):

\begin{align*}
\Phi &: S^2 \rightarrow \mathbb{R}^3 \\
on U : & \quad (x, y) \mapsto (X, Y, Z) = \left(\frac{2x}{x^2 + y^2 + 1}, \frac{2y}{x^2 + y^2 + 1}, \frac{x^2 + y^2 - 1}{x^2 + y^2 + 1}\right)
\end{align*}

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:

\begin{align*}
\Phi &: S^2 \rightarrow \mathbb{R}^3 \\
on U : & \quad (x, y) \mapsto (X, Y, Z) = \left(\frac{2x}{x^2 + y^2 + 1}, \frac{2y}{x^2 + y^2 + 1}, \frac{x^2 + y^2 - 1}{x^2 + y^2 + 1}\right)
\end{align*}

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:
If a chart covers entirely the map’s domain, the mention “on …” is omitted:

\[
\begin{align*}
\text{on U: } & (x, y) \mapsto (X, Y, Z) = (2x/(x^2 + y^2 + 1), 2y/(x^2 + y^2 + 1), (x^2 + y^2 - 1)/(x^2 + y^2 + 1)) \\
\text{on V: } & (u, v) \mapsto (X, Y, Z) = (2u/(u^2 + v^2 + 1), 2v/(u^2 + v^2 + 1), -(u^2 + v^2 - 1)/(u^2 + v^2 + 1))
\end{align*}
\]

A shortcut of \texttt{display()} is \texttt{disp()}:

\[
\begin{align*}
\text{sage: } & \Phi\text{.disp}() \\
\text{on U: } & (x, y) \mapsto (X, Y, Z) = (2x/(x^2 + y^2 + 1), 2y/(x^2 + y^2 + 1), (x^2 + y^2 - 1)/(x^2 + y^2 + 1))
\end{align*}
\]

Display when SymPy is the symbolic engine:

\[
\begin{align*}
\text{sage: } & M\text{.set\_calculus\_method('sympy')} \\
\text{sage: } & N\text{.set\_calculus\_method('sympy')} \\
\text{sage: } & \Phi\text{.display(c\_xy, c\_cart)} \\
\text{sage: } & \text{latex(\Phi\text{.display(c\_xy, c\_cart))}
\end{align*}
\]

\[
\begin{array}{llcl}
\Phi:& S^2 & \longrightarrow & \RR^3 \\
\text{on U : } & (x, y) & \longmapsto & \left(X, Y, Z\right) = \left(\frac{2 x}{x^{2} + y^{2} + 1}, \frac{2 y}{x^{2} + y^{2} + 1}, \frac{x^{2} + y^{2} - 1}{x^{2} + y^{2} + 1}\right)
\end{array}
\]

\textbf{expr (chart1=None, chart2=None)}

Return the expression of \texttt{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.

\textbf{INPUT:}

- \texttt{chart1} – (default: None) chart on the map’s domain; if None, the domain’s default chart is assumed
- \texttt{chart2} – (default: None) chart on the map’s codomain; if None, the codomain’s default chart is assumed

\textbf{OUTPUT:}

- symbolic expression representing the continuous map in the above two charts

\textbf{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 \to N
\quad (u, v) \mapsto (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
\ltypesage:symbolic.expression.Expression\gt
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)

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(rr:(0,+oo) ph:(0,2*pi): \phi)
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))
(continues on next page)
Check of the inverse coordinate transformation:

\[ r == r \ \text{*passed*} \]
\[ \phi == \arctan2(r \cdot \sin(\phi), r \cdot \cos(\phi)) \ \text{**failed**} \]
\[ x == x \ \text{*passed*} \]
\[ y == y \ \text{*passed*} \]

NB: a failed report can reflect a mere lack of simplification.

```
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
\quad (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
```

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)
```

(continues on next page)
A rotation in some Euclidean plane:

\[
\begin{align*}
\text{sage: } & M = \text{Manifold}(2, 'M', \text{structure}='\text{topological}') \quad \# \text{the plane (minus a segment to have global regular spherical coordinates)} \\
\text{sage: } & c_{\text{spher}}.<r,\phi> = M.\text{chart}(r:(0,+\infty) \ \phi:(0,2\pi)): \phi) \quad \# \text{spherical coordinates on the plane} \\
\text{sage: } & \text{rot} = M.\text{continuous_map}(M, (r, \phi+\pi/3), \text{name}='R') \quad \# \pi/3 \text{ rotation around r=0} \\
\text{sage: } & \text{rot}.\text{expr}() \\
& (r, 1/3*\pi + \phi)
\end{align*}
\]

Expression of the rotation in terms of Cartesian coordinates:

\[
\begin{align*}
\text{sage: } & c_{\text{cart}}.<x,y> = M.\text{chart()} \quad \# \text{Declaration of Cartesian coordinates} \\
\text{sage: } & \text{ch}_{\text{spher}}_{\text{cart}} = c_{\text{spher}}.\text{transition_map}(c_{\text{cart}}, \\
& \quad \ldots:\ [r*\cos(\phi), r*\sin(\phi)]) \quad \# \text{relation to spherical coordinates} \\
\text{sage: } & \text{ch}_{\text{spher}}_{\text{cart}}.\text{set_inverse}(\sqrt{x^2+y^2}, \arctan2(y,x)) \\
\end{align*}
\]

Check of the inverse coordinate transformation:

- \( r == r \text{ *passed*} \\
- \( \phi == \arctan2(r*\sin(\phi), r*\cos(\phi)) \text{ **failed**} \\
- \( x == x \text{ *passed*} \\
- \( y == y \text{ *passed*} \\
\)

NB: a failed report can reflect a mere lack of simplification.

\[
\text{sage: } \text{rot}.\text{expr}(c_{\text{cart}}, c_{\text{cart}}) \\
& (-1/2*\sqrt{3}*y + 1/2*x, 1/2*\sqrt{3}*x + 1/2*y)
\]

**image (subset=None, inverse=None)**

Return the image of self or the image of subset under self.

**INPUT:**

- inverse – (default: None) continuous map from map.codomain() to map.domain(), which once restricted to the image of \( \Phi \) is the inverse of \( \Phi \) onto its image if the latter exists (NB: no check of this is performed)
- subset – (default: the domain of map) a subset of the domain of self

**EXAMPLES:**

\[
\begin{align*}
\text{sage: } & M = \text{Manifold}(2, 'M', \text{structure}='\text{topological}') \\
\text{sage: } & N = \text{Manifold}(1, 'N', \text{ambient}=M, \text{structure}='\text{topological}') \\
\text{sage: } & \text{CM.<x,y> = M.\text{chart()}} \\
\text{sage: } & \text{CN.<u> = N.\text{chart(coord_restrictions=\text{lambda} u: [u > -1, u < 1])}} \\
\text{sage: } & \text{Phi} = N.\text{continuous_map}(M, \{(CN,CM): [u, u^2]\}, \text{name}='\text{Phi}') \\
\text{sage: } & \text{Phi.image()} \\
& \text{Image of the Continuous map Phi}
\end{align*}
\]
from the 1-dimensional topological submanifold $N$ immersed in the 2-dimensional topological manifold $M$

to the 2-dimensional topological manifold $M$

```
sage: S = N.subset('S')
```
```
sage: Phi_S = Phi.image(S); Phi_S
Image of the Subset $S$ of the
1-dimensional topological submanifold $N$
immersed in the 2-dimensional topological manifold $M$
under the Continuous map $\Phi$
from the 1-dimensional topological submanifold $N$
immersed in the 2-dimensional topological manifold $M$
to the 2-dimensional topological manifold $M$
```
```
sage: Phi_S.is_subset(M)
True
```

**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'\mathbb{R}^2', structure='topological')
sage: c_cart.<x,y> = M.chart()
```

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()
$R^{-1}: \mathbb{R}^2 \to \mathbb{R}^2$
$(x, y) \mapsto (1/2*sqrt(3)*y + 1/2*x, -1/2*sqrt(3)*x + 1/2*y)$
```

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 $^\text{(-1)}$ or $\sim$ can also be used for the inverse:

```
sage: rot^(-1) is rot.inverse()
True
```
(continues on next page)
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: 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:
The identity map is handled specially:

```python
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: M.identity_map().preimage(M)
2-dimensional topological manifold M
sage: M.identity_map().preimage(M) is M
True
```

Another trivial case:

```python
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: Q1 = R2.open_subset('Q1', coord_def={c_xy: [x>0, y>0]}); Q1
Open subset Q1 of the 2-dimensional topological manifold R^2
sage: Phi_inv_Q1 = Phi.preimage(Q1); Phi_inv_Q1
Subset Phi_inv_Q1 of the 1-dimensional topological manifold R
sage: R.point([pi/4]) in Phi_inv_Q1
True
sage: R.point([0]) in Phi_inv_Q1
False
sage: R.point([3*pi/4]) in Phi_inv_Q1
False
```
sage: M = Manifold(2, 'M', structure='topological')
    sage: X.<x,y> = M.chart()
    sage: D1 = M.open_subset('D1', coord_def={X: x^2+y^2<1})  # the open unit disk
    sage: D2 = M.open_subset('D2', coord_def={X: x^2+y^2<4})
    sage: f = Hom(D1,D2)((X.restrict(D1), X.restrict(D2)): (2*x, 2*y), name='f')
    sage: f.preimage(D2)
    Open subset D1 of the 2-dimensional topological manifold M
    sage: f.preimage(M)
    Open subset D1 of the 2-dimensional topological manifold M

**pullback** (codomain_subset, name=None, latex_name=None)

Return the preimage of codomain_subset under self.

An alias is pullback().

**INPUT:**

- codomain_subset – an instance of ManifoldSubset
- 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

**OUTPUT:**

- either a TopologicalManifold or a ManifoldSubsetPullback

**EXAMPLES:**

```python
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: Q1 = R2.open_subset('Q1', coord_def={c_xy: [x>0, y>0]}); Q1
Open subset Q1 of the 2-dimensional topological manifold R^2
sage: Phi_inv_Q1 = Phi.preimage(Q1); Phi_inv_Q1
Subset Phi_inv_Q1 of the 1-dimensional topological manifold R
sage: R.point([pi/4]) in Phi_inv_Q1
    True
sage: R.point([0]) in Phi_inv_Q1
    False
sage: R.point([3*pi/4]) in Phi_inv_Q1
    False
```

The identity map is handled specially:

```python
sage: M = Manifold(2, 'M', structure='topological')
    sage: X.<x,y> = M.chart()
    sage: M.identity_map().preimage(M)
    2-dimensional topological manifold M
    sage: M.identity_map().preimage(M) is M
    True
```

Another trivial case:
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: D1 = M.open_subset('D1', coord_def={X: x^2+y^2<1})  # the open unit disk
sage: D2 = M.open_subset('D2', coord_def={X: x^2+y^2<4})
sage: f = Hom(D1,D2)((X.restrict(D1), X.restrict(D2)), (2*x, 2*y), name='f')
sage: f.preimage(D2)
Open subset D1 of the 2-dimensional topological manifold M
sage: f.preimage(M)
Open subset D1 of the 2-dimensional topological manifold M

**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/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/<\( r < 2 \)
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 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/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'\mathbb{R}^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(rr:(0,+oo) ph:(0,2*pi): \phi) # spherical...
˓→coordinates on U
```

Links between spherical coordinates and Cartesian ones:

```
sage: ch_cart_spher = c_cart.transition_map(c_spher,
˓→\[sqrt(x^2+y^2), \atan2(y,x)\])
sage: ch_cart_spher.set_inverse(r*cos(ph), r*sin(ph))
Check of the inverse coordinate transformation:
\[
\begin{align*}
x &= x \quad \text{\texttt{\*passed\*}} \\
y &= y \quad \text{\texttt{\*passed\*}}
\end{align*}
\]
```
\[ r == r \quad \text{`passed`} \]
\[ \text{ph} == \arctan2(r\sin(\text{ph}), r\cos(\text{ph})) \quad \text{`failed`} \]

NB: a failed report can reflect a mere lack of simplification.

```
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:
Links between spherical coordinates and Cartesian ones:

```
sage: ch_cart_spher = c_cart.transition_map(c_spher,
....:       [sqrt(x^2+y^2), atan2(y,x)])
sage: ch_cart_spher.set_inverse(r*cos(ph), r*sin(ph))
Check of the inverse coordinate transformation:
  x == x *passed*
  y == y *passed*
  r == r *passed*
  ph == arctan2(r*sin(ph), r*cos(ph)) **failed**
NB: a failed report can reflect a mere lack of simplification.
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, (r, ph))): 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))}
```
1.7.3 Images of Manifold Subsets under Continuous Maps as Subsets of the Codomain

*ImageManifoldSubset* implements the image of a continuous map $\Phi$ from a manifold $M$ to some manifold $N$ as a subset $\Phi(M)$ of $N$, or more generally, the image $\Phi(S)$ of a subset $S \subseteq M$ as a subset of $N$.

```python
class sage.manifolds.continuous_map_image.ImageManifoldSubset
    (map, inverse=None,
     name=None,
     latex_name=None,
     domain_subset=None)
```

**Bases:** *ManifoldSubset*

Subset of a topological manifold that is a continuous image of a manifold subset.

**INPUT:**

- `map` – continuous map $\Phi$
- `inverse` – (default: None) continuous map from map.codomain() to map.domain(), which once restricted to the image of $\Phi$ is the inverse of $\Phi$ onto its image if the latter exists (NB: no check of this is performed)
- `name` – (default: computed from the names of the map and the subset)
  string: name (symbol) given to the subset
- `latex_name` – (default: None) string: LaTeX symbol to denote the subset; if none is provided, it is set to name
- `domain_subset` – (default: the domain of map) a subset of the domain of map

### 1.8 Submanifolds of topological manifolds

Given a topological manifold $M$ over a topological field $K$, a *topological submanifold of* $M$ is defined by a topological manifold $N$ over the same field $K$ of dimension lower than the dimension of $M$ and a topological embedding $\phi$ from $N$ to $M$ (i.e. $\phi$ is a homeomorphism onto its image).

In the case where the map $\phi$ is only an embedding locally, it is called an *topological immersion*, and defines an *immersed submanifold*.

The global embedding property cannot be checked in sage, so the immersed or embedded aspect of the manifold must be declared by the user, by calling either `set_embedding()` or `set_immersion()` while declaring the map $\phi$.

The map $\phi : N \to M$ can also depend on one or multiple parameters. As long as $\phi$ remains injective in these parameters, it represents a *foliation*. The *dimension* of the foliation is defined as the number of parameters.

**AUTHORS:**

- Florentin Jaffredo (2018): initial version
- Matthias Koeppe (2021): open subsets of submanifolds

**REFERENCES:**

- J. M. Lee: *Introduction to Smooth Manifolds* [Lee2013]
class sage.manifolds.topological_submanifold.TopologicalSubmanifold(n, name, field, structure, ambient=None, base_manifold=None, latex_name=None, start_index=0, category=None, unique_tag=None)

Bases: TopologicalManifold

Submanifold of a topological manifold.

Given a topological manifold $M$ over a topological field $K$, a topological submanifold of $M$ is defined by a topological manifold $N$ over the same field $K$ of dimension lower than the dimension of $M$ and a topological embedding $\phi$ from $N$ to $M$ (i.e. $\phi$ is an homeomorphism onto its image).

In the case where $\phi$ is only a topological immersion (i.e. is only locally an embedding), one says that $N$ is an immersed submanifold.

The map $\phi$ can also depend on one or multiple parameters. As long as $\phi$ remains injective in these parameters, it represents a foliation. The dimension of the foliation is defined as the number of parameters.

INPUT:

- $n$ – positive integer; dimension of the submanifold
- name – string; name (symbol) given to the submanifold
- field – field $K$ on which the submanifold is defined; allowed values are
  - 'real' or an object of type RealField (e.g., RR) for a manifold over $\mathbb{R}$
  - 'complex' or an object of type ComplexField (e.g., CC) for a manifold over $\mathbb{C}$
  - 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) codomain $M$ of the immersion $\phi$; must be a topological manifold. If None, it is set to self
- base_manifold – (default: None) if not None, must be a topological manifold; the created object is then an open subset of base_manifold
- latex_name – (default: None) string; LaTeX symbol to denote the submanifold; 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 submanifold, 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 via TopologicalManifold would return the previously constructed object corresponding to these arguments)
EXAMPLES:

Let $N$ be a 2-dimensional submanifold of a 3-dimensional manifold $M$:

```
sage: M = Manifold(3, 'M', structure="topological")
sage: N = Manifold(2, 'N', ambient=M, structure="topological")
sage: N
2-dimensional topological submanifold N immersed in the 3-dimensional topological manifold M
sage: CM.<x,y,z> = M.chart()
sage: CN.<u,v> = N.chart()
```

Let us define a 1-dimensional foliation indexed by $t$:

```
sage: t = var('t')
sage: phi = N.continuous_map(M, {(CN,CM): [u, v, t + u^2 + v^2]})
sage: phi.display()
N → M
(u, v) ↦ (x, y, z) = (u, v, u^2 + v^2 + t)
```

The foliation inverse maps are needed for computing the adapted chart on the ambient manifold:

```
sage: phi_inv = M.continuous_map(N, {(CM, CN): [x, y]})
sage: phi_inv.display()
M → N
(x, y, z) ↦ (u, v) = (x, y)
sage: phi_inv_t = M.scalar_field({CM: z-x^2-y^2})
sage: phi_inv_t.display()
M → ℝ
(x, y, z) ↦ -x^2 - y^2 + z
```

$\phi$ can then be declared as an embedding $N \rightarrow M$:

```
sage: N.set_embedding(phi, inverse=phi_inv, var=t,...:
....:       t_inverse={t: phi_inv_t})
```

The foliation can also be used to find new charts on the ambient manifold that are adapted to the foliation, i.e. in which the expression of the immersion is trivial. At the same time, the appropriate coordinate changes are computed:

```
sage: N.adapted_chart()
[Chart (M, (u_M, v_M, t_M))]  
sage: M.atlas()  
[Chart (M, (x, y, z)), Chart (M, (u_M, v_M, t_M))]  
sage: len(M.coord_changes())
2
```

The foliation parameters are always added as the last coordinates.

See also:

- `manifold`  
- `adapted_chart` (*postscript=None, latex_postscript=None*)

Create charts and changes of charts in the ambient manifold adapted to the foliation.

A manifold $M$ of dimension $m$ can be foliated by submanifolds $N$ of dimension $n$. The corresponding embedding needs $m - n$ free parameters to describe the whole manifold.
A chart adapted to the foliation is a set of coordinates \((x_1, \ldots, x_n, t_1, \ldots, t_{m-n})\) on \(M\) such that \((x_1, \ldots, x_n)\) are coordinates on \(N\) and \((t_1, \ldots, t_{m-n})\) are the \(m-n\) free parameters of the foliation.

Provided that an embedding with free variables is already defined, this function constructs such charts and coordinates changes whenever it is possible.

If there are restrictions of the coordinates on the starting chart, these restrictions are also propagated.

**INPUT:**

- `postscript` – (default: `None`) string defining the name of the coordinates of the adapted chart. This string will be appended to the names of the coordinates \((x_1, \ldots, x_n)\) and of the parameters \((t_1, \ldots, t_{m-n})\). If `None`, `"_" + self.ambient()._name` is used.

- `latex_postscript` – (default: `None`) string defining the LaTeX name of the coordinates of the adapted chart. This string will be appended to the LaTeX names of the coordinates \((x_1, \ldots, x_n)\) and of the parameters \((t_1, \ldots, t_{m-n})\). If `None`, `"_" + self.ambient()._latex_()` is used.

**OUTPUT:**

- list of adapted charts on \(M\) created from the charts of `self`.

**EXAMPLES:**

```sage
M = Manifold(3, 'M', structure='topological',
             latex_name=r'\mathcal{M}')
N = Manifold(2, 'N', ambient=M, structure='topological')
CM.<x,y,z> = M.chart()
CN.<u,v> = N.chart()
t = var('t')
phi = N.continuous_map(M, {(CN,CM): [u,v,t+t^2+v^2]})
phi_inv = M.continuous_map(N, {(CM, CN): [x,y]})
phi_inv_t = M.scalar_field({CM: z-x^2-y^2})
N.set_embedding(phi, inverse=phi_inv, var=t,
                t_inverse={t:phi_inv_t})
N.adapted_chart()
```

The adapted chart has been added to the atlas of \(M\):

```sage
M.atlas()
```

```sage
[Chart (M, (x, y, z)), Chart (M, (u_M, v_M, t_M))]
```

```sage
N.atlas()
```

```sage
[Chart (N, (u, v))]
```

The names of the adapted coordinates can be customized:

```sage
N.adapted_chart(postscript='1', latex_postscript='1')
```

```sage
M.ambient()()
```

`ambient()`

Return the manifold in which `self` is immersed or embedded.
EXAMPLES:

```python
sage: M = Manifold(3, 'M', structure="topological")
sage: N = Manifold(2, 'N', ambient=M, structure="topological")
sage: N.ambient()
3-dimensional topological manifold M
```

**as_subset()**

Return `self` as a subset of the ambient manifold.

self must be an embedded submanifold.

EXAMPLES:

```python
sage: M = Manifold(2, 'M', structure="topological")
sage: N = Manifold(1, 'N', ambient=M, structure="topological")
sage: CM.<x,y> = M.chart()
sage: CN.<u> = N.chart(coord_restrictions=lambda u: [u > -1, u < 1])
sage: phi = N.continuous_map(M, {(CN,CM): [u, u^2]})
sage: N.set_embedding(phi)
sage: N
1-dimensional topological submanifold N
embedded in the 2-dimensional topological manifold M
```

**declare_embedding()**

Declare that the immersion provided by set_immersion() is in fact an embedding.

A topological embedding is a continuous map that is a homeomorphism onto its image. A differentiable embedding is a topological embedding that is also a differentiable immersion.

EXAMPLES:

```python
sage: M = Manifold(3, 'M', structure="topological")
sage: N = Manifold(2, 'N', ambient=M, structure="topological")
sage: N
2-dimensional topological submanifold N immersed in the
3-dimensional topological manifold M
```

```python
sage: CM.<x,y,z> = M.chart()
sage: CN.<u,v> = N.chart()
sage: t = var(t)
sage: phi = N.continuous_map(M, {(CN,CM): [u,v,t+u^2+v^2]})
sage: phi_inv = M.continuous_map(N, {(CM,CN): [x,y]})
sage: phi_inv_t = M.scalar_field({CM: z-x^2-y^2})
sage: N.set_immersion(phi, inverse=phi_inv, var=t,
....: t_inverse={t: phi_inv_t})
sage: N._immersed
True
sage: N._embedded
False
```
embedding()  
Return the embedding of self into the ambient manifold.

EXAMPLES:

```python
sage: M = Manifold(3, 'M', structure="topological")
sage: N = Manifold(2, 'N', ambient=M, structure="topological")
sage: CM.<x,y,z> = M.chart()
sage: CN.<u,v> = N.chart()
sage: t = var('t')
sage: phi = N.continuous_map(M, {(CN,CM): [u,v,t+u^2+v^2]})
sage: phi_inv = M.continuous_map(N, {(CM,CN): [x,y]})
sage: phi_inv_t = M.scalar_field({CM: z-x^2-y^2})
sage: N.set_embedding(phi, inverse=phi_inv, var=t, 
.....: t_inverse={t: phi_inv_t})
sage: N.embedding()
Continuous map from the 2-dimensional topological submanifold N
embedded in the 3-dimensional topological manifold M to the
3-dimensional topological manifold M
```

immersion()  
Return the immersion of self into the ambient manifold.

EXAMPLES:

```python
sage: M = Manifold(3, 'M', structure="topological")
sage: N = Manifold(2, 'N', ambient=M, structure="topological")
sage: CM.<x,y,z> = M.chart()
sage: CN.<u,v> = N.chart()
sage: t = var('t')
sage: phi = N.continuous_map(M, {(CN,CM): [u,v,t+u^2+v^2]})
sage: phi_inv = M.continuous_map(N, {(CM,CN): [x,y]})
sage: phi_inv_t = M.scalar_field({CM: z-x^2-y^2})
sage: N.set_immersion(phi, inverse=phi_inv, var=t, 
.....: t_inverse={t: phi_inv_t})
sage: N.immersion()
Continuous map from the 2-dimensional topological submanifold N
immersed in the 3-dimensional topological manifold M to the
3-dimensional topological manifold M
```

open_subset (name, latex_name=None, coord_def={}, supersets=None)  
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.

As self is a submanifold of its ambient manifold, the new open subset is also considered a submanifold of that. Hence the returned object is an instance of TopologicalSubmanifold.

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
• supersets – (default: only self) list of sets that the new open subset is a subset of
OUTPUT:

• the open subset, as an instance of `TopologicalSubmanifold`

EXAMPLES:

```sage
sage: M = Manifold(3, 'M', structure="topological")
sage: N = Manifold(2, 'N', ambient=M, structure="topological"); N
2-dimensional topological submanifold N immersed in the
3-dimensional topological manifold M
sage: S = N.subset('S'); S
Subset S of the
2-dimensional topological submanifold N immersed in the
3-dimensional topological manifold M
sage: O = N.subset('O', is_open=True); O  # indirect doctest
Open subset O of the
2-dimensional topological submanifold N immersed in the
3-dimensional topological manifold M
sage: phi = N.continuous_map(M)
sage: N.set_embedding(phi)
sage: N
2-dimensional topological submanifold N embedded in the
3-dimensional topological manifold M
sage: S = N.subset('S'); S
Subset S of the
2-dimensional topological submanifold N embedded in the
3-dimensional topological manifold M
sage: O = N.subset('O', is_open=True); O  # indirect doctest
Open subset O of the
2-dimensional topological submanifold N embedded in the
3-dimensional topological manifold M
```

`plot (param, u, v, chart1=None, chart2=None, **kwargs)`

Plot an embedding.

Plot the embedding defined by the foliation and a set of values for the free parameters. This function can only plot 2-dimensional surfaces embedded in 3-dimensional manifolds. It ultimately calls `ParametricSurface`.

INPUT:

• `param` – dictionary of values indexed by the free variables appearing in the foliation.
• `u` – iterable of the values taken by the first coordinate of the surface to plot
• `v` – iterable of the values taken by the second coordinate of the surface to plot
• `chart1` – (default: `None`) chart in which `u` and `v` are considered. By default, the default chart of the submanifold is used
• `chart2` – (default: `None`) chart in the codomain of the embedding. By default, the default chart of the codomain is used
• `**kwargs` – other arguments as used in `ParametricSurface`

EXAMPLES:

```sage
sage: M = Manifold(3, 'M', structure="topological")
sage: N = Manifold(2, 'N', ambient=M, structure="topological")
sage: CM.<x,y,z> = M.chart()
```

(continues on next page)
sage: CN.<u,v> = N.chart()
sage: t = var('t')
sage: phi = N.continuous_map(M, {(CN,CM): [u,v,t+u^2+v^2]})
sage: phi_inv = M.continuous_map(N, {(CM,CN): [x,y]})
sage: phi_inv_t = M.scalar_field({CM: z-x^2-y^2})
sage: N.set_embedding(phi, inverse=phi_inv, var=t,
                 t_inverse={t:phi_inv_t})

sage: N.adapted_chart()

[Chart (M, (u_M, v_M, t_M))]

sage: P0 = N.plot({t:0}, srange(-1, 1, 0.1), srange(-1, 1, 0.1),
               CN, CM, opacity=0.3, mesh=True)
sage: P1 = N.plot({t:1}, srange(-1, 1, 0.1), srange(-1, 1, 0.1),
               CN, CM, opacity=0.3, mesh=True)
sage: P2 = N.plot({t:2}, srange(-1, 1, 0.1), srange(-1, 1, 0.1),
               CN, CM, opacity=0.3, mesh=True)
sage: P3 = N.plot({t:3}, srange(-1, 1, 0.1), srange(-1, 1, 0.1),
               CN, CM, opacity=0.3, mesh=True)
sage: P0 + P1 + P2 + P3

Graphics3d Object

See also:

ParametricSurface

set_embedding (phi, inverse=None, var=None, t_inverse=None)
Register the embedding of an embedded submanifold.

A **topological embedding** is a continuous map that is a homeomorphism onto its image. A **differentiable embedding** is a topological embedding that is also a differentiable immersion.

**INPUT:**

- `phi` – continuous map \( \phi \) from `self` to `self.ambient()` (\( \text{self} \) is the manifold where the embedding is performed, and `self.ambient()` refers to the ambient manifold of the manifold `self`).
- `inverse` – (default: None) continuous map from `self.ambient()` to `self`, which once restricted to the image of `phi` is the inverse of `phi` onto its image (NB: no check of this is performed).
- `var` – (default: None) list of parameters involved in the definition of `phi` (case of foliation); if `phi` depends on a single parameter `t`, one can write `var=t` as a shortcut for `var=[t]`
- `t_inverse` – (default: None) dictionary of scalar fields on `self.ambient()` providing the values of the parameters involved in the definition of `phi` (case of foliation), the keys being the parameters.

**EXAMPLES:**

```python
sage: M = Manifold(3, 'M', structure="topological")
sage: N = Manifold(2, 'N', ambient=M, structure="topological")
sage: N
2-dimensional topological submanifold N immersed in the
3-dimensional topological manifold M
sage: CM.<x,y,z> = M.chart()
sage: CN.<u,v> = N.chart()
sage: t = var('t')
sage: phi = N.continuous_map(M, {(CN,CM): [u,v,t+u^2+v^2]})
sage: phi.display()
N → M
(u, v) ↦ (x, y, z) = (u, v, u^2 + v^2 + t)
sage: phi_inv = M.continuous_map(N, {(CM,CN): [x,y]})
sage: phi_inv.display()
M → N
(x, y, z) ↦ (u, v) = (x, y)
sage: phi_inv_t = M.scalar_field({CM: z-x^2-y^2})
sage: phi_inv_t.display()
M → ℝ
(x, y, z) ↦ -x^2 - y^2 + z
sage: N.set_embedding(phi, inverse=phi_inv, var=t, ....: t_inverse={t: phi_inv_t})
```

Now `N` appears as an embedded submanifold:

```python
sage: N
2-dimensional topological submanifold N embedded in the
3-dimensional topological manifold M
```

**set_immersion**( `phi`, `inverse=None`, `var=None`, `t_inverse=None` )

Register the immersion of the immersed submanifold.

A **topological immersion** is a continuous map that is locally a topological embedding (i.e. a homeomorphism onto its image). A **differentiable immersion** is a differentiable map whose differential is injective at each point.

If an inverse of the immersion onto its image exists, it can be registered at the same time. If the immersion depends on parameters, they must also be declared here.

**INPUT:**

- `phi` – continuous map \( \phi \) from `self` to `self.ambient()`
• inverse – (default: None) continuous map from self.ambient() to self, which once restricted to the image of \( \phi \) is the inverse of \( \phi \) onto its image if the latter exists (NB: no check of this is performed)

• var – (default: None) list of parameters involved in the definition of \( \phi \) (case of foliation); if \( \phi \) depends on a single parameter \( t \), one can write var=t as a shortcut for var=[t]

• t_inverse – (default: None) dictionary of scalar fields on self.ambient() providing the values of the parameters involved in the definition of \( \phi \) (case of foliation), the keys being the parameters

EXAMPLES:

```python
sage: M = Manifold(3, 'M', structure="topological")
sage: N = Manifold(2, 'N', ambient=M, structure="topological")
sage: N
2-dimensional topological submanifold N immersed in the
3-dimensional topological manifold M
sage: CM.<x,y,z> = M.chart()
sage: CN.<u,v> = N.chart()
sage: t = var('t')
sage: phi = N.continuous_map(M, {(CN,CM): [u,v,t+u^2+v^2]})
sage: phi.display()
N → M
  (u, v) ↦ (x, y, z) = (u, v, u^2 + v^2 + t)
sage: phi_inv = M.continuous_map(N, {(CM,CN): [x,y]})
sage: phi_inv.display()
M → N
  (x, y, z) ↦ (u, v) = (x, y)
sage: phi_inv_t = M.scalar_field({CM: z-x^2-y^2})
sage: phi_inv_t.display()
M → ℝ
  (x, y, z) ↦ -x^2 - y^2 + z
sage: N.set_immersion(phi, inverse=phi_inv, var=t, ...
t_inverse={t: phi_inv_t})
```

1.9 Topological Vector Bundles

1.9.1 Topological Vector Bundle

Let \( K \) be a topological field. A vector bundle of rank \( n \) over the field \( K \) and over a topological manifold \( B \) (base space) is a topological manifold \( E \) (total space) together with a continuous and surjective map \( \pi : E \to B \) such that for every point \( p \in B \), we have:

• the set \( E_p = \pi^{-1}(p) \) has the vector space structure of \( K^n \),

• there is a neighborhood \( U \subset B \) of \( p \) and a homeomorphism (trivialization) \( \varphi : \pi^{-1}(p) \to U \times K^n \) such that \( \varphi \) is compatible with the fibers, namely \( \pi \circ \varphi^{-1} = \text{pr}_1 \), and \( v \mapsto \varphi^{-1}(q,v) \) is a linear isomorphism between \( K^n \) and \( E_q \) for any \( q \in U \).

AUTHORS:

• Michael Jung (2019) : initial version

REFERENCES:

• [Lee2013]
• [Mil1974]
An instance of this class is a topological vector bundle \( E \to B \) over a topological field \( K \).

**INPUT:**

- **rank** – positive integer; rank of the vector bundle
- **name** – string representation given to the total space
- **base_space** – the base space (topological manifold) over which the vector bundle is defined
- **field** – field \( K \) which gives the fibers the structure of a vector space over \( K \); allowed values are
  - 'real' or an object of type `RealField` (e.g., `RR`) for a vector bundle over \( \mathbb{R} \)
  - 'complex' or an object of type `ComplexField` (e.g., `CC`) for a vector bundle over \( \mathbb{C} \)
  - an object in the category of topological fields (see `Fields` and `TopologicalSpaces`) for other types of topological fields
- **latex_name** – (default: `None`) LaTeX representation given to the total space
- **category** – (default: `None`) to specify the category; if `None`, `VectorBundles(base_space, c_field)` is assumed (see the category `VectorBundles`)
- **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 would return the previously constructed object corresponding to these arguments)

**EXAMPLES:**

A real line bundle over some 4-dimensional topological manifold:

```python
sage: M = Manifold(4, 'M', structure='top')
sage: E = M.vector_bundle(1, 'E'); E
Topological real vector bundle E -> M of rank 1 over the base space
4-dimensional topological manifold M
sage: E.base_space()
4-dimensional topological manifold M
sage: E.base_ring()
Real Field with 53 bits of precision
sage: E.rank()
1
```

For a more sophisticated example, let us define a non-trivial 2-manifold to work with:

```python
sage: M = Manifold(2, 'M', structure='top')
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: E = M.vector_bundle(2, 'E'); E
```

(continues on next page)
Now, there are two ways to go. Most effortlessly, we define trivializations similar to charts (see Trivialization):

```python
sage: phi_U = E.trivialization('phi_U', domain=U); phi_U
Trivialization (phi_U, E|_U)
```

```python
sage: phi_V = E.trivialization('phi_V', domain=V); phi_V
Trivialization (phi_V, E|_V)
```

```python
sage: transf = phi_U.transition_map(phi_V, [[0,x],[x,0]]) # transition map...
˓→between trivializations
```

```python
sage: fU = phi_U.frame(); fU
Trivialization frame (E|_U, ((phi_U^*e_1),(phi_U^*e_2)))
```

```python
sage: fV = phi_V.frame(); fV
Trivialization frame (E|_V, ((phi_V^*e_1),(phi_V^*e_2)))
```

Then, the atlas of \( E \) consists of all known trivializations defined on \( E \):

```python
sage: E.atlas() # a shallow copy of the atlas
[Trivialization (phi_U, E|_U), Trivialization (phi_V, E|_V)]
```

Or we just define frames, an automorphism on the free section module over the intersection domain \( W \) and declare the change of frame manually (for more details consult LocalFrame):

```python
sage: eU = E.local_frame('eU', domain=U); eU
Local frame (E|_U, (eU_0,eU_1))
```

```python
sage: eUW = eU.restrict(W) # to trivialize E|_W
```

```python
sage: eV = E.local_frame('eV', domain=V); eV
Local frame (E|_V, (eV_0,eV_1))
```

```python
sage: eVW = eV.restrict(W)
```

```python
sage: a = E.section_module(domain=W).automorphism(); a
Automorphism of the Free module C^0(W;E) of sections on the Open subset W of the 2-dimensional topological manifold M with values in the real vector bundle E of rank 2
```

```python
sage: a[eUW, :] = [[0,x],[x,0]]
```

```python
sage: E.set_change_of_frame(eUW, eVW, a)
```

Now, the list of all known frames defined on \( E \) can be displayed via frames():

```python
sage: E.frames() # a shallow copy of all known frames on E
[Trivialization frame (E|_U, ((phi_U^*e_1),(phi_U^*e_2)))
(continues on next page)
By definition \( E \) is a manifold, in this case of dimension 4 (notice that the induced charts are not implemented, yet):

```python
sage: E.total_space()
4-dimensional topological manifold E
```

The method `section()` returns a section while the method `section_module()` returns the section module on the corresponding domain:

```python
sage: s = E.section(name='s'); s
Section s on the 2-dimensional topological manifold M with values in the real vector bundle E of rank 2
sage: s in E.section_module()
True
```

`atlas()`

Return the list of trivializations that have been defined for `self`.

**EXAMPLES:**

```python
sage: M = Manifold(3, 'M')
sage: U = M.open_subset('U')
sage: V = M.open_subset('V')
sage: E = M.vector_bundle(2, 'E')
sage: phi_U = E.trivialization('phi_U', domain = U)
sage: phi_V = E.trivialization('phi_V', domain = V)
sage: phi_M = E.trivialization('phi_M')
sage: E.atlas()
[Trivialization (phi_U, E|_U),
 Trivialization (phi_V, E|_V),
 Trivialization (phi_M, E|_M)]
```

`base_field()`

Return the field on which the fibers are defined.

**OUTPUT:**

- a topological field

**EXAMPLES:**

```python
sage: M = Manifold(3, 'M', structure='topological')
sage: E = M.vector_bundle(2, 'E', field=CC)
sage: E.base_field()
Complex Field with 53 bits of precision
```

`base_field_type()`

Return the type of topological field on which the fibers are defined.

**OUTPUT:**
• a string describing the field, with three possible values:

- 'real' for the real field $\mathbb{R}$
- 'complex' for the complex field $\mathbb{C}$
- 'neither_real_nor_complex' for a field different from $\mathbb{R}$ and $\mathbb{C}$

EXAMPLES:

```sage
sage: M = Manifold(2, 'M', structure='top')
sage: E = M.vector_bundle(2, 'E', field='CC')
sage: E.base_field_type()
'complex'
```

**base_space()**

Return the base space of the vector bundle.

EXAMPLES:

```sage
sage: M = Manifold(2, 'M', structure='top')
sage: E = M.vector_bundle(2, 'E')
sage: E.base_space()
2-dimensional topological manifold M
```

**change_of_frame(frame1, frame2)**

Return a change of local frames defined on self.

INPUT:

- frame1 – local frame 1
- frame2 – local frame 2

OUTPUT:

- a `FreeModuleAutomorphism` representing, at each point, the vector space automorphism $P$ that relates frame 1, $(e_i)$ say, to frame 2, $(f_i)$ say, according to $f_i = P(e_i)$

EXAMPLES:

```sage
sage: M = Manifold(3, 'M', structure='top')
sage: X.<x,y,z> = M.chart()
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame(e)
sage: a = E.section_module().automorphism() # Now, the section module is free
sage: a[:] = [[sqrt(3)/2, -1/2], [1/2, sqrt(3)/2]]
sage: f = e.new_frame(a, 'f')
sage: E.change_of_frame(e, f)
Automorphism of the Free module $C^0(M;E)$ of sections on the
3-dimensional topological manifold M with values in the real vector
bundle E of rank 2
sage: a == E.change_of_frame(e, f)
True
sage: a.inverse() == E.change_of_frame(f, e)
True
```

**changes_of_frame()**

Return all the changes of local frames defined on self.

OUTPUT:
• dictionary of vector bundle automorphisms representing the changes of frames, the keys being the pair of frames

EXAMPLES:

```python
sage: M = Manifold(3, 'M', structure='top')
sage: c_xyz.<x,y,z> = M.chart()
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e'); e
Local frame (E|_M, (e_0,e_1))
sage: auto_group = E.section_module().general_linear_group()
sage: e_to_f = auto_group([[0,1],[1,0]]); e_to_f
Automorphism of the Free module C^0(M;E) of sections on the 3-dimensional topological manifold M with values in the real vector bundle E of rank 2
sage: f_in_e = auto_group([[0,1],[1,0]])
sage: f = e.new_frame(f_in_e, 'f'); f
Local frame (E|_M, (f_0,f_1))
sage: E.changes_of_frame()
# random
{(Local frame (E|_M, (f_0,f_1)), Local frame (E|_M, (e_0,e_1))): Automorphism of the Free module C^0(M;E) of sections on the 3-dimensional topological manifold M with values in the real vector bundle E of rank 2,
 (Local frame (E|_M, (e_0,e_1)), Local frame (E|_M, (f_0,f_1))): Automorphism of the Free module C^0(M;E) of sections on the 3-dimensional topological manifold M with values in the real vector bundle E of rank 2}
```

coframes()

Return the list of coframes defined on self.

OUTPUT:

• list of coframes defined on self

EXAMPLES:

```python
sage: M = Manifold(3, 'M', structure='top')
sage: E = M.vector_bundle(2, 'E')
sage: U = M.open_subset('U'); V = M.open_subset('V')
sage: phi_U = E.trivialization(phi_U, domain=U)
sage: e = E.local_frame('e', domain=V)
sage: E.coframes()
[Trivialization coframe (E|_U, ((phi_U^*e^1),(phi_U^*e^2))),
 Local coframe (E|_V, (e^0,e^1))]
```

default_frame()

Return the default frame of on self.

OUTPUT:

• a local frame as an instance of LocalFrame

EXAMPLES:

```python
sage: M = Manifold(3, 'M', structure='top')
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e')
sage: E.default_frame()
Local frame (E|_M, (e_0,e_1))
```
**fiber** *(point)*

Return the vector bundle fiber over a point.

**INPUT:**
- `point` – *ManifoldPoint*: point \( p \) of the base space of `self`

**OUTPUT:**
- instance of `VectorBundleFiber` representing the fiber over \( p \)

**EXAMPLES:**

```sage
M = Manifold(3, 'M', structure='top')
sage: X.<x,y,z> = M.chart()
sage: p = M((0,2,1), name='p'); p
Point p on the 3-dimensional topological manifold M
sage: E = M.vector_bundle(2, 'E'); E
Topological real vector bundle E -> M of rank 2 over the base space
3-dimensional topological manifold M
sage: E.fiber(p)
Fiber of E at Point p on the 3-dimensional topological manifold M
```

**frames()**

Return the list of local frames defined on `self`.

**OUTPUT:**
- list of local frames defined on `self`

**EXAMPLES:**

```sage
M = Manifold(3, 'M', structure='top')
sage: E = M.vector_bundle(2, 'E')
sage: U = M.open_subset('U'); V = M.open_subset('V')
sage: phi_U = E.trivialization('phi_U', domain=U)
sage: e = E.local_frame('e', domain=V)
sage: E.frames()
[Trivialization frame (E|_U, ((phi_U^*e_1),(phi_U^*e_2))),
Local frame (E|_V, (e_0,e_1))]
```

**has_orientation()**

Check whether `self` admits an obvious or by user set orientation.

**See also:**
- Consult `orientation()` for details about orientations.

**Note:** Notice that if `has_orientation()` returns `False` this does not necessarily mean that the vector bundle admits no orientation. It just means that the user has to set an orientation manually in that case, see `set_orientation()`.

**EXAMPLES:**

The trivial case:

```sage
M = Manifold(3, 'M', structure='top')
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e')
```
Non-trivial case:

```python
sage: M = Manifold(3, 'M', structure='top')
sage: U = M.open_subset('U'); V = M.open_subset('V')
sage: M.declare_union(U, V)

sage: E = M.vector_bundle(2, 'E')

sage: e = E.local_frame('e', domain=U)
sage: f = E.local_frame('f', domain=V)

sage: E.has_orientation()
False

sage: E.set_orientation([e, f])

sage: E.has_orientation()
True
```

**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 `sage.manifolds.manifold.Manifold.start_index()` is assumed

**OUTPUT:**

- an iterable index, starting from \( i_0 \) and ending at \( i_0 + n - 1 \), where \( n \) is the vector bundle’s dimension

**EXAMPLES:**

Index range on a 4-dimensional vector bundle over a 5-dimensional manifold:

```python
sage: M = Manifold(5, 'M', structure='topological')
sage: E = M.vector_bundle(4, 'E')

sage: list(E.irange())
[0, 1, 2, 3]

sage: list(E.irange(2))
[2, 3]
```

Index range on a 4-dimensional vector bundle over a 5-dimensional manifold with starting index=1:

```python
sage: M = Manifold(5, 'M', structure='topological', start_index=1)
sage: E = M.vector_bundle(4, 'E')

sage: list(E.irange())
[1, 2, 3, 4]

sage: list(E.irange(2))
[2, 3, 4]
```

In general, one has always:

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

**is_manfestly_trivial()**

Return True if self is manifestly a trivial bundle, i.e. there exists a frame or a trivialization defined on the whole base space.

**EXAMPLES:**
local_frame (*args, **kwargs)

Define a local frame on self.

A local frame is a section on a subset $U \subset M$ in $E$ that provides, at each point $p$ of the base space, a vector basis of the fiber $E_p$ at $p$.

See also:

LocalFrame for complete documentation.

INPUT:

- symbol – either a string, to be used as a common base for the symbols of the sections constituting the local frame, or a list/tuple of strings, representing the individual symbols of the sections

- sections – tuple or list of $n$ linearly independent sections on self ($n$ being the rank of self) defining the local frame; can be omitted if the local frame is created from scratch

- latex_symbol – (default: None) either a string, to be used as a common base for the LaTeX symbols of the sections constituting the local frame, or a list/tuple of strings, representing the individual LaTeX symbols of the sections; if None, symbol is used in place of latex_symbol

- indices – (default: None; used only if symbol is a single string) tuple of strings representing the indices labelling the sections of the frame; if None, the indices will be generated as integers within the range declared on self

- latex_indices – (default: None) tuple of strings representing the indices for the LaTeX symbols of the sections; if None, indices is used instead

- symbol_dual – (default: None) same as symbol but for the dual coframe; if None, symbol must be a string and is used for the common base of the symbols of the elements of the dual coframe

- latex_symbol_dual – (default: None) same as latex_symbol but for the dual coframe

- domain – (default: None) domain on which the local frame is defined; if None, the whole base space is assumed

OUTPUT:

- a LocalFrame representing the defined local frame

EXAMPLES:

Defining a local frame from two linearly independent sections on a real rank-2 vector bundle:
If the sections are not linearly independent, an error is raised:

```sage```
```python```
sage: e = E.local_frame('z', (s0, -s0), domain=U)
Traceback (most recent call last):
  ...  
ValueError: the provided sections are not linearly independent
```
```sage```
```python```

It is also possible to create a local frame from scratch, without connecting it to previously defined local frames or sections (this can still be performed later via the method `set_change_of_frame()`):

```sage```
```python```
sage: f = E.local_frame('f', domain=U); f
Local frame (E|_U, (f_0,f_1))
```
```sage```
```python```

For a global frame, the argument `domain` is omitted:

```sage```
```python```
sage: g = E.local_frame('g'); g
Local frame (E|_M, (g_0,g_1))
```
```sage```
```python```

See also:

For more options, in particular for the choice of symbols and indices, see `LocalFrame`.

`orientation()`

Get the orientation of `self` if available.

An orientation on a vector bundle is a choice of local frames whose

1. union of domains cover the base space,

2. changes of frames are pairwise orientation preserving, i.e. have positive determinant.

A vector bundle endowed with an orientation is called orientable.

The trivial case corresponds to `self` being trivial, i.e. `self` can be covered by one frame. In that case, if no preferred orientation has been set before, one of those frames (usually the default frame) is set automatically to the preferred orientation and returned here.

EXAMPLES:

The trivial case is covered automatically:

```sage```
```python```
sage: M = Manifold(3, 'M', structure='top')
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e'); e
Local frame (E|_M, (e_0,e_1))
```
```sage```
```python```
(continues on next page)
The orientation can also be set by the user:

```sage
sage: f = E.local_frame('f'); f
Local frame (E|_M, (f_0,f_1))
sage: E.set_orientation(f)
sage: E.orientation()
[Local frame (E|_M, (f_0,f_1))]
```

In case of the non-trivial case, the orientation must be set manually, otherwise no orientation is returned:

```sage
sage: M = Manifold(3, 'M', structure='top')
sage: U = M.open_subset('U'); V = M.open_subset('V')
sage: M.declare_union(U, V)
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e', domain=U); e
Local frame (E|_U, (e_0,e_1))
sage: f = E.local_frame('f', domain=V); f
Local frame (E|_V, (f_0,f_1))
sage: E.orientation()
[]
sage: E.set_orientation([e, f])
sage: E.orientation()
[Local frame (E|_U, (e_0,e_1)),
 Local frame (E|_V, (f_0,f_1))]
```

### rank()

Return the rank of the vector bundle.

**EXAMPLES:**

```sage
sage: M = Manifold(2, 'M', structure='top')
sage: E = M.vector_bundle(3, 'E')
sage: E.rank()
3
```

### section(*comp, **kwargs)

Return a continuous section of self.

**INPUT:**

- domain – (default: None) domain on which the section shall be defined; if None, the base space is assumed
- name – (default: None) name of the local section
- latex_name – (default: None) latex representation of the local section

**OUTPUT:**

- an instance of `Section` representing a continuous section of $M$ with values on $E$

**EXAMPLES:**

A section on a non-trivial rank 2 vector bundle over a non-trivial 2-manifold:
sage: M = Manifold(2, 'M', structure='top')
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()
    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: E = M.vector_bundle(2, 'E')   # define the vector bundle
sage: phi_U = E.trivialization('phi_U', domain=U)   # define trivializations
sage: phi_V = E.trivialization('phi_V', domain=V)

sage: phi_U = E.trivialization('phi_U', domain=U)
    phi_V = E.trivialization('phi_V', domain=V)

sage: transf = phi_U.transition_map(phi_V, [[0,x],[x,0]])  # transition map

section_module(domain=None, force_free=False)

Return the section module of continuous sections on self.

See SectionModule for a complete documentation.

INPUT:

• domain – (default: None) the domain on which the module is defined; if None the base space is assumed

• force_free – (default: False) if set to True, force the construction of a free module (this implies that E is trivial)

OUTPUT:

• a SectionModule (or if E is trivial, a SectionFreeModule) representing the module of continuous sections on U taking values in E

EXAMPLES:

Module of sections on the Möbius bundle over the real-projective space $M = \mathbb{R}P^1$:

sage: M = Manifold(1, 'RP^1', structure='top', start_index=1)
sage: U = M.open_subset('U') # the complement of one point
    c_u.<u> = U.chart() # [1:u] in homogeneous coord.
sage: V = M.open_subset('V') # the complement of the point u=0
    c_v.<v> = V.chart() # [v:1] in homogeneous coord.
sage: c_u.<u> = U.chart() # [1:u] in homogeneous coord.
sage: c_v.<v> = V.chart()
sage: u_to_v = c_u.transition_map(c_v, (1/u),
    intersection_name='W',
    restrictions1 = u!=0,
    restrictions2 = v!=0)
sage: v_to_u = u_to_v.inverse()
sage: W = U.intersection(V)
sage: E = M.vector_bundle(1, 'E')
sage: phi_U = E.trivialization('phi_U', latex_name=r'\varphi_U',
    domain=U)
sage: phi_V = E.trivialization('phi_V', latex_name=r'\varphi_V',
    domain=V)

sage: transf = phi_U.transition_map(phi_V, [[0,u]])
sage: C0 = E.section_module(); C0

(continues on next page)
Module $C^0(\mathbb{RP}^1;E)$ of sections on the 1-dimensional topological manifold $\mathbb{RP}^1$ with values in the real vector bundle $E$ of rank 1

$C^0(\mathbb{RP}^1;E)$ is a module over the algebra $C^0(\mathbb{RP}^1)$:

```
sage: C0.category()
Category of modules over Algebra of scalar fields on the 1-dimensional topological manifold RP^1
sage: C0.base_ring() is M.scalar_field_algebra()
True
```

However, $C^0(\mathbb{RP}^1;E)$ is not a free module:

```
sage: isinstance(C0, FiniteRankFreeModule)
False
```

since the Möbius bundle is not trivial:

```
sage: E.is_manifestly_trivial()
False
```

The section module over $U$, on the other hand, is a free module since $E|_U$ admits a trivialization and therefore has a local frame:

```
sage: C0_U = E.section_module(domain=U)
sage: isinstance(C0_U, FiniteRankFreeModule)
True
```

The elements of $C^0(U)$ are sections on $U$:

```
sage: C0_U.an_element()
Section on the Open subset U of the 1-dimensional topological manifold RP^1 with values in the real vector bundle E of rank 1
sage: C0_U.an_element().display(phi_U.frame())
 2 (phi_U^*e_1)
```

`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 `FreeModuleAutomorphism` 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:**

```
sage: M = Manifold(3, 'M')
sage: c_xyz.<x,y,z> = M.chart()
sage: E = M.vector_bundle(2, 'E')
```
\texttt{sage: e = E.local_frame('e')}
\texttt{sage: f = E.local_frame('f')}
\texttt{sage: a = E.section_module().automorphism()}
\texttt{sage: a[e,:] = \begin{bmatrix} 1,2 \\ 0,3 \end{bmatrix}}
\texttt{sage: E.set_change_of_frame(e, f, a)}
\texttt{sage: f[0].display(e)}
\texttt{f_0 = e_0}
\texttt{sage: f[1].display(e)}
\texttt{f_1 = 2 e_0 + 3 e_1}
\texttt{sage: e[0].display(f)}
\texttt{e_0 = f_0}
\texttt{sage: e[1].display(f)}
\texttt{e_1 = -2/3 f_0 + 1/3 f_1}
\texttt{sage: E.change_of_frame(e,f)[e,:]}
\begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix}
\textit{set_default_frame} (\texttt{frame})
Set the default frame of \texttt{self}.
\textbf{INPUT:}
\begin{itemize}
\item \texttt{frame} – a local frame defined on \texttt{self} as an instance of \texttt{LocalFrame}
\end{itemize}
\textbf{EXAMPLES:}
\begin{verbatim}
\texttt{sage: M = Manifold(3, 'M', structure='top')}
\texttt{sage: E = M.vector_bundle(2, 'E')}
\texttt{sage: e = E.local_frame('e')}
\texttt{sage: E.default_frame()}
Local frame (E|_M, (e_0,e_1))
\texttt{sage: f = E.local_frame('f')}
\texttt{sage: E.set_default_frame(f)}
\texttt{sage: E.default_frame()}
Local frame (E|_M, (f_0,f_1))
\end{verbatim}
\textit{set_orientation} (\texttt{orientation})
Set the preferred orientation of \texttt{self}.
\textbf{INPUT:}
\begin{itemize}
\item \texttt{orientation} – a local frame or a list of local frames whose domains cover the base space
\end{itemize}
\textbf{Warning:} It is the user's responsibility that the orientation set here is indeed an orientation. There is no check going on in the background. See \texttt{orientation()} for the definition of an orientation.
\textbf{EXAMPLES:}
Set an orientation on a vector bundle:
\begin{verbatim}
\texttt{sage: M = Manifold(3, 'M', structure='top')}
\texttt{sage: E = M.vector_bundle(2, 'E')}
\texttt{sage: e = E.local_frame('e'); e}
Local frame (E|_M, (e_0,e_1))
\texttt{sage: f = E.local_frame('f'); f}
Local frame (E|_M, (f_0,f_1))
\end{verbatim}
(continues on next page)
Set an orientation in the non-trivial case:

```
sage: M = Manifold(3, 'M', structure='top')
sage: U = M.open_subset('U'); V = M.open_subset('V')
sage: E = M.vector_bundle(2, 'E')
```

```
sage: e = E.local_frame('e', domain=U); e
Local frame (E|_U, (e_0,e_1))
sage: f = E.local_frame('f', domain=V); f
Local frame (E|_V, (f_0,f_1))
sage: E.orientation()
[]
sage: E.set_orientation([e, f])
sage: E.orientation()
[Local frame (E|_U, (e_0,e_1)),
 Local frame (E|_V, (f_0,f_1))]
```

**total_space()**

Return the total space of self.

**Note:** At this stage, the total space does not come with induced charts.

**OUTPUT:**

- the total space of self as an instance of `TopologicalManifold`

**EXAMPLES:**

```
sage: M = Manifold(3, 'M')
sage: X.<x,y,z> = M.chart()
sage: U = M.open_subset('U')
sage: V = M.open_subset('V')
```

```
sage: E = M.vector_bundle(2, 'E')
sage: E.total_space()
6-dimensional topological manifold E
```

**transition**(triv1, triv2)

Return the transition map between two trivializations defined over the manifold.

The transition map must have been defined previously, for instance by the method `transition_map()`.

**INPUT:**

- triv1 – trivialization 1
- triv2 – trivialization 2

**OUTPUT:**

- instance of `TransitionMap` representing the transition map from trivialization 1 to trivialization 2

**EXAMPLES:**

```
sage: M = Manifold(3, 'M')
sage: X.<x,y,z> = M.chart()
sage: U = M.open_subset('U')
sage: V = M.open_subset('V')
```

(continues on next page)
transitions()  

Return the transition maps defined over subsets of the base space. 

OUTPUT:  

- dictionary of transition maps, with pairs of trivializations as keys 

EXAMPLES: 

```
sage: M = Manifold(3, 'M')  
sage: U = M.open_subset('U')  
sage: V = M.open_subset('V')  
sage: X_UV = X.restrict(U.intersection(V))  
sage: E = M.vector_bundle(2, 'E')  
sage: phi_U = E.trivialization('phi_U', domain=U)  
sage: phi_V = E.trivialization('phi_V', domain=V)  
sage: phi_U_to_phi_V = phi_U.transition_map(phi_V, 1)  
sage: E.transitions()  
# random  
{(Trivialization (phi_U, E|_U),  
  Trivialization (phi_U, E|_V)): Transition map from Trivialization  
  (phi_U, E|_U) to Trivialization  
  (phi_U, E|_V),  
  (Trivialization (phi_U, E|_V),  
  Trivialization (phi_U, E|_U)): Transition map from Trivialization  
  (phi_U, E|_V) to Trivialization  
  (phi_U, E|_U)}
```

trivialization(name=None, domain=None, latex_name=None)  

Return a trivialization of self over the domain domain. 

INPUT:  

- domain – (default: None) domain on which the trivialization is defined; if None the base space is assumed 
- name – (default: None) name given to the trivialization 
- latex_name – (default: None) LaTeX name given to the trivialization 

OUTPUT:  

- a Trivialization representing a trivialization of E 

EXAMPLES: 

```
sage: M = Manifold(3, 'M')  
sage: U = M.open_subset('U')  
sage: E = M.vector_bundle(2, 'E')  
sage: phi = E.trivialization('phi', domain=U); phi  
Trivialization (phi, E|_U)
```
1.9.2 Vector Bundle Fibers

The class `VectorBundleFiber` implements fibers over a vector bundle.

AUTHORS:
- Michael Jung (2019): initial version

```python
class sage.manifolds.vector_bundle_fiber.VectorBundleFiber(vector_bundle, point)
    Bases: FiniteRankFreeModule
```

Fiber of a given vector bundle at a given point.

Let $\pi : E \to M$ be a vector bundle of rank $n$ over the field $K$ (see `TopologicalVectorBundle`) and $p \in M$. The fiber $E_p$ at $p$ is defined via $E_p := \pi^{-1}(p)$ and takes the structure of an $n$-dimensional vector space over the field $K$.

INPUT:
- `vector_bundle` – `TopologicalVectorBundle`; vector bundle $E$ on which the fiber is defined
- `point` – `ManifoldPoint`; point $p$ at which the fiber is defined

EXAMPLES:

A vector bundle fiber in a trivial rank 2 vector bundle over a 4-dimensional topological manifold:

```python
sage: M = Manifold(4, M, structure='top')
sage: X.<x,y,z,t> = M.chart()
sage: p = M((0,0,0,0), name='p')
sage: E = M.vector_bundle(2, E)
sage: e = E.local_frame(e)
sage: Ep = E.fiber(p); Ep
Fiber of E at Point p on the 4-dimensional topological manifold M
```

Fibers are free modules of finite rank over `SymbolicRing` (actually vector spaces of finite dimension over the vector bundle field $K$, here $K = \mathbb{R}$):

```python
sage: Ep.base_ring()
Symbolic Ring
sage: Ep.category()
Category of finite dimensional vector spaces over Symbolic Ring
sage: Ep.rank()
2
sage: dim(Ep)
2
```

The fiber is automatically endowed with bases deduced from the local frames around the point:

```python
sage: Ep.bases()
[Basis (e_0,e_1) on the Fiber of E at Point p on the 4-dimensional topological manifold M]
sage: E.frames()
[Local frame (E|_M, (e_0,e_1))]
```

At this stage, only one basis has been defined in the fiber, but new bases can be added from local frames on the vector bundle by means of the method `at()`:

```python
sage: aut = E.section_module().automorphism()
sage: aut[:]= [[-1, x], [y, 2]]
sage: f = e.new_frame(aut, 'f')
```

(continues on next page)
sage: fp = f.at(p); fp
Basis (f_0,f_1) on the Fiber of E at Point p on the 4-dimensional
topological manifold M
sage: Ep.bases()
[Basis (e_0,e_1) on the Fiber of E at Point p on the 4-dimensional
topological manifold M,
  Basis (f_0,f_1) on the Fiber of E at Point p on the 4-dimensional
  topological manifold M]

The changes of bases are applied to the fibers:

sage: f[1].display(e)  # second component of frame f
f_1 = x e_0 + 2 e_1
sage: ep = e.at(p)
sage: fp[1].display(ep)  # second component of frame f at p
f_1 = 2 e_1

All the bases defined on \( E_p \) are on the same footing. Accordingly the fiber is not in the category of modules with
a distinguished basis:

sage: Ep in ModulesWithBasis(SR)
False

It is simply in the category of modules:

sage: Ep in Modules(SR)
True

Since the base ring is a field, it is actually in the category of vector spaces:

sage: Ep in VectorSpaces(SR)
True

A typical element:

sage: v = Ep.an_element(); v
Vector in the fiber of E at Point p on the 4-dimensional topological
manifold M
sage: v.display()
e_0 + 2 e_1
sage: v.parent()
Fiber of E at Point p on the 4-dimensional topological manifold M

The zero vector:

sage: Ep.zero()
Vector zero in the fiber of E at Point p on the 4-dimensional
topological manifold M
sage: Ep.zero().display()
zero = 0
sage: Ep.zero().parent()
Fiber of E at Point p on the 4-dimensional topological manifold M

Fibers are unique:
sage: E.fiber(p) is Ep
True
sage: p1 = M.point((0,0,0,0))
sage: E.fiber(p1) is Ep
True

even if points are different instances:

sage: p1 is p
False

but p1 and p share the same fiber because they compare equal:

sage: p1 == p
True

See also:

FiniteRankFreeModule for more documentation.

Element

alias of VectorBundleFiberElement

base_point()

Return the manifold point over which self is defined.

EXAMPLES:

sage: M = Manifold(2, 'M', structure='top')
sage: X.<x,y> = M.chart()
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e')
sage: p = M.point((3,-2), name='p')
sage: Ep = E.fiber(p)
sage: Ep.base_point()
Point p on the 2-dimensional topological manifold M
sage: p is Ep.base_point()
True

collection()}

dim()

Return the vector space dimension of self.

EXAMPLES:

sage: M = Manifold(3, 'M', structure='top')
sage: X.<x,y,z> = M.chart()
sage: p = M((0,0,0), name='p')
sage: E = M.vector_bundle(2, 'E')
sage: Ep = E.fiber(p)
sage: Ep.dim()
2

dimension()

Return the vector space dimension of self.

EXAMPLES:
1.9.3 Vector Bundle Fiber Elements

The class `VectorBundleFiberElement` implements vectors in the fiber of a vector bundle.

**AUTHORS:**
- Michael Jung (2019): initial version

```python
class sage.manifolds.vector_bundle_fiber_element.VectorBundleFiberElement(parent, name=None, latex_name=None)
```

**Bases:** `FiniteRankFreeModuleElement`

Vector in a fiber of a vector bundle at the given point.

**INPUT:**
- `parent` – `VectorBundleFiber`; the fiber 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:**
A vector $v$ in a fiber of a rank 2 vector bundle:

```python
sage: M = Manifold(2, 'M', structure='top')
sage: X.<x,y> = M.chart()
sage: p = M((1,-1), name='p')
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e')
sage: Ep = E.fiber(p)
sage: v = Ep((-2,1), name='v'); v
Vector v in the fiber of E at Point p on the 2-dimensional topological manifold M
sage: v.display()
v = -2 e_0 + e_1
sage: v.parent()
Fiber of E at Point p on the 2-dimensional topological manifold M
sage: v in Ep
True
```

See also:
`FiniteRankFreeModuleElement` for more documentation.
1.9.4 Trivializations

The class `Trivialization` implements trivializations on vector bundles. The corresponding transition maps between two trivializations are represented by `TransitionMap`.

**AUTHORS:**
- Michael Jung (2019): initial version

```python
class sage.manifolds.trivialization.TransitionMap(triv1, triv2, transf, compute_inverse=True)
    Bases: SageObject

    Transition map between two trivializations.

    Given a vector bundle $\pi : E \to M$ of class $C^k$ and rank $n$ over the field $K$, and two trivializations $\varphi_U : \pi^{-1}(U) \to U \times K^n$ and $\varphi_V : \pi^{-1}(V) \to V \times K^n$, the transition map from $\varphi_U$ to $\varphi_V$ is given by the composition

    \[ \varphi_V \circ \varphi_U^{-1} : U \cap V \times K^n \to U \cap V \times K^n. \]

    This composition is of the form

    \[(p, v) \mapsto (p, g(p)v),\]

    where $p \mapsto g(p)$ is a $C^k$ family of invertible $n \times n$ matrices.

**INPUT:**
- `triv1` – trivialization 1
- `triv2` – trivialization 2
- `transf` – the transformation between both trivializations in form of a matrix of scalar fields (`ScalarField`) or coordinate functions (`ChartFunction`), or a bundle automorphism (`FreeModuleAutomorphism`)
- `compute_inverse` – (default: True) determines whether the inverse shall be computed or not

**EXAMPLES:**
Transition map of two trivializations on a real rank 2 vector bundle of the 2-sphere:

```python
S2 = Manifold(2, 'S^2', structure='top')
U = S2.open_subset('U') ; V = S2.open_subset('V')
# complement of the North and South pole, respectively
S2.declare_union(U,V)
c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
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)
W = U.intersection(V)
xy_to_uv = xy_to_uv.inverse()
E = S2.vector_bundle(2, 'E')
phi_U = E.trivialization('phi_U', domain=U)
phi_V = E.trivialization('phi_V', domain=V)
phi_U_to_phi_V = phi_U.transition_map(phi_V, [[0,1],[1,0]])
phi_U_to_phi_V
```

Transition map from Trivialization `(phi_U, E|_U)` to Trivialization `(phi_V, E|_V)`

272 Chapter 1. Topological Manifolds
automorphism()

Return the automorphism connecting both trivializations.

The family of matrices $p \mapsto g(p)$ given by the transition map induce a bundle automorphism

$$\varphi_U^{-1} \circ \varphi_V : \pi^{-1}(U \cap V) \rightarrow \pi^{-1}(U \cap V)$$

correlating the local frames induced by the trivializations in the following way:

$$(\varphi_U^{-1} \circ \varphi_V)(\varphi_V^* e_i) = \varphi_U^* e_i.$$ Then, for each point $p \in M$, the matrix $g(p)$ is the representation of the induced automorphism on the fiber $E_p = \pi^{-1}(p)$ in the basis $((\varphi_V^* e_i)(p))_{i=1, \ldots, n}$.

EXAMPLES:

```
sage: S2 = Manifold(2, 'S^2', structure='top')
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: W = U.intersection(V)
sage: uv_to_xy = xy_to_uv.inverse() # stereographic coordinates from the South pole
sage: phi_U = E.trivialization('phi_U', latex_name=r'\varphi_U',
    ...: domain=U); phi_U
Trivialization (phi_U, E|_U)
sage: phi_V = E.trivialization('phi_V', latex_name=r'\varphi_V',
    ...: domain=V); phi_V
Trivialization (phi_V, E|_V)
sage: phi_U_to_phi_V = phi_U.transition_map(phi_V, [[ 0,1],
    ...: [1,0]])
sage: aut = phi_U_to_phi_V.automorphism(); aut
Automorphism phi_U^(-1)*phi_V of the Free module C^0(W;E) of sections on the Open subset W of the 2-dimensional topological manifold S^2 with values in the real vector bundle E of rank 2
sage: aut.display(phi_U.frame().restrict(W))
phi_U^(-1)*phi_V = (phi_U^* e_1)⊗(phi_U^* e^2) + (phi_U^* e_2)⊗(phi_U^* e^1)
```
intersection_name=W, restrictions1 = x^2+y^2!=0,
.....: restrictions2= u^2+v^2!=0)
sage: W = U.intersection(V)
sage: uv_to_xy = xy_to_uv.inverse()
sage: E = S2.vector_bundle(2, 'E')
sage: phi_U = E.trivialization('phi_U', latex_name=r'$\varphi_U$',
.....: domain=U); phi_U
Trivialization (phi_U, E|_U)
sage: phi_V = E.trivialization('phi_V', latex_name=r'$\varphi_V$',
.....: domain=V); phi_V
Trivialization (phi_V, E|_V)
sage: phi_U.to.phi_V = phi_U.transition_map(phi_V, [[0,1],[1,0]])
sage: det = phi_U.to.phi_V.det(); det
Scalar field det(phi_U^{-1}*phi_V) on the Open subset W of the
2-dimensional topological manifold S^2
sage: det.display()
det(phi_U^{-1}*phi_V): W → ℝ
(x, y) ↦ -1
(u, v) ↦ -1

inverse()
Return the inverse transition map.

EXAMPLES:

matrix()
Return the matrix representation the transition map.

EXAMPLES:

Local trivializations on a real rank 2 vector bundle over the 2-sphere:
The input is coerced into a bundle automorphism. From there, the matrix can be recovered:

```
sage: phi_U_to_phi_V = phi_U.transition_map(phi_V, [[0,1],[1,0]])
sage: matrix = phi_U_to_phi_V.matrix(); matrix
```

Let us check the matrix components:

```
sage: matrix[0,0].display()
zero: W → ℝ
  (x, y) ↦ 0
  (u, v) ↦ 0
sage: matrix[0,1].display()
1: W → ℝ
  (x, y) ↦ 1
  (u, v) ↦ 1
sage: matrix[1,0].display()
1: W → ℝ
  (x, y) ↦ 1
  (u, v) ↦ 1
sage: matrix[1,1].display()
zero: W → ℝ
  (x, y) ↦ 0
  (u, v) ↦ 0
```

### class sage.manifolds.trivialization.Trivialization(vector_bundle, name, domain, latex_name=None)

Bases: UniqueRepresentation, SageObject

A local trivialization of a given vector bundle.

Let \( \pi : E \to M \) be a vector bundle of rank \( n \) and class \( C^k \) over the field \( K \) (see TopologicalVectorBundle or DifferentiableVectorBundle). A local trivialization over an open subset \( U \subseteq M \) is a \( C^k \)-diffeomorphism \( \varphi : \pi^{-1}(U) \to U \times K^n \) such that \( \pi \circ \varphi^{-1} = \text{pr}_1 \) and \( v \mapsto \varphi^{-1}(q, v) \) is a linear isomorphism for any \( q \in U \).

**Note:** Notice that frames and trivializations are equivalent concepts (for further details see LocalFrame). However, in order to facilitate applications and being consistent with the implementations of charts, trivializations are introduced separately.
EXAMPLES:

Local trivializations on a real rank 2 vector bundle over the 2-sphere:

```
sage: S2 = Manifold(2, 'S^2', structure='top')
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: W = U.intersection(V)
sage: uv_to_xy = xy_to_uv.inverse()
sage: E = S2.vector_bundle(2, 'E')
sage: phi_U = E.trivialization(phi_U, latex_name=r'\varphi_U',
.....: domain=U); phi_U
Trivialization (phi_U, E|_U)
sage: phi_V = E.trivialization(phi_V, latex_name=r'\varphi_V',
.....: domain=V); phi_V
Trivialization (phi_V, E|_V)
sage: phi_U_to_phi_V = phi_U.transition_map(phi_V, [0,1],[1,0])
Transition map from Trivialization (phi_U, E|_U) to Trivialization (phi_V, E|_V)
```

The LaTeX output gives the following:

```
sage: latex(phi_U)
\varphi_U : E |_{U} \to U \times \Bold{R}^2
sage: latex(phi_V)
\varphi_V : E |_{V} \to V \times \Bold{R}^2
```

The trivializations are part of the vector bundle atlas:

```
sage: E.atlas()
[(Trivialization (phi_U, E|_U), Trivialization (phi_V, E|_V)]
```

Each trivialization induces a local trivialization frame:

```
sage: fU = phi_U.frame(); fU
Trivialization frame (E|_U, ((phi_U^*e_1),(phi_U^*e_2)))
sage: fV = phi_V.frame(); fV
Trivialization frame (E|_V, ((phi_V^*e_1),(phi_V^*e_2)))
```

and the transition map connects these two frames via a bundle automorphism:

```
sage: aut = phi_U_to_phi_V.automorphism(); aut
Automorphism phi_U^(-1)*phi_V of the Free module C^0(W;E) of sections on
the Open subset W of the 2-dimensional topological manifold S^2 with
values in the real vector bundle E of rank 2
sage: aut.display(fU.restrict(W))
phi_U^(-1)*phi_V = (phi_U^*e_1)⊗(phi_U^*e_2) + (phi_U^*e_2)⊗(phi_U^*e^1)
sage: aut.display(fV.restrict(W))
phi_U^(-1)*phi_V = (phi_V^*e_1)⊗(phi_V^*e_2) + (phi_V^*e_2)⊗(phi_V^*e^1)
```

The automorphisms are listed in the frame changes of the vector bundle:
Let us check the components of $f_U$ with respect to the frame $f_V$:

```plaintext
sage: fU[0].comp(fV.restrict(W))[:]
[0, 1]
sage: fU[1].comp(fV.restrict(W))[:]
[1, 0]
```

**base_space()**

Return the manifold on which the trivialization is defined.

**EXAMPLES:**

```plaintext
sage: M = Manifold(2, 'M', structure='top')
sage: U = M.open_subset('U')
sage: E = M.vector_bundle(2, 'E')
sage: phi = E.trivialization('phi', domain=U)
sage: phi.base_space()
2-dimensional topological manifold M
```

**coframe()**

Return the standard coframe induced by self.

**See also:**

*LocalCoFrame*

**EXAMPLES:**

```plaintext
sage: M = Manifold(2, 'M', structure='top')
sage: E = M.vector_bundle(2, 'E')
sage: phi = E.trivialization('phi')
sage: phi.coframe()
Trivialization coframe (E|_M, ((phi^*e^1),(phi^*e^2)))
```

**domain()**

Return the domain on which the trivialization is defined.

**EXAMPLES:**

```plaintext
sage: M = Manifold(2, 'M', structure='top')
sage: U = M.open_subset('U')
sage: E = M.vector_bundle(2, 'E')
sage: phi = E.trivialization('phi', domain=U)
sage: phi.domain()
Open subset U of the 2-dimensional topological manifold M
```
frame()

Return the standard frame induced by self. If \( \psi \) is a trivialization then the corresponding frame can be obtained by the maps \( p \mapsto \psi^{-1}(p, e_i) \), where \( (e_1, \ldots, e_n) \) is the standard basis of \( K^n \). We briefly denote \( (\psi^* e_i) \) instead of \( \psi^{-1}(\cdot, e_i) \).

See also:

LocalFrame

EXAMPLES:

```sage
M = Manifold(2, 'M', structure='top')
E = M.vector_bundle(2, 'E')
phi = E.trivialization('phi')
phi.frame()
```

transition_map(other, transf, compute_inverse=True)

Return the transition map between self and other.

INPUT:

- other - the trivialization where the transition map from self goes to
- transf - transformation of the transition map
- intersection_name - (default: None) name to be given to the subset \( U \cap V \) if the latter differs from \( U \) or \( V \)

EXAMPLES:

```sage
M = Manifold(2, 'M', structure='top')
X.<x,y> = M.chart()
U = M.open_subset('U'); V = M.open_subset('V')
XU = X.restrict(U); XV = X.restrict(U)
W = U.intersection(V)
XW = X.restrict(W)
E = M.vector_bundle(2, 'E')
phi_U = E.trivialization('phi_U', domain=U)
phi_V = E.trivialization('phi_V', domain=V)
phi_U.transition_map(phi_V, 1)
```

vector_bundle()

Return the vector bundle on which the trivialization is defined.

EXAMPLES:

```sage
M = Manifold(2, 'M', structure='top')
U = M.open_subset('U')
E = M.vector_bundle(2, 'E')
phi = E.trivialization('phi', domain=U)
phi.vector_bundle()
```

Topological real vector bundle \( E \to M \) of rank 2 over the base space 2-dimensional topological manifold \( M \)
1.9.5 Local Frames

The class `LocalFrame` implements local frames on vector bundles (see `TopologicalVectorBundle` or `DifferentiableVectorBundle`).

For $k = 0, 1, \ldots$, a local frame on a vector bundle $E \to M$ of class $C^k$ and rank $n$ is a local section $(e_1, \ldots, e_n): U \to E^n$ of class $C^k$ defined on some subset $U$ of the base space $M$, such that $e(p)$ is a basis of the fiber $E_p$ for any $p \in U$.

AUTHORS:

- Michael Jung (2019): initial version

EXAMPLES:

Defining a global frame on a topological vector bundle of rank 3:

```sage
M = Manifold(3, 'M', structure='top')
E = M.vector_bundle(3, 'E')
e = E.local_frame('e'); e
Local frame (E|_M, (e_0,e_1,e_2))
```

This frame is now the default frame of the corresponding section module and saved in the vector bundle:

```sage
e in E.frames()
True
sec_module = E.section_module(); sec_module
Free module C^0(M;E) of sections on the 3-dimensional topological manifold M
with values in the real vector bundle E of rank 3
sec_module.default_basis()
Local frame (E|_M, (e_0,e_1,e_2))
```

However, the default frame can be changed:

```sage
sec_module.set_default_basis(e)
sec_module.default_basis()
Local frame (E|_M, (e_0,e_1,e_2))
```

The elements of a local frame are local sections in the vector bundle:

```sage
for vec in e:
    print(vec)
```

Each element of a vector frame can be accessed by its index:

```sage
e[0]
Section e_0 on the 3-dimensional topological manifold M with values in the
real vector bundle E of rank 3
```

The slice operator `:` can be used to access to more than one element:

```sage
e[0:2]
(Section e_0 on the 3-dimensional topological manifold M with values in the
real vector bundle E of rank 3,
```
The index range depends on the starting index defined on the manifold:

```
sage: M = Manifold(3, 'M', structure='top', start_index=1)
sage: c_xyz.<x,y,z> = M.chart()
sage: U = M.open_subset('U')
sage: c_xyz_U = c_xyz.restrict(U)
sage: E = M.vector_bundle(3, 'E')
sage: e = E.local_frame('e', domain=U); e
Local frame (E|_U, (e_1,e_2,e_3))
sage: [e[i] for i in M.irange()]
[Section e_1 on the Open subset U of the 3-dimensional topological manifold
M with values in the real vector bundle E of rank 3, Section e_2 on the Open subset U of the 3-dimensional topological manifold
M with values in the real vector bundle E of rank 3, Section e_3 on the Open subset U of the 3-dimensional topological manifold
M with values in the real vector bundle E of rank 3]
sage: e[1], e[2], e[3]
(Section e_1 on the Open subset U of the 3-dimensional topological manifold
M with values in the real vector bundle E of rank 3, Section e_2 on the Open subset U of the 3-dimensional topological manifold
M with values in the real vector bundle E of rank 3, Section e_3 on the Open subset U of the 3-dimensional topological manifold
M with values in the real vector bundle E of rank 3)
```

Let us check that the local sections $e[i]$ are indeed 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 local frame on a vector bundle, the dual coframe is automatically created, which, by default, bares the same name (here $e^i$):

```
sage: E.coframes()
[Local coframe (E|_U, (e^1,e^2,e^3))]
sage: e_dual = E.coframes()[0] ; e_dual
Local coframe (E|_U, (e^1,e^2,e^3))
sage: e_dual is e.coframe()
True
```

Let us check that the coframe $(e^i)$ is indeed the dual of the vector frame $(e_i)$:
sage: e_dual[1](e[1]) # linear form e^1 applied to local section e_1
Scalar field e^1(e_1) on the Open subset U of the 3-dimensional topological manifold M
sage: e_dual[1](e[1]).expr() # the explicit expression of e^1(e_1)
1
sage: e_dual[1](e[1]).expr(), e_dual[1](e[2]).expr(), e_dual[1](e[3]).expr()
(1, 0, 0)

Via bundle automorphisms, a new frame can be created from an existing one:

sage: sec_module_U = E.section_module(domain=U)

sage: change_frame = sec_module_U.automorphism()

sage: change_frame[:,:] = [[0,1,0],[0,0,1],[1,0,0]]

sage: f = e.new_frame(change_frame, f); f
Local frame (E|_U, (f_1,f_2,f_3))

A copy of this automorphism and its inverse is now part of the vector bundle’s frame changes:

sage: E.change_of_frame(e, f)
Automorphism of the Free module C^0(U;E) of sections on the Open subset U of the 3-dimensional topological manifold M with values in the real vector bundle E of rank 3
sage: E.change_of_frame(e, f) == change_frame
True
sage: E.change_of_frame(f, e) == change_frame.inverse()
True

Let us check the components of f with respect to the frame e:

sage: f[1].comp(e)[:]
[0, 0, 1]

sage: f[2].comp(e)[:]
[1, 0, 0]

sage: f[3].comp(e)[:]
[0, 1, 0]

class sage.manifolds.local_frame.LocalCoFrame(frame, symbol=None, latex_symbol=None, indices=None, latex_indices=None)

Bases: FreeModuleCoBasis

Local coframe on a vector bundle.

A local coframe on a vector bundle $E \to M$ of class $C^k$ is a local section $e^* : U \to E^n$ of class $C^k$ on some subset $U$ of the base space $M$, such that $e^*(p)$ is a basis of the fiber $E^n_p$ of the dual bundle for any $p \in U$.

INPUT:

• frame – the local frame dual to the coframe

• symbol – either a string, to be used as a common base for the symbols of the linear forms constituting the coframe, or a tuple of strings, representing the individual symbols of the linear forms

• latex_symbol – (default: None) either a string, to be used as a common base for the LaTeX symbols of the linear forms constituting the coframe, or a tuple of strings, representing the individual LaTeX symbols of the linear forms; if None, symbol is used in place of latex_symbol

1.9. Topological Vector Bundles
• indices – (default: None; used only if symbol is a single string) tuple of strings representing the indices labelling the linear forms of the coframe; if None, the indices will be generated as integers within the range declared on the coframe’s domain

• latex_indices – (default: None) tuple of strings representing the indices for the LaTeX symbols of the linear forms of the coframe; if None, indices is used instead

EXAMPLES:

Local coframe on a topological vector bundle of rank 3:

```python
sage: M = Manifold(3, 'M', structure='top', start_index=1)
sage: X.<x,y,z> = M.chart()
sage: E = M.vector_bundle(3, 'E')
sage: e = E.local_frame('e')
sage: from sage.manifolds.local_frame import LocalCoFrame
sage: f = LocalCoFrame(e, 'f'); f
Local coframe (E|_M, (f^1,f^2,f^3))
```

The local coframe can also be obtained by using the method `dual_basis()` or `coframe()`:

```python
sage: e_dual = e.dual_basis(); e_dual
Local coframe (E|_M, (e^1,e^2,e^3))
sage: e_dual is e.coframe()
True
sage: e_dual is f
False
sage: e_dual[: ] == f[: ]
True
sage: f[1].display(e)
f^1 = e^1
```

The consisted linear forms can be obtained via the operator []:

```python
sage: f[1], f[2], f[3]
(Linear form f^1 on the Free module C^0(M;E) of sections on the 3-dimensional topological manifold M with values in the real vector bundle E of rank 3,
Linear form f^2 on the Free module C^0(M;E) of sections on the 3-dimensional topological manifold M with values in the real vector bundle E of rank 3,
Linear form f^3 on the Free module C^0(M;E) of sections on the 3-dimensional topological manifold M with values in the real vector bundle E of rank 3)
```

Checking that $f$ is the dual of $e$:

```python
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)
```

at (point)

Return the value of self at a given point on the base space, this value being a basis of the dual vector bundle at this 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 \( E_p^* \), dual to the vector bundle fiber \( E_p \)

**EXAMPLES:**

Cobasis of a vector bundle fiber:

```python
sage: M = Manifold(2, 'M', structure='top', start_index=1)
sage: X.<x,y> = M.chart()
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e')
Local coframe (E|_M, (e^1,e^2))
sage: p = M.point((-1,2), name='p')
Local coframe (E|_M, (e^1,e^2)) on the Fiber of E at Point p on the 2-dimensional topological manifold M
sage: type(e_dual_p)
<class 'sage.tensor.modules.free_module_basis.FreeModuleCoBasis_with_category'>
sage: e_dual_p[1]
Linear form e^1 on the Fiber of E at Point p on the 2-dimensional topological manifold M
sage: e_dual_p[2]
Linear form e^2 on the Fiber of E at Point p on the 2-dimensional topological manifold M
sage: e_dual_p is e.at(p).dual_basis()
True
```

**set_name** *(symbol, latex_symbol=None, indices=None, latex_indices=None, index_position='up', include_domain=True)*

Set (or change) the text name and LaTeX name of self.

**INPUT:**

• **symbol** – either a string, to be used as a common base for the symbols of the linear forms constituting the coframe, or a list/tuple of strings, representing the individual symbols of the linear forms

• **latex_symbol** – (default: None) either a string, to be used as a common base for the LaTeX symbols of the linear forms constituting the coframe, or a list/tuple of strings, representing the individual LaTeX symbols of the linear forms; if None, symbol is used in place of latex_symbol

• **indices** – (default: None; used only if symbol is a single string) tuple of strings representing the indices labelling the linear forms of the coframe; if None, the indices will be generated as integers within the range declared on self

• **latex_indices** – (default: None) tuple of strings representing the indices for the LaTeX symbols of the linear forms; if None, indices is used instead

• **index_position** – (default: 'up') determines the position of the indices labelling the linear forms of the coframe; can be either 'down' or 'up'

• **include_domain** – (default: True) boolean determining whether the name of the domain is included in the beginning of the coframe name

**EXAMPLES:**
```python
sage: M = Manifold(3, 'M', structure='top')
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e').coframe(); e
Local coframe (E|_M, (e^0,e^1))
sage: e.set_name('f'); e
Local coframe (E|_M, (f^0,f^1))
sage: e.set_name('e', latex_symbol=r'\epsilon')
sage: latex(e)
\left(E|_{M}, \left(\epsilon^{0},\epsilon^{1}\right)\right)
sage: e.set_name(e, include_domain=False); e
Local coframe (e^0,e^1)
sage: latex(e)
\left(E|_{M}, \left(\epsilon^{0},\epsilon^{1}\right)\right)
sage: e.set_name([a, b], latex_symbol=[r'\alpha, r'\beta ]); e
Local coframe (E|_M, (a,b))
sage: latex(e)
\left(E|_{M}, \left(\alpha,\beta\right)\right)
sage: e.set_name(e, indices=[x,y],
....: latex_indices=[r'\xi, r'\zeta ]); e
Local coframe (E|_M, (e^x,e^y))
sage: latex(e)
\left(E|_{M}, \left(e^{\xi},e^{\zeta}\right)\right)
```

**class** `sage.manifolds.local_frame.LocalFrame` *(section_module, symbol, latex_symbol=None, indices=None, latex_indices=None, symbol_dual=None, latex_symbol_dual=None)*

Bases: `FreeModuleBasis`

Local frame on a vector bundle.

A local frame on a vector bundle $E \to M$ of class $C^k$ is a local section $(e_1, \ldots, e_n) : U \to E^n$ of class $C^k$
defined on some subset $U$ of the base space $M$, such that $e(p)$ is a basis of the fiber $E_p$ for any $p \in U$.

For each instantiation of a local frame, a local coframe is automatically created, as an instance of the class `LocalCoFrame`. It is returned by the method `coframe()`.

**INPUT:**

- `section_module` – free module of local sections over $U$ in the given vector bundle $E \to M$
- `symbol` – either a string, to be used as a common base for the symbols of the local sections constituting the local frame, or a tuple of strings, representing the individual symbols of the local sections
- `latex_symbol` – (default: `None`) either a string, to be used as a common base for the LaTeX symbols of the local sections constituting the local frame, or a tuple of strings, representing the individual LaTeX symbols of the local sections; if `None`, symbol is used in place of `latex_symbol`
- `indices` – (default: `None`; used only if `symbol` is a single string) tuple of strings representing the indices labelling the local sections of the frame; if `None`, the indices will be generated as integers within the range declared on the local frame’s domain
- `latex_indices` – (default: `None`) tuple of strings representing the indices for the LaTeX symbols of the local sections; if `None`, indices is used instead
- `symbol_dual` – (default: `None`) same as symbol but for the dual coframe; if `None`, symbol must be a string and is used for the common base of the symbols of the elements of the dual coframe
- `latex_symbol_dual` – (default: `None`) same as `latex_symbol` but for the dual coframe

**EXAMPLES:**

Defining a local frame on a 3-dimensional vector bundle over a 3-dimensional manifold:
sage: M = Manifold(3, 'M', start_index=1, structure='top')
sage: E = M.vector_bundle(3, 'E')
sage: e = E.local_frame('e'); e
Local frame (E|_M, (e_1,e_2,e_3))
sage: latex(e)
\left(E|_{M}, \left(e_{1},e_{2},e_{3}\right)\right)

The individual elements of the vector frame are accessed via square brackets, with the possibility to invoke the slice operator `:` to get more than a single element:

sage: e[2]
Section e_2 on the 3-dimensional topological manifold M with values in 
the real vector bundle E of rank 3
sage: e[1:3]
(Section e_1 on the 3-dimensional topological manifold M with values in 
the real vector bundle E of rank 3, 
Section e_2 on the 3-dimensional topological manifold M with values in 
the real vector bundle E of rank 3)
sage: e[:]
(Section e_1 on the 3-dimensional topological manifold M with values in 
the real vector bundle E of rank 3, 
Section e_2 on the 3-dimensional topological manifold M with values in 
the real vector bundle E of rank 3, 
Section e_3 on the 3-dimensional topological manifold M with values in 
the real vector bundle E of rank 3)

The \LaTeX\ symbol can be specified:

sage: eps = E.local_frame('eps', latex_symbol=r'\epsilon')
sage: latex(eps)
\left(E|_{M}, \left(\epsilon_{1},\epsilon_{2},\epsilon_{3}\right)\right)

By default, the elements of the local frame are labelled by integers within the range specified at the manifold declaration. It is however possible to fully customize the labels, via the argument `indices`:

sage: u = E.local_frame('u', indices=('x', 'y', 'z')) ; u
Local frame (E|_M, (u_x,u_y,u_z))
sage: u[1]
Section u_x on the 3-dimensional topological manifold M with values in 
the real vector bundle E of rank 3
sage: u.coframe()
Local coframe (E|_M, (u^x,u^y,u^z))

The \LaTeX\ format of the indices can be adjusted:

sage: v = E.local_frame('v', indices=('a', 'b', 'c'), 
....: latex_indices=(r'\alpha', r'\beta', r'\gamma'))
sage: v
Local frame (E|_M, (v_a,v_b,v_c))
sage: latex(v)
\left(E|_{M}, \left(v_{\alpha},v_{\beta},v_{\gamma}\right)\right)
sage: latex(v.coframe())
\left(E|_{M}, \left(v^{\alpha},v^{\beta},v^{\gamma}\right)\right)

The symbol of each element of the local frame can also be freely chosen, by providing a tuple of symbols as the first argument of `local_frame`; it is then mandatory to specify as well some symbols for the dual coframe:
Local frames are bases of free modules formed by local sections:

sage: N = Manifold(2, 'N', structure='top', start_index=1)
sage: X.<x,y> = N.chart()
sage: U = N.open_subset('U')
sage: F = N.vector_bundle(2, 'F')
sage: f = F.local_frame('f', domain=U)
sage: f.module()
Free module C^0(U;F) of sections on the Open subset U of the 2-dimensional topological manifold N with values in the real vector bundle F of rank 2
sage: f.module().base_ring()
Algebra of scalar fields on the Open subset U of the 2-dimensional topological manifold N
sage: f.module() is F.section_module(domain=f.domain())
True
sage: f in F.section_module(domain=U).bases()
True

The value of the local frame at a given point is a basis of the corresponding fiber:

sage: X_U = X.restrict(U) # We need coordinates on the subset
sage: p = N((0,1), name='p'); p
Point p on the 2-dimensional topological manifold N
sage: f.at(p)
Basis (f_1,f_2) on the Fiber of F at Point p on the 2-dimensional topological manifold N

at (point)
Return the value of self at a given point, this value being a basis of the vector bundle fiber at the point.

INPUT:
• point = ManifoldPoint; point p in the domain U of the local frame (denoted e hereafter)

OUTPUT:
• FreeModuleBasis representing the basis e(p) of the vector bundle fiber \( E_p \)

EXAMPLES:
Basis of a fiber of a trivial vector bundle:

sage: M = Manifold(2, 'M', structure='top')
sage: X.<x,y> = M.chart()
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e'); e
Local frame (E|_M, (e_0,e_1))

sage: p = M.point((-1,2), name='p')
sage: ep = e.at(p) ; ep
Basis (e_0,e_1) on the Fiber of E at Point p on the 2-dimensional topological manifold M

sage: type(ep)
<class 'sage.tensor.modules.free_module_basis.FreeModuleBasis_with_category'>
sage: ep[0]
Vector e_0 in the fiber of E at Point p on the 2-dimensional topological manifold M

sage: ep[1]
Vector e_1 in the fiber of E at Point p on the 2-dimensional topological 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 fiber at p:

sage: Ep = E.fiber(p)
sage: Ep.bases()
[Basis (e_0,e_1) on the Fiber of E at Point p on the 2-dimensional topological manifold M]

Let us consider another local frame:

sage: aut = E.section_module().automorphism()
sage: aut[: ] = [[[1+y^2, 0], [0, 2]]
sage: f = e.new_frame(aut, 'f') ; f
Local frame (E|_M, (f_0,f_1))
sage: fp = f.at(p) ; fp
Basis (f_0,f_1) on the Fiber of E at Point p on the 2-dimensional topological manifold M

There are now two bases on the fiber:

sage: Ep.bases()
[Basis (e_0,e_1) on the Fiber of E at Point p on the 2-dimensional topological manifold M,
Basis (f_0,f_1) on the Fiber of E at Point p on the 2-dimensional topological manifold M]

Moreover, the changes of bases in the tangent space have been computed from the known relation between the frames e and f (via the automorphism aut defined above):

sage: Ep.change_of_basis(ep, fp)
Automorphism of the Fiber of E at Point p on the 2-dimensional topological manifold M

sage: Ep.change_of_basis(ep, fp).display()
5 e_0⊗e^0 + 2 e_1⊗e^1

sage: Ep.change_of_basis(fp, ep)
Automorphism of the Fiber of E at Point p on the 2-dimensional topological manifold M

sage: Ep.change_of_basis(fp, ep).display()
1/5 e_0⊗e^0 + 1/2 e_1⊗e^1

The dual bases:
```python
sage: e.coframe()
Local coframe (E|_M, (e^0,e^1))
sage: ep.dual_basis()
Dual basis (e^0,e^1) on the Fiber of E at Point p on the 2-dimensional topological manifold M
sage: ep.dual_basis() is e.coframe().at(p)
True
sage: f.coframe()
Local coframe (E|_M, (f^0,f^1))
sage: fp.dual_basis()
Dual basis (f^0,f^1) on the Fiber of E at Point p on the 2-dimensional topological manifold M
sage: fp.dual_basis() is f.coframe().at(p)
True
```

**base_space()**

Return the base space on which the overlying vector bundle is defined.

**EXAMPLES:**

```python
sage: M = Manifold(3, 'M', structure='top')
sage: U = M.open_subset('U')
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e', domain=U)
sage: e.base_space()
3-dimensional topological manifold M
```

**coframe()**

Return the coframe of self.

**EXAMPLES:**

```python
sage: M = Manifold(2, 'M', structure='top')
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e'); e
Local frame (E|_M, (e_0,e_1))
sage: e.coframe()
Local coframe (E|_M, (e^0,e^1))
```

**domain()**

Return the domain on which self is defined.

**EXAMPLES:**

```python
sage: M = Manifold(3, 'M', structure='top')
sage: U = M.open_subset('U')
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e', domain=U); e
Local frame (E|_M, (e_0,e_1))
sage: e.domain()
Open subset U of the 3-dimensional topological manifold M
```

**new_frame**(change_of_frame, symbol, latex_symbol=None, indices=None, latex_indices=None, symbol_dual=None, latex_symbol_dual=None)

Define a new local frame from self.

The new local frame is defined from vector bundle automorphisms; its module is the same as that of the current frame.
INPUT:

- `change_of_frame` - A `FreeModuleAutomorphism` representing a vector bundle automorphism \( P \) that relates the current frame \((e_i)\) to the new frame \((f_i)\) according to \( f_i = P(e_i) \)

- `symbol` - An optional parameter; either a string, to be used as a common base for the symbols of the sections constituting the local frame, or a list/tuple of strings, representing the individual symbols of the sections

- `latex_symbol` - An optional parameter; if `None`, symbol is used in place of `latex_symbol` for the symbols of the sections

- `indices` - A tuple of strings representing the indices labelling the sections of the frame; if `None`, the indices will be generated as integers within the range declared on `self`

- `latex_indices` - A tuple of strings representing the indices for the LaTeX symbols of the sections; if `None`, `indices` is used instead

- `symbol_dual` - An optional parameter; same as `symbol` but for the dual coframe

- `latex_symbol_dual` - An optional parameter; same as `latex_symbol` but for the dual coframe

OUTPUT:

- The new frame \((f_i)\), as an instance of `LocalFrame`

EXAMPLES:

Orthogonal transformation of a frame on the 2-dimensional trivial vector bundle over the Euclidean plane:

```python
sage: M = Manifold(2, 'R^2', structure='top', start_index=1)
sage: c_cart.<x,y> = M.chart()
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e'); e
Local frame (E|_R^2, (e_1,e_2))
sage: orth = E.section_module().automorphism()
sage: orth[:]=[[sqrt(3)/2, -1/2], [1/2, sqrt(3)/2]]
sage: f = e.new_frame(orth, 'f')
sage: f[1][:]
[1/2*sqrt(3), 1/2]
sage: f[2][:]
[-1/2, 1/2*sqrt(3)]
sage: a = E.change_of_frame(e,f)
sage: a[:]
[1/2*sqrt(3) -1/2]
[ 1/2 1/2*sqrt(3)]
sage: a == orth
True
sage: a is orth
False
sage: a._components # random (dictionary output)
{Local frame (E|_D_0, (e_1,e_2)): 2-indices components w.r.t. Local frame (E|_D_0, (e_1,e_2)),
 Local frame (E|_D_0, (f_1,f_2)): 2-indices components w.r.t. Local frame (E|_D_0, (f_1,f_2))}
sage: a.comp(f)[:]
[1/2*sqrt(3) -1/2]
[ 1/2 1/2*sqrt(3)]
sage: a1 = E.change_of_frame(f,e)
```

(continues on next page)
```python
sage: a1[:]
[1/2*sqrt(3) 1/2]
[-1/2  1/2*sqrt(3)]
sage: a1 == orth.inverse()
True
sage: a1 is orth.inverse()
False
sage: e[1].comp(f)[:]
[1/2*sqrt(3), -1/2]
sage: e[2].comp(f)[:]
[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 `LocalFrame`

**EXAMPLES:**

Restriction of a frame defined on $\mathbb{R}^2$ to the unit disk:

```python
sage: M = Manifold(2, 'R^2', structure='top', start_index=1)
sage: c_cart.<x,y> = M.chart() # Cartesian coordinates on R^2
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e'); e
Local frame (E|_R^2, (e_1,e_2))
sage: a = E.section_module().automorphism()
sage: a[:] = [[1-y^2,0], [ 1+x^2, 2]]
sage: f = e.new_frame(a, 'f'); f
Local frame (E|_R^2, (f_1,f_2))
sage: U = M.open_subset('U', coord_def ={c_cart: x^2+y^2<1})
sage: e_U = e.restrict(U); e_U
Local frame (E|_U, (e_1,e_2))
sage: f_U = f.restrict(U) ; f_U
Local frame (E|_U, (f_1,f_2))
```

The vectors of the restriction have the same symbols as those of the original frame:

```python
sage: f_U[1].display()
f_1 = (-y^2 + 1) e_1 + (x^2 + 1) e_2
sage: f_U[2].display()
f_2 = 2 e_2
```

Actually, the components are the restrictions of the original frame vectors:

```python
sage: f_U[1] is f[1].restrict(U)
True
sage: f_U[2] is f[2].restrict(U)
True
```
set_name (symbol, latex_symbol=None, indices=None, latex_indices=None, index_position='down', include_domain=True)

Set (or change) the text name and LaTeX name of self.

INPUT:

- **symbol** – either a string, to be used as a common base for the symbols of the local sections constituting the local frame, or a list/tuple of strings, representing the individual symbols of the local sections
- **latex_symbol** – (default: None) either a string, to be used as a common base for the LaTeX symbols of the local sections constituting the local frame, or a list/tuple of strings, representing the individual LaTeX symbols of the local sections; if None, symbol is used in place of latex_symbol
- **indices** – (default: None; used only if symbol is a single string) tuple of strings representing the indices labelling the local sections of the frame; if None, the indices will be generated as integers within the range declared on self
- **latex_indices** – (default: None) tuple of strings representing the indices for the LaTeX symbols of the local sections; if None, indices is used instead
- **index_position** – (default: 'down') determines the position of the indices labelling the local sections of the frame; can be either 'down' or 'up'
- **include_domain** – (default: True) boolean determining whether the name of the domain is included in the beginning of the vector frame name

EXAMPLES:

```
sage: M = Manifold(3, 'M', structure='top')
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e'); e
Local frame (E|_M, (e_0,e_1))
sage: e.set_name(f); e
Local frame (E|_M, (f_0,f_1))
sage: e.set_name(e, include_domain=False); e
Local frame (e_0,e_1)
sage: e.set_name(['a', 'b']); e
Local frame (E|_M, (a,b))
sage: e.set_name('e', indices=['x', 'y']); e
Local frame (E|_M, (e_x,e_y))
sage: e.set_name('e', latex_symbol=r'{\epsilon}')
sage: latex(e)
\left(E|_{\mathcal{M}}, \left(\epsilon_{0},\epsilon_{1}\right)\right)
sage: e.set_name('e', latex_symbol=[r'{\alpha}', r'{\beta}'])
sage: latex(e)
\left(E|_{\mathcal{M}}, \left(\alpha,\beta\right)\right)
sage: e.set_name('e', latex_symbol='E', latex_indices=[r'{\alpha}', r'{\beta}'])
sage: latex(e)
\left(E|_{\mathcal{M}}, \left(E_{\alpha},E_{\beta}\right)\right)
```
class sage.manifolds.local_frame.TrivializationCoFrame(triv_frame, symbol, latex_symbol=None, indices=None, latex_indices=None)

Bases: LocalCoFrame

Trivialization coframe on a vector bundle.

A trivialization coframe is the coframe of the trivialization frame induced by a trivialization (see: TrivializationFrame).

More precisely, a trivialization frame on a vector bundle $E \to M$ of class $C^k$ and rank $n$ over the topological field $K$ and over a topological manifold $M$ is a local coframe induced by a local trivialization $\varphi : E|_U \to U \times K^n$ of the domain $U \subset M$. Namely, the local dual sections

$$\varphi^*e^i := \varphi(\cdot, e^i)$$

on $U$ induce a local frame $(\varphi^*e^1, \ldots, \varphi^*e^n)$, where $(e^1, \ldots, e^n)$ is the dual of the standard basis of $K^n$.

INPUT:

- **triv_frame** – trivialization frame dual to the trivialization coframe
- **symbol** – either a string, to be used as a common base for the symbols of the dual sections constituting the coframe, or a tuple of strings, representing the individual symbols of the dual sections
- **latex_symbol** – (default: None) either a string, to be used as a common base for the LaTeX symbols of the dual sections constituting the coframe, or a tuple of strings, representing the individual LaTeX symbols of the dual sections; if None, symbol is used in place of latex_symbol
- **indices** – (default: None; used only if symbol is a single string) tuple of strings representing the indices labelling the dual sections of the coframe; if None, the indices will be generated as integers within the range declared on the local frame’s domain
- **latex_indices** – (default: None) tuple of strings representing the indices for the LaTeX symbols of the dual sections of the coframe; if None, indices is used instead

EXAMPLES:

Trivialization coframe on a trivial vector bundle of rank 3:

```sage
sage: M = Manifold(3, 'M', start_index=1, structure='top')
sage: X.<x,y,z> = M.chart()  
sage: E = M.vector_bundle(3, 'E')
sage: phi = E.trivialization('phi'); phi
Trivialization (phi, E|_M)

sage: E.frames()
[Trivialization frame (E|_M, ((phi^*e_1),(phi^*e_2),(phi^*e_3)))]

sage: E.coframes()
[Trivialization coframe (E|_M, ((phi^*e^1),(phi^*e^2),(phi^*e^3)))]

sage: f = E.coframes()[0]; f
Trivialization coframe (E|_M, ((phi^*e^1),(phi^*e^2),(phi^*e^3)))
```

The linear forms composing the coframe are obtained via the operator []:
The coframe is the dual of the trivialization frame:

```python
sage: e = phi.frame() ; e
Trivialization frame (E|_U, ((phi_U^*e_1),(phi_U^*e_2)))
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)
```

```python
class sage.manifolds.local_frame.TrivializationFrame(trivialization)
Bases: LocalFrame

Trivialization frame on a topological vector bundle.

A trivialization frame on a topological vector bundle $E \to M$ of rank $n$ over the topological field $K$ and over a topological manifold $M$ is a local frame induced by a local trivialization $\varphi : E|_U \to U \times K^n$ of the domain $U \subseteq M$. More precisely, the local sections

$$\varphi^*e_i := \varphi(\cdot, e_i)$$

on $U$ induce a local frame $(\varphi^*e_1, \ldots, \varphi^*e_n)$, where $(e_1, \ldots, e_n)$ is the standard basis of $K^n$.

INPUT:

- `trivialization` -- the trivialization defined on the vector bundle

EXAMPLES:

```python
sage: M = Manifold(3, 'M')
sage: U = M.open_subset('U')
sage: E = M.vector_bundle(2, 'E')
sage: phi_U = E.trivialization(phi_U, domain=U)
sage: phi_U.frame()
Trivialization frame (E|_U, ((phi_U^*e_1),(phi_U^*e_2)))
sage: latex(phi_U.frame())
\left(E|_{\text{right}}, \left(\left(\text{phi}_U^* e_{- 1}\right)\right)\right)\right)
```
trivialization()

Return the underlying trivialization of self.

EXAMPLES:

```python
sage: M = Manifold(3, 'M')
sage: U = M.open_subset('U')
sage: E = M.vector_bundle(2, 'E')
sage: phi_U = E.trivialization('phi_U', domain=U)
sage: e = phi_U.frame()
sage: e.trivialization()
Trivialization (phi_U, E|_U)
```

1.9.6 Section Modules

The set of sections over a vector bundle $E \to M$ of class $C^k$ on a domain $U \in M$ is a module over the algebra $C^k(U)$ of scalar fields on $U$.

Depending on the domain, there are two classes of section modules:

- `SectionModule` for local sections over a non-trivial part of a topological vector bundle
- `SectionFreeModule` for local sections over a trivial part of a topological vector bundle

AUTHORS:

- Michael Jung (2019): initial version

```python
class sage.manifolds.section_module.SectionFreeModule(vbundle, domain)
```

Bases: `FiniteRankFreeModule`

Free module of sections over a vector bundle $E \to M$ of class $C^k$ on a domain $U \in M$ which admits a trivialization or local frame.

The section module $C^k(U; E)$ is the set of all $C^k$-maps, called sections, of type

$$s : U \rightarrow E$$

such that

$$\forall p \in U, \ s(p) \in E_p,$$

where $E_p$ is the vector bundle fiber of $E$ at the point $p$.

Since the domain $U$ admits a local frame, the corresponding vector bundle $E|_U \to U$ is trivial and $C^k(U; E)$ is a free module over $C^k(U)$.

**Note:** If $E|_U$ is not trivial, the class `SectionModule` should be used instead, for $C^k(U; E)$ is no longer a free module.

**INPUT:**

- `vbundle` – vector bundle $E$ on which the sections takes its values
- `domain` – (default: `None`) subdomain $U$ of the base space on which the sections are defined
EXAMPLES:

Module of sections on the 2-rank trivial vector bundle over the Euclidean plane $\mathbb{R}^2$:

```
sage: M = Manifold(2, 'R^2', structure='top')
sage: c_cart.<x,y> = M.chart()
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e')  # Trivializes the vector bundle
sage: C0 = E.section_module(); C0
Free module $C^0(\mathbb{R}^2;E)$ of sections on the 2-dimensional topological
manifold $\mathbb{R}^2$ with values in the real vector bundle $E$ of rank 2
sage: C0.category()
Category of finite dimensional modules over Algebra of scalar fields on
the 2-dimensional topological manifold $\mathbb{R}^2$
sage: C0.base_ring() is M.scalar_field_algebra()
True
```

The vector bundle admits a global frame and is therefore trivial:

```
sage: E.is_manifestly_trivial()
True
```

Since the vector bundle is trivial, its section module of global sections is a free module:

```
sage: isinstance(C0, FiniteRankFreeModule)
True
```

Some elements are:

```
sage: C0.an_element().display()
2 e_0 + 2 e_1
sage: C0.zero().display()
zero = 0
sage: s = C0([-y,x]); s
Section on the 2-dimensional topological manifold $\mathbb{R}^2$ with values in the
real vector bundle $E$ of rank 2
sage: s.display()
-y e_0 + x e_1
```

The rank of the free module equals the rank of the vector bundle:

```
sage: C0.rank()
2
```

The basis is given by the definition above:

```
sage: C0.bases()
[Local frame $(E|_{\mathbb{R}^2}, (e_0,e_1))$]
```

The test suite is passed as well:

```
sage: TestSuite(C0).run()
```

```
Element
   alias of TrivialSection
base_space()
   Return the base space of the sections in this module.
```

1.9. Topological Vector Bundles
EXAMPLES:

```python
sage: M = Manifold(3, 'M', structure='top')
sage: U = M.open_subset('U')
sage: E = U.vector_bundle(2, 'E')
sage: C0 = E.section_module(force_free=True); C0
Free module C^0(U;E) of sections on the Open subset U of the
3-dimensional topological manifold M with values in the real
vector bundle E of rank 2
sage: C0.base_space()
Open subset U of the 3-dimensional topological manifold M
```

`basis` *(symbol=None, latex_symbol=None, from_frame=None, indices=None, latex_indices=None, symbol_dual=None, latex_symbol_dual=None)*

Define a basis of `self`.

A basis of the section module is actually a local frame on the differentiable manifold `U` over which the section 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` – (default: `None`) either a string, to be used as a common base for the symbols of the elements of the basis, or a tuple of strings, representing the individual symbols of the elements of the basis
- `latex_symbol` – (default: `None`) either a string, to be used as a common base for the LaTeX symbols of the elements of the basis, or a tuple of strings, representing the individual LaTeX symbols of the elements of the basis; if `None`, `symbol` is used in place of `latex_symbol`
- `indices` – (default: `None`; used only if `symbol` is a single string) tuple of strings representing the indices labelling the elements of the basis; if `None`, the indices will be generated as integers within the range declared on `self`
- `latex_indices` – (default: `None`) tuple of strings representing the indices for the LaTeX symbols of the elements of the basis; if `None`, `indices` is used instead
- `symbol_dual` – (default: `None`) same as `symbol` but for the dual basis; if `None`, `symbol` must be a string and is used for the common base of the symbols of the elements of the dual basis
- `latex_symbol_dual` – (default: `None`) same as `latex_symbol` but for the dual basis

**OUTPUT:**

- a `LocalFrame` representing a basis on `self`

**EXAMPLES:**

```python
sage: M = Manifold(2, 'M', structure='top')
sage: E = M.vector_bundle(2, 'E')
sage: C0 = E.section_module(force_free=True)
sage: e = C0.basis('e'); e
Local frame (E|_M, (e_0,e_1))
```

See `LocalFrame` for more examples and documentation.

`default_frame()`

Return the default basis of the free module `self`.

296 Chapter 1. Topological Manifolds
The *default basis* is simply a basis whose name can be skipped in methods requiring a basis as an argument. By default, it is the first basis introduced on the module. It can be changed by the method `set_default_basis()`.

**OUTPUT:**

- instance of `FreeModuleBasis`

**EXAMPLES:**

At the module construction, no default basis is assumed:

```sage`
M = FiniteRankFreeModule(ZZ, 2, name='M', start_index=1)
M.default_basis()
```

No default basis has been defined on the Rank-2 free module `M` over the Integer Ring

The first defined basis becomes the default one:

```sage`
e = M.basis('e'); e
```

Basis \((e_1,e_2)\) on the Rank-2 free module `M` over the Integer Ring

```sage`
f = M.basis('f'); f
```

Basis \((f_1,f_2)\) on the Rank-2 free module `M` over the Integer Ring

```sage`
f = M.default_basis()
```

Basis \((e_1,e_2)\) on the Rank-2 free module `M` over the Integer Ring

**domain()**

Return the domain of the section module.

**EXAMPLES:**

```sage`
M = Manifold(3, 'M', structure='top')
U = M.open_subset('U')
E = M.vector_bundle(2, 'E')
C0_U = E.section_module(domain=U, force_free=True); C0_U
```

Free module \(C^0(U;E)\) of sections on the Open subset `U` of the 3-dimensional topological manifold `M` with values in the real vector bundle `E` of rank 2

```sage`
C0_U.domain()
```

Open subset `U` of the 3-dimensional topological manifold `M`

**set_default_frame(basis)**

Sets the default basis of `self`.

The *default basis* is simply a basis whose name can be skipped in methods requiring a basis as an argument. By default, it is the first basis introduced on the module.

**INPUT:**

- `basis` – instance of `FreeModuleBasis` representing a basis on `self`

**EXAMPLES:**

Changing the default basis on a rank-3 free module:

```sage`
M = FiniteRankFreeModule(ZZ, 3, name='M', start_index=1)
e = M.basis('e'); e
Basis \((e_1,e_2,e_3)\) on the Rank-3 free module `M` over the Integer Ring
```
vector_bundle()

Return the overlying vector bundle on which the section module is defined.

EXAMPLES:

```python
sage: M = Manifold(3, 'M', structure='top')
sage: E = M.vector_bundle(2, 'E')
sage: C0 = E.section_module(force_free=True); C0
Free module \(C^0(M;E)\) of sections on the 3-dimensional topological manifold \(M\) with values in the real vector bundle \(E\) of rank 2
sage: C0.vector_bundle()
Topological real vector bundle \(E \to M\) of rank 2 over the base space
3-dimensional topological manifold \(M\)
```

class sage.manifolds.section_module.SectionModule(vbundle, domain)

Module of sections over a vector bundle \(E \to M\) of class \(C^k\) on a domain \(U \in M\).

The section module \(C^k(U;E)\) is the set of all \(C^k\)-maps, called sections, of type

\[
s : U \longrightarrow E
\]

such that

\[
\forall p \in U, \ s(p) \in E_p,
\]

where \(E_p\) is the vector bundle fiber of \(E\) at the point \(p\). \(C^k(U;E)\) is a module over \(C^k(U)\), the algebra of \(C^k\) scalar fields on \(U\).

INPUT:

- `vbundle` – vector bundle \(E\) on which the sections takes its values
- `domain` – (default: None) subdomain \(U\) of the base space on which the sections are defined

EXAMPLES:

Module of sections on the Möbius bundle:

```python
sage: M = Manifold(1, 'RP^1', structure='top', start_index=1)
sage: U = M.open_subset('U')  # the complement of one point
sage: c_u.<u> = U.chart()  # [1:u] in homogeneous coord.
sage: V = M.open_subset('V')  # the complement of the point u=0
sage: M.declare_union(U,V)  # [v:1] in homogeneous coord.
sage: c_v.<v> = V.chart()
sage: u_to_v = c_u.transition_map(c_v, (1/u),
....: intersection_name='W',
....: restrictions1 = u!=0,
```
Manifolds, Release 10.3

(continued from previous page)

sage: v_to_u = u_to_v.inverse()

sage: Ω = U.intersection(V)

sage: E = M.vector_bundle(1, 'E')

sage: ϕ_U = E.trivialization('ϕ_U', latex_name=r'$\varphi_U$',
                           domain=U)

sage: ϕ_V = E.trivialization('ϕ_V', latex_name=r'$\varphi_V$',
                           domain=V)

sage: transf = ϕ_U.transition_map(ϕ_V, [[u]])

Module $C^0(\mathbb{RP}^1;E)$ of sections on the 1-dimensional topological manifold $\mathbb{RP}^1$ with values in the real vector bundle $E$ of rank 1

$C^0(\mathbb{RP}^1;E)$ is a module over the algebra $C^0(\mathbb{RP}^1)$:

```
sage: C0.category()
Category of modules over Algebra of scalar fields on the 1-dimensional topological manifold $\mathbb{RP}^1$
```

```
sage: C0.base_ring() is M.scalar_field_algebra()
True
```

However, $C^0(\mathbb{RP}^1;E)$ is not a free module:

```
sage: isinstance(C0, FiniteRankFreeModule)
False
```

since the Möbius bundle is not trivial:

```
sage: E.is_manifestly_trivial()
False
```

The section module over $U$, on the other hand, is a free module since $E|_U$ admits a trivialization and therefore has a local frame:

```
sage: C0_U = E.section_module(domain=U)
sage: isinstance(C0_U, FiniteRankFreeModule)
True
```

The zero element of the module:

```
sage: z = C0.zero(); z
Section zero on the 1-dimensional topological manifold $\mathbb{RP}^1$ with values in the real vector bundle $E$ of rank 1
```

```
sage: z.display(ϕ_U.frame())
zero = 0
```

```
sage: z.display(ϕ_V.frame())
zero = 0
```

The module $C^0(M;E)$ coerces to any module of sections defined on a subdomain of $M$, for instance $C^0(U;E)$:

```
sage: C0_U.has_coerce_map_from(C0)
True
```

```
sage: C0_U.coerce_map_from(C0)
Coercion map:
  From: Module $C^0(\mathbb{RP}^1;E)$ of sections on the 1-dimensional topological manifold $\mathbb{RP}^1$ with values in the real vector bundle $E$ of rank 1
(continues on next page)
To: Free module $C^0(U;E)$ of sections on the Open subset $U$ of the 1-dimensional topological manifold $\mathbb{RP}^1$ with values in the real vector bundle $E$ of rank 1

The conversion map is actually the restriction of sections defined on $M$ to $U$.

**Element**

alias of `Section`

**base_space()**

Return the base space of the sections in this module.

**EXAMPLES:**

```sage
sage: M = Manifold(3, 'M', structure='top')
sage: U = M.open_subset('U')
sage: E = U.vector_bundle(2, 'E')
sage: C0 = E.section_module(); C0
Module $C^0(U;E)$ of sections on the Open subset $U$ of the 3-dimensional topological manifold $M$ with values in the real vector bundle $E$ of rank 2
sage: C0.base_space()
Open subset $U$ of the 3-dimensional topological manifold $M$
```

**default_frame()**

Return the default frame defined on `self`.

**EXAMPLES:**

Get the default local frame of a non-trivial section module:

```sage
sage: M = Manifold(3, 'M', structure='top')
sage: U = M.open_subset('U')
sage: V = M.open_subset('V')
sage: M.declare_union(U, V)
sage: E = M.vector_bundle(2, 'E')
sage: C0 = E.section_module()
sage: e = E.local_frame('e', domain=U)
sage: C0.default_frame()
Local frame $(E|_U, (e_0, e_1))$
```

The local frame is indeed the same, and not a copy:

```sage
sage: e is C0.default_frame()
True
```

**domain()**

Return the domain of the section module.

**EXAMPLES:**

```sage
sage: M = Manifold(3, 'M', structure='top')
sage: U = M.open_subset('U')
sage: E = M.vector_bundle(2, 'E')
sage: C0_U = E.section_module(domain=U); C0_U
Module $C^0(U;E)$ of sections on the Open subset $U$ of the 3-dimensional topological manifold $M$ with values in the real vector bundle $E$ of rank 2
```
set_default_frame (basis)
Set the default local frame on self.

EXAMPLES:
Set a default frame of a non-trivial section module:

```python
sage: M = Manifold(3, 'M', structure='top')
sage: U = M.open_subset('U')
sage: V = M.open_subset('V')
sage: M.declare_union(U, V)
sage: E = M.vector_bundle(2, 'E')
sage: C0 = E.section_module(); C0
Module C^0(M;E) of sections on the 3-dimensional topological manifold M with values in the real vector bundle E of rank 2
sage: e = E.local_frame('e', domain=U)
sage: C0.set_default_frame(e)
sage: C0.default_frame()
Local frame (E|_U, (e_0,e_1))
```

The local frame is indeed the same, and not a copy:

```python
sage: e is C0.default_frame()
True
```

Notice, that the local frame is defined on a subset and is not part of the section module $C^0(M;E)$:

```python
sage: C0.default_frame().domain()
Open subset U of the 3-dimensional topological manifold M
```

vector_bundle()
Return the overlying vector bundle on which the section module is defined.

EXAMPLES:

```python
sage: M = Manifold(3, 'M', structure='top')
sage: E = M.vector_bundle(2, 'E')
sage: C0 = E.section_module(); C0
Module C^0(M;E) of sections on the 3-dimensional topological manifold M with values in the real vector bundle E of rank 2
sage: C0.vector_bundle()
Topological real vector bundle E -> M of rank 2 over the base space 3-dimensional topological manifold M
sage: E is C0.vector_bundle()
True
```

zero()
Return the zero of self.

EXAMPLES:

```python
sage: M = Manifold(2, 'M', structure='top')
sage: X.<x,y> = M.chart()
```
1.9.7 Sections

The class *Section* implements sections on vector bundles. The derived class *TrivialSection* is devoted to sections on trivial parts of a vector bundle.

AUTHORS:

- Michael Jung (2019): initial version

```python
class sage.manifolds.section.Section(section_module, name=None, latex_name=None)

Bases: ModuleElementWithMutability

Section in a vector bundle.

An instance of this class is a section in a vector bundle $E \rightarrow M$ of class $C^k$, where $E | U$ is not manifestly trivial. More precisely, a *(local)* section on a subset $U \in M$ is a map of class $C^k$

$$s : U \rightarrow E$$

such that

$$\forall p \in U, s(p) \in E_p$$

where $E_p$ denotes the vector bundle fiber of $E$ over the point $p \in U$.

If $E | U$ is trivial, the class *TrivialSection* should be used instead.

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

INPUT:

- *section_module* – module $C^k(U; E)$ of sections on $E$ over $U$ (cf. *SectionModule*)
- *name* – (default: None) name given to the section
- *latex_name* – (default: None) LaTeX symbol to denote the section; if none is provided, the LaTeX symbol is set to *name*

EXAMPLES:

A section on a non-trivial rank 2 vector bundle over a non-trivial 2-manifold:

```python
sage: M = Manifold(2, 'M', structure='top')
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),
\quad: intersection_name='W', restrictions1= x>0,
\quad: restrictions2= u+v>0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: W = U.intersection(V)
```
Manifolds, Release 10.3

(continued from previous page)

sage: E = M.vector_bundle(2, 'E')  # define the vector bundle
sage: phi_U = E.trivialization('phi_U', domain=U)  # define trivializations
sage: phi_V = E.trivialization('phi_V', domain=V)
sage: transf = phi_U.transition_map(phi_V, [[0,x],[x,0]])  # transition map between trivializations
sage: EU = phi_U.frame(); EV = phi_V.frame()  # define induced frames
sage: s = E.section(name='s'); s
Section s on the 2-dimensional topological manifold M with values in the real vector bundle E of rank 2

The parent of s is not a free module, since E is not trivial:

sage: isinstance(s.parent(), FiniteRankFreeModule)
False

To fully define s, we have to specify its components in some local frames defined on the trivial parts of E. The components consist of scalar fields defined on the corresponding domain. Let us start with E|U:

sage: s[fU,:,:] = [x^2, 1-y]
sage: s.display(fU)
s = x^2 (phi_U^*e_1) + (-y + 1) (phi_U^*e_2)

To set the components of s on V consistently, we copy the expressions of the components in the common subset W:

sage: fUW = fU.restrict(W); fVW = fV.restrict(W)
sage: c_uvW = c_uv.restrict(W)
sage: s[fV,0] = s[fVW,0,c_uvW].expr()  # long time
sage: s[fV,1] = s[fVW,1,c_uvW].expr()  # long time

Actually, the operation above can be performed in a single line by means of the method add_comp_by_continuation():

sage: s.add_comp_by_continuation(fV, W, chart=c_uv)

At this stage, s is fully defined, having components in frames fU and fV and the union of the domains of fU and fV being the whole manifold:

sage: s.display(fV)
s = (-1/4*u^2 + 1/4*v^2 + 1/2*u + 1/2*v) (phi_V^*e_1)
 + (1/8*u^3 + 3/8*u^2*v + 3/8*u*v^2 + 1/8*v^3) (phi_V^*e_2)

Sections can be pointwisely added:

sage: t = E.section([x,y], frame=fU, name='t'); t
Section t on the 2-dimensional topological manifold M with values in the real vector bundle E of rank 2
sage: t.add_comp_by_continuation(fV, W, chart=c_uv)
sage: t.display(fV)
t = (1/4*u^2 - 1/4*v^2) (phi_V^*e_1) + (1/4*u^2 + 1/2*u*v + 1/4*v^2) (phi_V^*e_2)
sage: a = s + t; a
Section a+s+t on the 2-dimensional topological manifold M with values in the real vector bundle E of rank 2
sage: a.display(fU)
s+t = (x^2 + x) (phi_U^*e_1) + (phi_U^*e_2)
sage: a.display(fV)

(continues on next page)
and multiplied by scalar fields:

\[
\begin{align*}
\text{sage: } & f = M.\text{scalar_field}(y^2-x^2, \text{name}='f') \\
\text{sage: } & f.\text{add_expr_by_continuation}(c_{uv}, W) \\
\text{sage: } & f.\text{display()} \\
& f: M \to \mathbb{R} \\
& \text{on } U: (x, y) \mapsto -x^2 + y^2 \\
& \text{on } V: (u, v) \mapsto -u v \\
\text{sage: } & b = f*s; b \\
\text{sage: } & b.\text{display()} \\
& f*s = (-x^4 + x^2 y^2) (\phi_U^*e_1) + (x^2 y - y^3 - x^2 + y^2) (\phi_U^*e_2) \\
\text{sage: } & b.\text{display()} \\
& f*s = (-1/4 u v^3 - 1/2 u v^2 + 1/4 (u^3 - 2 u^2) v) (\phi_V^*e_1) \\
& + (1/8 u^4 v^2 + 3/8 u^3 v^3 + 1/4 u v^4) (\phi_V^*e_2)
\end{align*}
\]

The domain on which the section should be defined, can be stated via the \texttt{domain} option in \texttt{section()}:

\[
\begin{align*}
\text{sage: } & cU = E.\text{section}([1,x], \text{domain}=U, \text{name}='c'); cU \\
\text{sage: } & cU.\text{display()} \\
& c = (\phi_U^*e_1) + x (\phi_U^*e_2)
\end{align*}
\]

Since \(E|_U\) is trivial, \(cU\) now belongs to the free module:

\[
\begin{align*}
\text{sage: } & \text{isinstance}(cU.\text{parent}(), \text{FiniteRankFreeModule}) \\
& \text{True}
\end{align*}
\]

Omitting the \texttt{domain} option, the section is defined on the whole base space:

\[
\begin{align*}
\text{sage: } & c = E.\text{section}(\text{name}='c'); c \\
\text{sage: } & c.\text{display}() \\
& c = (\phi_U^*e_1) + x (\phi_U^*e_2)
\end{align*}
\]

Via \texttt{set_restriction()}, \(cU\) can be defined as the restriction of \(c\) to \(U\):

\[
\begin{align*}
\text{sage: } & c.\text{set_restriction}(cU) \\
\text{sage: } & c.\text{display}() \\
& c = (\phi_U^*e_1) + x (\phi_U^*e_2) \\
\text{sage: } & c.\text{restrict}(U) == cU \\
& \text{True}
\end{align*}
\]

Notice that the zero section is immutable, and therefore its components cannot be changed:

\[
\begin{align*}
\text{sage: } & \text{zer} = E.\text{section_module}().\text{zero()} \\
\text{sage: } & \text{zer.is_immutable()} \\
& \text{True} \\
\text{sage: } & \text{zer.set_comp()} \\
& \text{traceback (most recent call last):} \\
& \ldots \\
& \text{ValueError: the components of an immutable element cannot be changed}
\end{align*}
\]
Other sections can be declared immutable, too:

```python
sage: c.is_immutable()
False
sage: c.set_immutable()
sage: c.is_immutable()
True
sage: c.set_comp()
Traceback (most recent call last):
  ... ValueError: the components of an immutable element cannot be changed
sage: c.set_name('b')
Traceback (most recent call last):
  ... ValueError: the name of an immutable element cannot be changed
```

**add_comp** *(basis=None)*

Return the components of *self* in a given local frame for assignment.

The components with respect to other frames having the same domain as the provided local frame are kept. To delete them, use the method `set_comp()` instead.

**INPUT:**

- basis – (default: None) local frame in which the components are defined; if None, the components are assumed to refer to the section domain’s default frame

**OUTPUT:**

- components in the given frame, as a `Components`; if such components did not exist previously, they are created

**EXAMPLES:**

```python
sage: S2 = Manifold(2, 'S^2', structure='top', start_index=1)
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: stereoN.<x,y> = U.chart()  # stereographic coordinates from the North pole
sage: stereoS.<u,v> = V.chart()  # stereographic coordinates from the South pole
sage: xy_to_uv = 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: W = U.intersection(V)
sage: uv_to_xy = xy_to_uv.inverse()
sage: E = S2.vector_bundle(2, 'E')  # define vector bundle
sage: phi_U = E.trivialization('phi_U', domain=U)  # define trivializations
sage: phi_V = E.trivialization('phi_V', domain=V)
sage: transf = phi_U.transition_map(phi_V, [[0,1],[1,0]])
sage: fN = phi_U.frame(); fS = phi_V.frame()  # get induced frames
sage: s = E.section(name='s')
sage: s.add_comp(fS)
1-index components w.r.t. Trivialization frame (E|_V, ((phi_V^*e_1),(phi_V^*e_2))
```

```python
sage: s.add_comp(fS)[1] = u+v
sage: s.display(fS)
s = (u + v) (phi_V^*e_1)
```
Setting the components in a new frame:

```
sage: e = E.local_frame('e', domain=V)
sage: s.add_comp(e)
1-index components w.r.t. Local frame (E|_V, (e_1,e_2))
sage: s.add_comp(e)[1] = u*v
sage: s.display(e)
s = u*v e_1
```

The components with respect to $fS$ are kept:

```
sage: s.display(fS)
s = (u + v) (phi_V^*e_1)
```

**add_comp_by_continuation**(frame, subdomain, chart=None)

Set components with respect to a local 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 – local 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: S2 = Manifold(2, 'S^2', structure='top', start_index=1)
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: stereoN.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: stereoS.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: xy_to_uv = 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: W = U.intersection(V)
sage: uv_to_xy = xy_to_uv.inverse()
sage: E = S2.vector_bundle(2, 'E') # define vector bundle
sage: phi_U = E.trivialization(phi_U, domain=U) # define trivializations
sage: phi_V = E.trivialization(phi_V, domain=V)
sage: transf = phi_U.transition_map(phi_V, [[0,1],[1,0]])
sage: fN = phi_U.frame(); fS = phi_V.frame() # get induced frames
sage: a = E.section({fN: [x, 2+y]}, name='a')
```

At this stage, the section has been defined only on the open subset $U$ (through its components in the frame $fN$):
The components with respect to the restriction of $f_S$ to the common subdomain $W$, in terms of the $(u, v)$ coordinates, are obtained by a change-of-frame formula on $W$:

$$a = \frac{2u^2 + 2v^2 + v}{u^2 + v^2} (\phi_V^*e_1) + \frac{u}{u^2 + v^2} (\phi_V^*e_2)$$

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 $e_V$ have the same coordinate expression as the above one:

$$a = \frac{2u^2 + 2v^2 + v}{u^2 + v^2} (\phi_V^*e_1) + \frac{u}{u^2 + v^2} (\phi_V^*e_2)$$

and $a$ is defined on the entire manifold $S^2$.

**add_expr_from_subdomain (frame, subdomain)**

Add an expression to an existing component from a subdomain.

**INPUT:**

- frame – local frame $e$ in which the components are to be set
- subdomain – open subset of $e$’s domain in which the components have additional expressions.

**EXAMPLES:**

We are going to consider a section on the trivial rank 2 vector bundle over the 2-sphere:

$$sage: S2 = Manifold(2, 'S^2', structure='top', start_index=1)$$
$$sage: U = S2.open_subset('U') ; V = S2.open_subset('V') # complement of the North and South pole, respectively$$
$$sage: S2.declare_union(U,V)$$
$$sage: stereoN.<x,y> = U.chart() # stereographic coordinates from the North pole$$
$$sage: stereoS.<u,v> = V.chart() # stereographic coordinates from the South pole$$
$$sage: xy_to_uv = stereoN.transition_map(stereoS, (x/(x^2+y^2), y/(x^2+y^2)), intersection_name=W, restrictions1 = x^2+y^2!=0, restrictions2= u^2+v^2!=0)$$
$$sage: W = U.intersection(V)$$
$$sage: uv_to_xy = xy_to_uv.inverse()$$
$$sage: E = S2.vector_bundle(2, 'E') # define vector bundle$$
$$sage: e = E.local_frame('e') # frame to trivialize E$$
$$sage: eU = e.restrict(U); eV = e.restrict(V); eW = e.restrict(W) # this step is essential since U, V and W must be trivial$$

To define a section $s$ on $S^2$, we first set the components on $U$:
But because $E$ is trivial, these components can be extended with respect to the global frame $e$ onto $S^2$:

```
sage: s.add_comp_by_continuation(e, U)
```

One can see that $s$ is not yet fully defined: the components (scalar fields) do not have values on the whole manifold:

```
sage: sorted(s._components.values())[0]._comp[(1,)].display()
S^2 \to \mathbb{R}
on U: (x, y) \mapsto x
on W: (u, v) \mapsto u/(u^2 + v^2)
```

To fix that, we extend the components from $W$ to $V$ first, using `add_comp_by_continuation()`:

```
sage: s.add_comp_by_continuation(eV, W, stereoS)
```

Then, the expression on the subdomain $V$ is added to the components on $S^2$ already known by:

```
sage: s.add_expr_from_subdomain(e, V)
```

The definition of $s$ is now complete:

```
sage: sorted(s._components.values())[0]._comp[(2,)].display()
S^2 \to \mathbb{R}
on U: (x, y) \mapsto y
on V: (u, v) \mapsto v/(u^2 + v^2)
```

Value of `self` at a point of its domain.

If the current section is

$$ s : U \to E, $$

then for any point $p \in U$, $s(p)$ is a vector in the fiber $E_p$ of $E$ at $p$.

**INPUT:**

- `point` - `ManifoldPoint`; point $p$ in the domain of the section $U$

**OUTPUT:**

- `VectorBundleFiberElement` representing the vector $s(p)$ in the fiber $E_p$ of $E$ at $p$.

**EXAMPLES:**

Vector on a rank 2 vector bundle fiber over a non-parallelizable 2-dimensional manifold:
```python
sage: inv = transf.inverse()
sage: W = U.intersection(V)
sage: E = M.vector_bundle(2, 'E') # define vector bundle
dsage: phi_U = E.trivialization('phi_U', domain=U) # define trivializations
dsage: phi_V = E.trivialization('phi_V', domain=V)
sage: transf = phi_U.transition_map(phi_V, [[0,x],[x,0]])
sage: FU = phi_U.frame(); FV = phi_V.frame() # get induced frames
sage: s = E.section({fU: [1+y, x]}, name='s')
sage: s.add_comp_by_continuation(fV, W, chart=c_uv)
sage: s.display(fU)
s = (y + 1) (phi_U^*e_1) + x (phi_U^*e_2)
sage: s.display(fV)
s = (1/4*u^2 + 1/2*u*v + 1/4*v^2) (phi_V^*e_1) + (1/4*u^2 - 1/4*v^2 + 1/2*u + 1/2*v) (phi_V^*e_2)
sage: p = M.point((2,3), chart=c_xy, name='p')
sage: sp = s.at(p) ; sp
Vector s in the fiber of E at Point p on the 2-dimensional topological manifold M
sage: sp.parent()
Fiber of E at Point p on the 2-dimensional topological manifold M
sage: sp.display(fU.at(p))
s = 4 (phi_U^*e_1) + 2 (phi_U^*e_2)
sage: sp.display(fV.at(p))
s = 4 (phi_V^*e_1) + 8 (phi_V^*e_2)
sage: p.coord(c_uv) # to check the above expression
(5, -1)
```

**base_module()**

Return the section module on which `self` acts as a section.

**OUTPUT:**

- instance of `SectionModule`

**EXAMPLES:**

```python
sage: M = Manifold(3, 'M', structure='top')
sage: U = M.open_subset('U')
sage: E = M.vector_bundle(2, 'E')
sage: s = E.section(domain=U)
sage: s.base_module()
Module C^0(U;E) of sections on the Open subset U of the 3-dimensional topological manifold M with values in the real vector bundle E of rank 2
```

**comp** *(basis=None, from_basis=None)*

Return the components in a given local frame.

If the components are not known already, they are computed by the change-of-basis formula from components in another local frame.

**INPUT:**

- **basis** – (default: None) local frame in which the components are required; if none is provided, the components are assumed to refer to the section module’s default frame on the corresponding domain
- **from_basis** – (default: None) local frame from which the required components are computed, via the change-of-basis formula, if they are not known already in the basis `basis`

1.9. Topological Vector Bundles 309
OUTPUT:

• components in the local frame basis, as a Components

EXAMPLES:

Components of a section defined on a rank 2 vector bundle over two open subsets:

```
sage: M = Manifold(2, 'M', structure='top')
sage: X.<x, y> = M.chart()
sage: U = M.open_subset('U'); V = M.open_subset('V')
sage: M.declare_union(U, V)
sage: XU = X.restrict(U); XV = X.restrict(V)
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e', domain=U)
Local frame (E|_U, (e_0,e_1))
sage: f = E.local_frame('f', domain=V)
Local frame (E|_V, (f_0,f_1))
sage: s = E.section(name='s')
sage: s[e, :] = - x + y^3, 2+x
sage: s[f,0] = x^2
sage: s[f,1] = x + y
sage: s.comp(e)
1-index components w.r.t. Local frame (E|_U, (e_0,e_1))
sage: s.comp(e)[:]
[y^3 - x, x + 2]
sage: s.comp(f)
1-index components w.r.t. Local frame (E|_V, (f_0,f_1))
sage: s.comp(f)[:]
[x^2, x + y]
```

Since \( e \) is the default frame of \( E|_U \), the argument \( e \) can be omitted after restricting:

```
sage: e is E.section_module(domain=U).default_frame()
True
sage: s.restrict(U).comp() is s.comp(e)
True
```

**copy** (name=None, latex_name=None)

Return an exact copy of self.

INPUT:

• name – (default: None) name given to the copy

• latex_name – (default: None) LaTeX symbol to denote the copy; if none is provided, the LaTeX symbol is set to name

**Note:** The name and the derived quantities are not copied.

EXAMPLES:

Copy of a section on a rank 2 vector bundle over a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='top')
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: W = U.intersection(V)
sage: E = M.vector_bundle(2, 'E') # define vector bundle
sage: phi_U = E.trivialization('phi_U', domain=U) # define trivializations
sage: phi_V = E.trivialization('phi_V', domain=V)
sage: transf = phi_U.transition_map(phi_V, [[0,x],[x,0]])
sage: fU = phi_U.frame(); fV = phi_V.frame()
sage: s = E.section(name='s')
sage: s[fU,0] = [2, 1-y]
sage: s.add_comp_by_continuation(fV, U.intersection(V), c_uv)
sage: t = s.copy(); t
Section on the 2-dimensional topological manifold M with values in
the real vector bundle E of rank 2
sage: t.display(fU)
2 (phi_U^*e_1) + (-y + 1) (phi_U^*e_2)
sage: t == s
True
If the original section is modified, the copy is not:
sage: s[fU,0] = -1
sage: s.display(fU)
s = -(phi_U^*e_1) + (-y + 1) (phi_U^*e_2)
sage: t.display(fU)
2 (phi_U^*e_1) + (-y + 1) (phi_U^*e_2)
sage: t == s
False

copy_from(other)

Make self a copy of other.

INPUT:

• other — other section, in the same module as self

Note: While the derived quantities are not copied, the name is kept.

Warning: All previous defined components and restrictions will be deleted!

EXAMPLES:

sage: M = Manifold(2, 'M', structure='top')
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: E = M.vector_bundle(2, 'E') # define vector bundle
If the original section is modified, the copy is not:

```
sage: s[fU,0] = -1
sage: s.display(fU)
s = -(phi_U^*e_1) + (-y + 1) (phi_U^*e_2)
sage: t.display(fU)
t = 2 (phi_U^*e_1) + (-y + 1) (phi_U^*e_2)
sage: s == t
False
```

\[ \text{disp (frame} = \text{None, chart} = \text{None)} \]

Display the section in terms of its expansion with respect to a given local frame.

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

INPUT:

- \text{frame} – (default: None) local frame with respect to which the section is expanded; if \text{frame} is None and \text{chart} is not None, the default frame in the corresponding section module is assumed
- \text{chart} – (default: None) chart with respect to which the components of the section in the selected frame are expressed; if None, the default chart of the local frame domain is assumed

EXAMPLES:

Display of section on a rank 2 vector bundle over the 2-sphere:

```
sage: S2 = Manifold(2, 'S^2', structure='top', start_index=1)
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: stereoN.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: stereoS.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: xy_to_uv = 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: W = U.intersection(V)
sage: uv_to_xy = xy_to_uv.inverse()
sage: E = S2.vector_bundle(2, 'E') # define vector bundle
sage: phi_U = E.trivialization('phi_U', domain=U) # define trivializations
sage: phi_V = E.trivialization('phi_V', domain=V)
sage: transf = phi_U.transition_map(phi_V, [[0,x],[x,0]])
sage: fU = phi_U.frame(); fV = phi_V.frame()
sage: s = E.section(name='s')
sage: s[fU,:]= [2, 1-y]
sage: s.add_comp_by_continuation(fV, U.intersection(V), c_uv)
sage: t = E.section(name='t')
sage: t.copy_from(s)
sage: t.display(fU)
t = 2 (phi_U^*e_1) + (-y + 1) (phi_U^*e_2)
sage: s == t
True
```

(continues on next page)
sage: phi_V = E.trivialization('phi_V', domain=V)
sage: transf = phi_U.transition_map(phi_V, [[0,1],[1,0]])
sage: fN = phi_U.frame(); fS = phi_V.frame()  # get induced frames
sage: s = E.section(name='s')
sage: s[fN,:]= [x, y]
sage: s.add_comp_by_continuation(fS, W, stereoS)
sage: s.display(fN)
s = x (phi_U^*e_1) + y (phi_U^*e_2)
sage: s.display(fS)
s = v/(u^2 + v^2) (phi_V^*e_1) + u/(u^2 + v^2) (phi_V^*e_2)

Since fN is the default frame on E|_U, the argument fN can be omitted after restricting:

sage: fN is E.section_module(domain=U).default_frame()
True
sage: s.restrict(U).display()
s = x (phi_U^*e_1) + y (phi_U^*e_2)

Similarly, since fS is V’s default frame, the argument fS can be omitted when considering the restriction of s to V:

sage: s.restrict(V).display()
s = v/(u^2 + v^2) (phi_V^*e_1) + u/(u^2 + v^2) (phi_V^*e_2)

The second argument comes into play whenever the frame’s domain is covered by two distinct charts. Since stereoN.restrict(W) is the default chart on W, the second argument can be omitted for the expression in this chart:

sage: s.display(fS.restrict(W))
s = y (phi_V^*e_1) + x (phi_V^*e_2)

To get the expression in the other chart, the second argument must be used:

sage: s.display(fN.restrict(W), stereoS.restrict(W))
s = u/(u^2 + v^2) (phi_U^*e_1) + v/(u^2 + v^2) (phi_U^*e_2)

One can ask for the display with respect to a frame in which s has not been initialized yet (this will automatically trigger the use of the change-of-frame formula for tensors):

sage: a = E.section_module(domain=U).automorphism()
sage: a[i] = [[1+x^2,0],[0,1+y^2]]
sage: e = fN.new_frame(a, 'e')
sage: [e[i].display() for i in S2.irange()]
[e_1 = (x^2 + 1) (phi_U^*e_1), e_2 = (y^2 + 1) (phi_U^*e_2)]
sage: s.display(e)
s = x/(x^2 + 1) e_1 + y/(y^2 + 1) e_2

A shortcut of display() is disp() :

sage: s.disp(fS)
s = v/(u^2 + v^2) (phi_V^*e_1) + u/(u^2 + v^2) (phi_V^*e_2)

display(frame=None, chart=None)

Display the section in terms of its expansion with respect to a given local frame.
The output is either text-formatted (console mode) or LaTeX-formatted (notebook mode).
INPUT:

- **frame** – (default: None) local frame with respect to which the section is expanded; if **frame** is None and **chart** is not None, the default frame in the corresponding section module is assumed
- **chart** – (default: None) chart with respect to which the components of the section in the selected frame are expressed; if None, the default chart of the local frame domain is assumed

EXAMPLES:

Display of section on a rank 2 vector bundle over the 2-sphere:

```python
sage: S2 = Manifold(2, 'S^2', structure='top', start_index=1)
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: stereoN.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: stereoS.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: xy_to_uv = 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: W = U.intersection(V)
sage: uv_to_xy = xy_to_uv.inverse()
sage: E = S2.vector_bundle(2, 'E') # define vector bundle
sage: phi_U = E.trivialization('phi_U', domain=U) # define trivializations
sage: phi_V = E.trivialization('phi_V', domain=V)
sage: transf = phi_U.transition_map(phi_V, [[0,1],[1,0]])
sage: fN = phi_U.frame(); fS = phi_V.frame() # get induced frames
sage: s = E.section(name=s)
sage: s[fN,:] = [x, y]
sage: s.add_comp_by_continuation(fS, W, stereoS)
sage: s.display(fN)
s = x (phi_U^*e_1) + y (phi_U^*e_2)
sage: s.display(fS)
s = v/(u^2 + v^2) (phi_V^*e_1) + u/(u^2 + v^2) (phi_V^*e_2)
```

Since **fN** is the default frame on **E|_U**, the argument **fN** can be omitted after restricting:

```python
sage: fN is E.section_module(domain=U).default_frame()
 True
sage: s.restrict(U).display()
s = x (phi_U^*e_1) + y (phi_U^*e_2)
```

Similarly, since **fS** is **V**’s default frame, the argument **fS** can be omitted when considering the restriction of **s** to **V**:

```python
sage: s.restrict(V).display()
s = v/(u^2 + v^2) (phi_V^*e_1) + u/(u^2 + v^2) (phi_V^*e_2)
```

The second argument comes into play whenever the frame’s domain is covered by two distinct charts. Since **stereoN.restrict(W)** is the default chart on **W**, the second argument can be omitted for the expression in this chart:

```python
sage: s.display(fS.restrict(W))
s = y (phi_V^*e_1) + x (phi_V^*e_2)
```
To get the expression in the other chart, the second argument must be used:

```sage
sage: s.display(fN.restrict(W), stereoS.restrict(W))
s = u/(u^2 + v^2) (phi_U^*e_1) + v/(u^2 + v^2) (phi_U^*e_2)
```

One can ask for the display with respect to a frame in which \( s \) has not been initialized yet (this will automatically trigger the use of the change-of-frame formula for tensors):

```sage
sage: a = E.section_module(domain=U).automorphism()
sage: a[:][:] = [[1+x^2,0],[0,1+y^2]]
sage: e = fN.new_frame(a, 'e')
sage: [e[i].display() for i in S2.irange()]
[e_1 = (x^2 + 1) (phi_U^*e_1), e_2 = (y^2 + 1) (phi_U^*e_2)]
sage: s.display(e)
s = x/(x^2 + 1) e_1 + y/(y^2 + 1) e_2
```

A shortcut of `display()` is `disp()`:

```sage
sage: s.disp(fS)
s = v/(u^2 + v^2) (phi_V^*e_1) + u/(u^2 + v^2) (phi_V^*e_2)
```

### `display_comp(frame=None, chart=None, only_nonzero=True)`

Display the section 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) local frame with respect to which the section components are defined; if None, then the default frame on the section module is used
- `chart` - (default: None) chart specifying the coordinate expression of the components; if None, the default chart of the section domain is used
- `only_nonzero` - (default: True) boolean; if True, only nonzero components are displayed

**EXAMPLES:**

Display of the components of a section defined on two open subsets:

```sage
sage: M = Manifold(2, 'M', structure='top')
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: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e', domain=U)
sage: f = E.local_frame('f', domain=V)
sage: s = E.section(name='s')
sage: s[e,0] = - x + y^3
sage: s[e,1] = 2+x
sage: s[f,1] = - u*v
sage: s.display_comp(e)
s^0 = y^3 - x
s^1 = x + 2
sage: s.display_comp(f)
s^1 = -u*v
```

See documentation of `sage.manifolds.section.TrivialSection.display_comp()` for more options.
domain()

Return the manifold on which self is defined.

OUTPUT:

- instance of class TopologicalManifold

EXAMPLES:

```
sage: M = Manifold(3, 'M', structure='top')
sage: U = M.open_subset('U')
sage: E = M.vector_bundle(2, 'E')
sage: C0_U = E.section_module(domain=U, force_free=True)
sage: z = C0_U.zero()
sage: z.domain()
Open subset U of the 3-dimensional topological manifold M
```

restrict (subdomain)

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 section domain $S$

OUTPUT:

- Section representing the restriction

EXAMPLES:

Restrictions of a section on a rank 2 vector bundle over the 2-sphere:

```
sage: S2 = Manifold(2, 'S^2', structure='top', start_index=1)
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: stereoN.<x,y> = U.chart() # stereographic coordinates from the North→pole
sage: stereoS.<u,v> = V.chart() # stereographic coordinates from the South→pole
sage: xy_to_uv = 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: W = U.intersection(V)
sage: uv_to_xy = xy_to_uv.inverse()
sage: E = S2.vector_bundle(2, 'E') # define vector bundle
sage: phi_U = E.trivialization('phi_U', domain=U) # define trivializations
sage: phi_V = E.trivialization('phi_V', domain=V)
sage: transf = phi_U.transition_map(phi_V, [[0,x],[y,0]])
sage: fN = phi_U.frame(); fS = phi_V.frame() # get induced frames
sage: fN_W = fN.restrict(W); fS_W = fS.restrict(W) # restrict them
sage: stereoN_W = stereoN.restrict(W) # restrict charts, too
sage: stereoS_W = stereoS.restrict(W)
sage: s = E.section({fN: [1, 0]}, name='s')
sage: s.display(fN)
s = (phi_U^*e_1)
sage: sU = s.restrict(U) ; sU
```

(continues on next page)
At this stage, defining the restriction of \( s \) to the open subset \( V \) fully specifies \( s \):

```python
sage: s.restrict(V)[1] = sW[fS_W, 1, stereoS_W].expr() # note that fS is the...
```

The restriction of the section to its own domain is of course itself:

```python
sage: s.restrict(S2) is s
True
sage: sU.restrict(U) is sU
True
```

**set_comp**(basis=None)

Return the components of \( \text{self} \) in a given local frame for assignment.

The components with respect to other frames having the same domain as the provided local frame are deleted, in order to avoid any inconsistency. To keep them, use the method `add_comp()` instead.

**INPUT:**

- **basis** – (default: None) local frame in which the components are defined; if none is provided, the components are assumed to refer to the section domain’s default frame

**OUTPUT:**

- components in the given frame, as a `Components`; if such components did not exist previously, they are created

**EXAMPLES:**

```python
sage: S2 = Manifold(2, 'S^2', structure='top', start_index=1)
sage: U = S2.open_subset('U'); V = S2.open_subset('V') # complement of the...
```

(continues on next page)
sage: S2.declare_union(U,V)
sage: stereoN.<x,y> = U.chart()  # stereographic coordinates from the North
  pole
sage: stereoS.<u,v> = V.chart()  # stereographic coordinates from the South
  pole
sage: xy_to_uv = 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: W = U.intersection(V)
sage: uv_to_xy = xy_to_uv.inverse()
sage: E = S2.vector_bundle(2, E)  # define vector bundle
sage: phi_U = E.trivialization(phi_U, domain=U)  # define trivializations
sage: phi_V = E.trivialization('phi_V', domain=V)
sage: transf = phi_U.transition_map(phi_V, [[0,x],[y,0]])
sage: fN = phi_U.frame(); fS = phi_V.frame()  # get induced frames
sage: s = E.section(name=s)
sage: s.set_comp(fS)
1-index components w.r.t. Trivialization frame (E|_V, ((phi_V^*e_1),(phi_V^*e_2)))
sage: s.set_comp(fS)[1] = u+v
sage: s.display(fS)
s = (u + v) (phi_V^*e_1)

Setting the components in a new frame (e):

sage: e = E.local_frame('e', domain=V)
sage: s.set_comp(e)
1-index components w.r.t. Local frame (E|_V, (e_1,e_2))
sage: s.set_comp(e)[1] = u*v
sage: s.display(e)
s = u*v e_1

Since the frames e and fS are defined on the same domain, the components w.r.t. fS have been erased:

sage: s.display(phi_V.frame())
Traceback (most recent call last):
... ValueError: no basis could be found for computing the components in the Trivialization frame (E|_V, ((phi_V^*e_1),(phi_V^*e_2)))

\texttt{set\_immutable()}

Set \texttt{self} and all restrictions of \texttt{self} immutable.

EXAMPLES:

sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: U = M.open_subset('U', coord_def={X: x^2+y^2<1})
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e')
sage: s = E.section([1+y,x], name='s')
sage: sU = s.restrict(U)
sage: s.set_immutable()
sage: s.is_immutable()
True
sage: sU.is Immutable()
True

**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 section
- latex\_name – string (default: None); LaTeX symbol to denote the section; if None while name is provided, the LaTeX symbol is set to name

**EXAMPLES:**

```
sage: M = Manifold(3, 'M', structure='top')
sage: E = M.vector\_bundle(2, 'E')
sage: s = E.section(); s
Section on the 3-dimensional topological manifold \(M\) with values in
the real vector bundle \(E\) of rank 2
sage: s.set\_name(name='s')
sage: s
Section s on the 3-dimensional topological manifold \(M\) with values in
the real vector bundle \(E\) of rank 2
sage: latex(s)
s
sage: s.set\_name(latex\_name=r'\sigma')
sage: latex(s)
\sigma
sage: s.set\_name(name='a')
sage: s
Section a on the 3-dimensional topological manifold \(M\) with values in
the real vector bundle \(E\) of rank 2
sage: latex(s)
a
```

**set\_restriction**(rst)

Define a restriction of self to some subdomain.

**INPUT:**

- rst – Section defined on a subdomain of the domain of self

**EXAMPLES:**

```
sage: S2 = Manifold(2, 'S^2', structure='top')
sage: U = S2.open\_subset('U') ; V = S2.open\_subset('V') \# complement of the
\rightarrow North and South pole, respectively
sage: S2.declare\_union(U,V)
sage: stereoN.<x,y> = U.chart() \# stereographic coordinates from the North...
\rightarrow pole
sage: stereoS.<u,v> = V.chart() \# stereographic coordinates from the South...
\rightarrow pole
sage: xy\_to\_uv = 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)
```

(continues on next page)
class sage.manifolds.section.TrivialSection(section_module, name=None, latex_name=None)

Bases: FiniteRankFreeModuleElement, Section

Section in a trivial vector bundle.

An instance of this class is a section in a vector bundle $E \to M$ of class $C^k$, where $E|_U$ is manifestly trivial. More precisely, a (local) section on a subset $U \subseteq M$ is a map of class $C^k$

$$s : U \to E$$

such that

$$\forall p \in U, s(p) \in E_p$$

where $E_p$ denotes the vector bundle fiber of $E$ over the point $p \in U$. $E$ being trivial means $E$ being homeomorphic to $E \times F$, for $F$ is the typical fiber of $E$, namely the underlying topological vector space. By this means, $s$ can be seen as a map of class $C^k(U; E)$

$$s : U \to F,$$

so that the set of all sections $C^k(U; E)$ becomes a free module over the algebra of scalar fields on $U$.

Note: If $E|_U$ is not manifestly trivial, the class Section should be used instead.

This is a Sage element class, the corresponding parent class being SectionFreeModule.

INPUT:

- section_module -- free module $C^k(U; E)$ of sections on $E$ over $U$ (cf. SectionFreeModule)
- name -- (default: None) name given to the section
- latex_name -- (default: None) LaTeX symbol to denote the section; if none is provided, the LaTeX symbol is set to name

EXAMPLES:

A section on a trivial rank 3 vector bundle over the 3-sphere:

sage: M = Manifold(3, 'S^3', structure='top')

sage: U = M.open_subset('U') ; V = M.open_subset('V') # complement of the North_ and South pole, respectively

sage: M.declare_union(U,V)

sage: E = S2.vector_bundle(3, 'E')

sage: phi_U = E.trivialization(phi_U, domain=U)

sage: phi_V = E.trivialization(phi_V, domain=V)

sage: s = E.section(name=s)

sage: sU = E.section(domain=U, name=s)

sage: sU[:] = x+y, x

sage: s.set_restriction(sU)

sage: s.display(phi_U.frame())

s = (x + y) (phi_U^*e_1) + x (phi_U^*e_2)

sage: s.restrict(U) == sU

True
Manifolds, Release 10.3

(continued from previous page)

```sage
sage: stereoN.<x,y,z> = U.chart()  # stereographic coordinates from the North pole
sage: stereoS.<u,v,t> = V.chart()  # stereographic coordinates from the South pole
sage: xyz_to_uvt = stereoN.transition_map(stereoS,
    ....: (x/(x^2+y^2+z^2), y/(x^2+y^2+z^2), z/(x^2+y^2+z^2)),
    ....: intersection_name='W',
    ....: restrictions1= x^2+y^2+z^2!=0,
    ....: restrictions2= u^2+v^2+t^2!=0)
```

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

```sage
sage: uvt_to_xyz = xyz_to_uvt.inverse()
```

```sage
sage: E = M.vector_bundle(3, E)
```

```sage
sage: e = E.local_frame(e)  # Trivializes E
```

```sage
sage: s = E.section(name=s); s
Section s on the 3-dimensional topological manifold S^3 with values in
the real vector bundle E of rank 3
```

```sage
sage: s[e, :) = z^2, x - y, 1 - x
sage: s.display()
s = z^2 e_0 + (x - y) e_1 + (-x + 1) e_2
```

Since $E$ is trivial, $s$ is now element of a free section module:

```sage
sage: s.parent()  # Free module $C^0(S^3;E)$ of sections on the 3-dimensional topological manifold $S^3$ with values in the real vector bundle $E$ of rank 3
Free module $C^0(S^3;E)$ of sections on the 3-dimensional topological manifold $S^3$ with values in the real vector bundle $E$ of rank 3
```

```sage
sage: isinstance(s.parent(), FiniteRankFreeModule)
True
```

**add_comp** *(basis=None)*

Return the components of the section in a given local frame for assignment.

The components with respect to other frames on the same domain are kept. To delete them, use the method *set_comp()* instead.

**INPUT:**
- *basis* – (default: None) local frame in which the components are defined; if none is provided, the components are assumed to refer to the section module’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
sage: M = Manifold(2, 'M', structure='top')
sage: X.<x,y> = M.chart()
sage: E = M.vectorBundle(2, 'E')
sage: e = E.localFrame('e')  # makes E trivial
sage: s = E.section(name='s')
sage: s.add_comp(e)
sage: s.add_comp(e)[0] = 2
sage: s.display(e)
s = 2 e_0
```

Adding components with respect to a new frame ($f$):

```sage
```
```python
sage: f = E.local_frame('f')
sage: s.add_comp(f)
1-index components w.r.t. Local frame (E|_M, (f_0,f_1))
sage: s.add_comp(f)[0] = x
sage: s.display(f)
s = x f_0
```

The components with respect to the frame e are kept:

```python
sage: s.display(e)
s = 2 e_0
```

Adding components in a frame defined on a subdomain:

```python
sage: U = M.open_subset('U', coord_def={X: x>0})
sage: g = E.local_frame('g', domain=U)
sage: s.add_comp(g)
1-index components w.r.t. Local frame (E|_U, (g_0,g_1))
sage: s.add_comp(g)[0] = 1+y
sage: s.display(g)
s = (y + 1) g_0
```

The components previously defined are kept:

```python
sage: s.display(e)
s = 2 e_0
sage: s.display(f)
s = x f_0
```

**at (point)**

Value of `self` at a point of its domain.

If the current section is

\[ s : U \rightarrow E, \]

then for any point \( p \in U \), \( s(p) \) is a vector in the fiber \( E_p \) of \( E \) at the point \( p \in U \).

**INPUT:**

- `point` - `ManifoldPoint` point \( p \) in the domain of the section \( U \)

**OUTPUT:**

- `FreeModuleTensor` representing the vector \( s(p) \) in the vector space \( E_p \)

**EXAMPLES:**

Vector in a tangent space of a 2-dimensional manifold:

```python
sage: M = Manifold(2, 'M', structure='top')
sage: X.<x,y> = M.chart()
sage: p = M.point((-2,3), name='p')
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e')  # makes E trivial
sage: s = E.section(y, x^2, name='s')
sage: s.display()
s = y e_0 + x^2 e_1
sage: sp = s.at(p) ; sp
```

(continues on next page)
Vector $s$ in the fiber of $E$ at Point $p$ on the 2-dimensional topological manifold $M$

```sage```
sp.parent()
```
Fiber of $E$ at Point $p$ on the 2-dimensional topological manifold $M$

```sage```
sp.display()
```
s = 3 e_0 + 4 e_1
```

`comp` *(basis=None, from_basis=None)*

Return the components in a given local frame.

If the components are not known already, they are computed by the tensor change-of-basis formula from components in another local frame.

**INPUT:**

- `basis` — (default: None) local frame in which the components are required; if none is provided, the components are assumed to refer to the section module’s default frame
- `from_basis` — (default: None) local frame from which the required components are computed, via the tensor change-of-basis formula, if they are not known already in the basis `basis`

**OUTPUT:**

- components in the local frame `basis`, as an instance of the class `Components`

**EXAMPLES:**

```sage```
M = Manifold(2, 'M', structure='top', start_index=1)
sage: X.<x,y> = M.chart()
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e') # makes E trivial
sage: s = E.section(name='s')
sage: s[1] = x*y
sage: s.comp(e)
1-index components w.r.t. Local frame (E|_M, (e_1,e_2))
sage: s.comp() # the default frame is e
1-index components w.r.t. Local frame (E|_M, (e_1,e_2))
sage: s.comp()[[x*y, 0]
sage: f = E.local_frame('f')
sage: s[f, 1] = x-3
sage: s.comp(f)
1-index components w.r.t. Local frame (E|_M, (f_1,f_2))
sage: s.comp(f)[[x - 3, 0]
```

`display_comp` *(frame=None, chart=None, only_nonzero=False)*

Display the section 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) local frame with respect to which the section components are defined; if None, then the default basis of the section module on which the section is defined is used
- `chart` — (default: None) chart specifying the coordinate expression of the components; if None, the default chart of the section module domain is used
- `only_nonzero` — (default: False) boolean; if True, only nonzero components are displayed
EXAMPLES:

Display of the components of a section on a rank 4 vector bundle over a 2-dimensional manifold:

```sage
M = Manifold(2, 'M', structure='top')
x.<x,y> = M.chart()
E = M.vector_bundle(3, 'E')
e = E.local_frame('e') # makes E trivial
s = E.section(name='s')
s[0], s[2] = x+y, x*y
s.display_comp()
s^0 = x + y
s^1 = 0
s^2 = x*y
```

By default, the vanishing components are displayed, too; to see only non-vanishing components, the argument `only_nonzero` must be set to `True`:

```sage
s.display_comp(only_nonzero=True)
s^0 = x + y
s^2 = x*y
```

Display in a frame different from the default one:

```sage
a = E.section_module().automorphism()
a[0] = [1+y^2, 0, 0, 0, 2+x^2, 0, 0, 0, 1]
f = e.new_frame(a, 'f')
s.display_comp(frame=f)
s^0 = (x + y)/(y^2 + 1)
s^1 = 0
s^2 = x*y
```

Display with respect to a chart different from the default one:

```sage
Y.<u,v> = M.chart()
X_to_Y = X.transition_map(Y, [x+y, x-y])
Y_to_X = X_to_Y.inverse()
s.display_comp(chart=Y)
s^0 = u
s^1 = 0
s^2 = 1/4*u^2 - 1/4*v^2
```

Display of the components with respect to a specific frame, expressed in terms of a specific chart:

```sage
s.display_comp(frame=f, chart=Y)
s^0 = 4*u/(u^2 - 2*u*v + v^2 + 4)
s^1 = 0
s^2 = 1/4*u^2 - 1/4*v^2
```

`restrict` `subdomain`

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 section module domain `S`

OUTPUT:
instance of *TrivialSection* representing the restriction

**EXAMPLES:**

Restriction of a section defined over $\mathbb{R}^2$ to a disk:

```python
sage: M = Manifold(2, 'R^2')
sage: c_cart.<x,y> = M.chart() # Cartesian coordinates on $\mathbb{R}^2$
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e') # makes E trivial
sage: s = E.section(x+y, -1+x^2, name='s')
sage: D = M.open_subset('D') # the unit open disc
sage: e_D = e.restrict(D)
sage: c_cart_D = c_cart.restrict(D, x^2+y^2<1)
sage: s_D = s.restrict(D) ; s_D
Section s on the Open subset D of the 2-dimensional differentiable manifold $\mathbb{R}^2$ with values in the real vector bundle E of rank 2
sage: s_D.display(e_D)
s = (x + y) e_0 + (x^2 - 1) e_1
```

The symbolic expressions of the components with respect to Cartesian coordinates are equal:

```python
sage: bool( s_D[1].expr() == s[1].expr() )
True
```

but neither the chart functions representing the components (they are defined on different charts):

```python
sage: s_D[1] == s[1]
False
```

nor the scalar fields representing the components (they are defined on different open subsets):

```python
sage: s_D[[1]] == s[[1]]
False
```

The restriction of the section to its own domain is of course itself:

```python
sage: s.restrict(M) is s
True
```

**set_comp** (*basis=None*)

Return the components of the section in a given local 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*) local frame in which the components are defined; if none is provided, the components are assumed to refer to the section module’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:**

```python
sage: M = Manifold(2, 'M', structure='top')
sage: X.<x,y> = M.chart()
```
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e') # makes E trivial
sage: s = E.section(name='s')
sage: s.set_comp(e)
1-index components w.r.t. Local frame (E|_M, (e_0,e_1))
sage: s.set_comp(e)[0] = 2
sage: s.display(e)
s = 2 e_0

Setting components in a new frame (f):

sage: f = E.local_frame('f')
sage: s.set_comp(f)
1-index components w.r.t. Local frame (E|_M, (f_0,f_1))
sage: s.set_comp(f)[0] = x
sage: s.display(f)
s = x f_0

The components with respect to the frame e have been erased:

sage: s.display(e)
Traceback (most recent call last):
  ...  
ValueError: no basis could be found for computing the components
in the Local frame (E|_M, (e_0,e_1))

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: g = E.local_frame('g', domain=U)
sage: s.set_comp(g)
1-index components w.r.t. Local frame (E|_U, (g_0,g_1))
sage: s.set_comp(g)[0] = 1+y
sage: s.display(g)
s = (y + 1) g_0
sage: s.display(f)
Traceback (most recent call last):
  ...  
ValueError: no basis could be found for computing the components
in the Local frame (E|_M, (f_0,f_1))

1.10 Families of Manifold Objects

The class ManifoldObjectFiniteFamily is a subclass of FiniteFamily that provides an associative container of manifold objects, indexed by their _name attributes.

ManifoldObjectFiniteFamily instances are totally ordered according to their lexicographically ordered element names.

The subclass ManifoldSubsetFiniteFamily customizes the print representation further.

AUTHORS:

• Matthias Koeppe (2021): initial version
**Manifolds, Release 10.3**

```python
class sage.manifolds.family.ManifoldObjectFiniteFamily(objects=(), keys=None)
Bases: FiniteFamily

Finite family of manifold objects, indexed by their names.

The class `ManifoldObjectFiniteFamily` inherits from `FiniteFamily`. Therefore it is an associative container.

It provides specialized `__repr__` and `latex` methods.

`ManifoldObjectFiniteFamily` instances are totally ordered according to their lexicographically ordered element names.

**EXAMPLES:**

```python
sage: from sage.manifolds.family import ManifoldObjectFiniteFamily
sage: M = Manifold(2, 'M', structure='topological')
sage: A = M.subset('A')
sage: B = M.subset('B')
sage: C = B.subset('C')
sage: F = ManifoldObjectFiniteFamily([A, B, C]); F
Set {A, B, C} of objects of the 2-dimensional topological manifold M
sage: latex(F)\{A, B, C\}
sage: F['B']Subset B of the 2-dimensional topological manifold M
```

All objects must have the same base manifold:

```python
sage: N = Manifold(2, 'N', structure='topological')
sage: ManifoldObjectFiniteFamily([M, N])
Traceback (most recent call last):
...TypeError: all objects must have the same manifold
```

```python
class sage.manifolds.family.ManifoldSubsetFiniteFamily(objects=(), keys=None)
Bases: ManifoldObjectFiniteFamily

Finite family of subsetsof a topological manifold, indexed by their names.

The class `ManifoldSubsetFiniteFamily` inherits from `ManifoldObjectFiniteFamily`. It provides an associative container with specialized `__repr__` and `latex` methods.

`ManifoldSubsetFiniteFamily` instances are totally ordered according to their lexicographically ordered element (subset) names.

**EXAMPLES:**

```python
sage: from sage.manifolds.family import ManifoldSubsetFiniteFamily
sage: M = Manifold(2, 'M', structure='topological')
sage: A = M.subset('A')
sage: B = M.subset('B')
sage: C = B.subset('C')
sage: ManifoldSubsetFiniteFamily([A, B, C])
Set {A, B, C} of subsets of the 2-dimensional topological manifold M
sage: latex(_)\{A, B, C\}
```

All subsets must have the same base manifold:

```python
sage: N = Manifold(2, 'N', structure='topological')
sage: ManifoldSubsetFiniteFamily([M, N])
Traceback (most recent call last):
...TypeError: all objects must have the same manifold
```

1.10. Families of Manifold Objects
sage: N = Manifold(2, 'N', structure='topological')
sage: ManifoldSubsetFiniteFamily([M, N])
Traceback (most recent call last):
  ... TypeError: all open subsets must have the same manifold

classmethod from_subsets_or_families(*subsets_or_families)

Construct a ManifoldSubsetFiniteFamily from given subsets or iterables of subsets.

EXAMPLES:

sage: from sage.manifolds.family import ManifoldSubsetFiniteFamily
sage: M = Manifold(2, 'M', structure='topological')
sage: A = M.subset('A')
sage: Bs = (M.subset('B' + str(i)) for i in range(5))
sage: Cs = ManifoldSubsetFiniteFamily([M.subset('C0'), M.subset('C1')])
sage: ManifoldSubsetFiniteFamily.from_subsets_or_families(A, Bs, Cs)
Set {A, B0, B1, B2, B3, B4, C0, C1} of subsets of the 2-dimensional topological manifold M

1.11 Topological Closures of Manifold Subsets

ManifoldSubsetClosure implements the topological closure of a manifold subset in the topology of the manifold.

class sage.manifolds.subsets.closure.ManifoldSubsetClosure(subset, name=None, latex_name=None)

Bases: ManifoldSubset

Topological closure of a manifold subset in the topology of the manifold.

INPUT:

- subset – a ManifoldSubset
- name – (default: computed from the name of the subset) string; name (symbol) given to the closure
- latex_name – (default: None) string; LaTeX symbol to denote the subset; if none is provided, it is set to name

EXAMPLES:

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', coord_def={c_cart: x^2+y^2<1}); D
Open subset D of the 2-dimensional topological manifold R^2
sage: cl_D = D.closure()
sage: cl_D
Topological closure cl_D of the Open subset D of the 2-dimensional topological manifold R^2
sage: latex(cl_D)
\mathop{\mathrm{cl}}(D)
sage: type(cl_D)
<class 'sage.manifolds.subsets.closure.ManifoldSubsetClosure_with_category'>
sage: cl_D.category()
Category of subobjects of sets
The closure of the subset $D$ is a subset of every closed superset of $D$:

```python
sage: S = D.superset('S')
sage: S.declare_closed()
sage: c1_D.is_subset(S)
True
```

**is_closed()**

Return if self is a closed set.

This implementation of the method always returns True.

**EXAMPLES:**

```python
sage: from sage.manifolds.subsets.closure import ManifoldSubsetClosure
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', coord_def={c_cart: x^2+y^2<1}); D
Open subset D of the 2-dimensional topological manifold R^2
sage: cl_D = D.closure(); cl_D  # indirect doctest
Topological closure cl_D of the Open subset D of the 2-dimensional...
→topological manifold R^2
sage: cl_D.is_closed()
True
```

## 1.12 Manifold Subsets Defined as Pullbacks of Subsets under Continuous Maps

**class** `sage.manifolds.subsets.pullback.ManifoldSubsetPullback(map, codomain_subset, inverse, name, latex_name)`

**Bases:** `ManifoldSubset`

Manifold subset defined as a pullback of a subset under a continuous map.

**INPUT:**

- `map` - an instance of `ContinuousMap`, `ScalarField`, or `Chart`
- `codomain_subset` - an instance of `ManifoldSubset`, `RealSet`, or `ConvexSet_base`

**EXAMPLES:**

```python
sage: from sage.manifolds.subsets.pullback import ManifoldSubsetPullback
sage: M = Manifold(2, 'R^2', structure='topological')
sage: c_cart.<x,y> = M.chart() # Cartesian coordinates on R^2
sage: r_squared = M.scalar_field(x^2+y^2)
sage: r_squared.set_immutable()
sage: cl_I = RealSet([1, 4]); cl_I
[1, 4]
sage: cl_O = ManifoldSubsetPullback(r_squared, cl_I); cl_O
Subset f_inv_[1, 4] of the 2-dimensional topological manifold R^2
sage: M.point((0, 0)) in cl_O
True
```

(continues on next page)
Pulling back an open real interval gives an open subset:

```
sage: I = RealSet((1, 4)); I
(1, 4)
sage: O = ManifoldSubsetPullback(r_squared, I); O
Open subset f_inv_(1, 4) of the 2-dimensional topological manifold R^2
sage: M.point((1, 0)) in O
False
sage: M.point((1, 1)) in O
True
```

Pulling back a polytope under a chart:

```
sage: # needs sage.geometry.polyhedron
sage: P = Polyhedron(3, vertices=[[0, 0], [1, 2], [2, 1]]); P
A 2-dimensional polyhedron in ZZ^2 defined as the convex hull of 3 vertices
sage: S = ManifoldSubsetPullback(c_cart, P); S
Subset x_y_inv_P of the 2-dimensional topological manifold R^2
sage: M((1, 2)) in S
True
sage: M((2, 0)) in S
False
```

Pulling back the interior of a polytope under a chart:

```
sage: # needs sage.geometry.polyhedron
sage: int_P = P.interior(); int_P
Relative interior of a 2-dimensional polyhedron in ZZ^2 defined as the convex hull of 3 vertices
sage: int_S = ManifoldSubsetPullback(c_cart, int_P, name='int_S'); int_S
Open subset int_S of the 2-dimensional topological manifold R^2
sage: M((0, 0)) in int_S
False
sage: M((1, 1)) in int_S
True
```

Using the embedding map of a submanifold:

```
sage: M = Manifold(3, 'M', structure="topological")
sage: N = Manifold(2, 'N', ambient=M, structure="topological"); N
2-dimensional topological submanifold N
immersed in the 3-dimensional topological manifold M
sage: CM.<x,y,z> = M.chart()
sage: CN.<u,v> = N.chart()
sage: t = var('t')
sage: phi = N.continuous_map(M, {(CN,CM): [u,v,t+u^2+v^2]})
sage: phi_inv = M.continuous_map(N, {(CM,CN): [x,y]})
sage: phi_inv_t = M.scalar_field({CM: z-x^2-y^2})
sage: N.set_immersion(phi, inverse=phi_inv, var=t, ...
..: t_inverse={t: phi_inv_t})
sage: N.declare_embedding()
```

Chapter 1. Topological Manifolds
sage: S = M.open_subset('S', coord_def={CM: z<1})
sage: phi_without_t = N.continuous_map(M, {(CN, CM): [expr .subs(t=0) 
.....: for expr in phi.expr()]})
sage: phi_without_t
Continuous map
  from the 2-dimensional topological submanifold N
  embedded in the 3-dimensional topological manifold M
  to the 3-dimensional topological manifold M
sage: phi_without_t.expr()
(u, v, u^2 + v^2)
sage: D = ManifoldSubsetPullback(phi_without_t, S); D
Subset f_inv_S of the 2-dimensional topological submanifold N
  embedded in the 3-dimensional topological manifold M
sage: N.point((2,0)) in D
False

```
closure(name=None, latex_name=None)
```

Return the topological closure of self in the manifold.

Because self is a pullback of some subset under a continuous map, the closure of self is the pullback of the closure.

EXAMPLES:

```
sage: from sage.manifolds.subsets.pullback import ManifoldSubsetPullback
sage: M = Manifold(2, R^2, structure='topological')
sage: c_cart.<x,y> = M.chart() # Cartesian coordinates on R^2
sage: r_squared = M.scalar_field(x^2+y^2)
sage: r_squared.set_immutable()
sage: I = RealSet.open_closed(1, 2); I
(1, 2]
sage: O = ManifoldSubsetPullback(r_squared, I); O
Subset f_inv_(1, 2] of the 2-dimensional topological manifold R^2
sage: latex(O)
f^{-1}((1, 2])
sage: cl_O = O.closure(); cl_O
Subset f_inv_[1, 2] of the 2-dimensional topological manifold R^2
sage: cl_O.is_closed()
True
```

```
is_closed()
```

Return if self is (known to be) a closed subset of the manifold.

EXAMPLES:

```
sage: from sage.manifolds.subsets.pullback import ManifoldSubsetPullback
sage: M = Manifold(2, R^2, structure='topological')
sage: c_cart.<x,y> = M.chart() # Cartesian coordinates on R^2
sage: r_squared = M.scalar_field(x^2+y^2)
sage: r_squared.set_immutable()
sage: cl_I = RealSet([1, 2]); cl_I
[1, 2]
sage: cl_O = ManifoldSubsetPullback(r_squared, cl_I); cl_O
Subset f_inv_[1, 2] of the 2-dimensional topological manifold R^2
```

The pullback of a closed real interval under a scalar field is closed:

```
sage: r_squared = M.scalar_field(x^2+y^2)
sage: r_squared.set_immutable()
sage: cl_I = RealSet([1, 2]); cl_I
[1, 2]
sage: cl_O = ManifoldSubsetPullback(r_squared, cl_I); cl_O
Subset f_inv_[1, 2] of the 2-dimensional topological manifold R^2
```

1.12. Manifold Subsets Defined as Pullbacks of Subsets under Continuous Maps
The pullback of a (closed convex) polyhedron under a chart is closed:

```
sage: # needs sage.geometry.polyhedron
sage: P = Polyhedron( vertices=[[0, 0], [1, 2], [3, 4]]); P
A 2-dimensional polyhedron in ZZ^2 defined as the convex hull of 3 vertices
sage: McP = ManifoldSubsetPullback( c_cart, P, name='McP'); McP
Subset McP of the 2-dimensional topological manifold R^2
sage: McP.is_closed()
True
```

The pullback of real vector subspaces under a chart is closed:

```
sage: V = span([[1, 2]], RR); V
Vector space of degree 2 and dimension 1 over Real Field with 53 bits of...
Basis matrix:
[1.00000000000000 2.00000000000000]
sage: McV = ManifoldSubsetPullback( c_cart, V, name='McV'); McV
Subset McV of the 2-dimensional topological manifold R^2
sage: McV.is_closed()
True
```

The pullback of point lattices under a chart is closed:

```
sage: W = span([[1, 0], [3, 5]], ZZ); W
Free module of degree 2 and rank 2 over Integer Ring
Echelon basis matrix:
[1 0]
[0 5]
sage: McW = ManifoldSubsetPullback( c_cart, W, name='McW'); McW
Subset McW of the 2-dimensional topological manifold R^2
sage: McW.is_closed()
True
```

The pullback of finite sets is closed:

```
sage: F = Family([ vector(QQ, [1, 2], immutable=True), vector(QQ, [2, 3],...
--immutable=True)])
sage: McF = ManifoldSubsetPullback( c_cart, F, name='McF'); McF
Subset McF of the 2-dimensional topological manifold R^2
sage: McF.is_closed()
True
```

Return if self is (known to be) an open set.

This version of the method always returns False.

Because the map is continuous, the pullback is open if the codomain_subset is open.

However, the design of ManifoldSubset requires that open subsets are instances of the subclass sage.manifolds.manifold.TopologicalManifold. The constructor of ManifoldSubsetPullback delegates to a subclass of sage.manifolds.manifold.TopologicalManifold for some open subsets.
EXAMPLES:

```python
# Sage
sage: from sage.manifolds.subsets.pullback import ManifoldSubsetPullback
sage: M = Manifold(2, 'R^2', structure='topological')
sage: c_cart.<x,y> = M.chart() # Cartesian coordinates on R^2

sage: # needs sage.geometry.polyhedron
sage: P = Polyhedron(vertices=[[0, 0], [1, 2], [3, 4]]); P
A 2-dimensional polyhedron in ZZ^2 defined as the convex hull of 3 vertices

sage: P.is_open()
False

sage: McP = ManifoldSubsetPullback(c_cart, P, name='McP'); McP
Subset McP of the 2-dimensional topological manifold R^2

sage: McP.is_open()
False

```
some_elements()
Generate some elements of self.

EXAMPLES:

```python
# Sage
sage: from sage.manifolds.subsets.pullback import ManifoldSubsetPullback
sage: M = Manifold(3, 'R^3', structure='topological')
sage: c_cart.<x,y,z> = M.chart() # Cartesian coordinates on R^3

sage: Cube = polytopes.cube(); Cube
A 3-dimensional polyhedron in ZZ^3 defined as the convex hull of 8 vertices

sage: McCube = ManifoldSubsetPullback(c_cart, Cube, name='McCube'); McCube
Subset McCube of the 3-dimensional topological manifold R^3

sage: L = list(McCube.some_elements()); L
[Point on the 3-dimensional topological manifold R^3,
 Point on the 3-dimensional topological manifold R^3,
 Point on the 3-dimensional topological manifold R^3,
 Point on the 3-dimensional topological manifold R^3,
 Point on the 3-dimensional topological manifold R^3,
 Point on the 3-dimensional topological manifold R^3]

sage: list(p.coordinates(c_cart) for p in L)
[(0, 0, 0),
 (1, -1, -1),
 (1, 0, -1),
 (1, 1/2, 0),
 (1, -1/4, 1/2),
 (0, -5/8, 3/4)]

sage: # needs sage.geometry.polyhedron
sage: Empty = Polyhedron(ambient_dim=3)

sage: McEmpty = ManifoldSubsetPullback(c_cart, Empty, name='McEmpty')

sage: McEmpty
Subset McEmpty of the 3-dimensional topological manifold R^3

sage: list(McEmpty.some_elements())
[]
```

1.12. Manifold Subsets Defined as Pullbacks of Subsets under Continuous Maps
2.1 Differentiable Manifolds

Given a non-discrete topological field $K$ (in most applications, $K = \mathbb{R}$ or $K = \mathbb{C}$; see however [Ser1992] for $K = \mathbb{Q}_p$ and [Ber2008] for other fields), a differentiable manifold over $K$ is a topological manifold $M$ over $K$ equipped with an atlas whose transitions 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 $K = \mathbb{R}$ is assumed;
- if $K = \mathbb{C}$, any $C^k$-manifold with $k \geq 1$ is actually a $C^\infty$-manifold (even an analytic manifold);
- if $K = \mathbb{R}$, 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 $\mathbb{R}$

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, $\mathbb{R}$ 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 $U$:
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 $<x, y>$ 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)
<class '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 $\bar{W}$ 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 \texttt{inverse}():

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

\begin{verbatim}
sage: M.subset_family()
Set {S^2, U, V, W} of open subsets of the 2-dimensional differentiable manifold S^2
\end{verbatim}

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

\begin{verbatim}
sage: N in W or S in W
False
\end{verbatim}

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

\begin{verbatim}
sage: N in V, N in U
(True, False)
sage: S in U, S in V
(True, False)
\end{verbatim}

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

\begin{verbatim}
sage: M.atlas()
[Chart (U, (x, y)), Chart (V, (u, v)), Chart (W, (x, y)), Chart (W, (u, v))]
\end{verbatim}

Let us consider the point of coordinates $(1, 2)$ in the chart $\text{stereoN}$:

\begin{verbatim}
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
\end{verbatim}

The coordinates of $p$ in the chart $\text{stereoS}$ are computed by letting the chart act on the point:

\begin{verbatim}
sage: stereoS(p)
(1/5, 2/5)
\end{verbatim}

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

\begin{verbatim}
sage: stereoN(p)
(1, 2)
\end{verbatim}

Similarly:

\begin{verbatim}
sage: stereoS(N)
(0, 0)
sage: stereoN(S)
(0, 0)
\end{verbatim}

A differentiable scalar field on the sphere:
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, (∂/∂x, ∂/∂y))
 sage: latex(M.default_frame())
 \left(U, \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)\right)
```

A vector field on the sphere:

```
 sage: w = M.vector_field(name='w')
 sage: w[stereoN.frame(), i] = [x, y]
 sage: w.add_comp_by_continuation(stereoS.frame(), W, stereoS)
 sage: w.display() # display in the default frame (stereoN.frame())
 w = x ∂/∂x + y ∂/∂y
 sage: w.display(stereoS.frame())
 w = -u ∂/∂u - v ∂/∂v
```

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:

```python
sage: w.at(p)
Tangent vector \( w \) at Point \( p \) on the 2-dimensional differentiable manifold \( S^2 \)
sage: w.at(p).display()
w = \partial/\partial x + 2 \partial/\partial y
sage: w.at(p).parent()
Tangent space at Point \( p \) on the 2-dimensional differentiable manifold \( S^2 \)
```

A 1-form on the sphere:

```python
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 \( \Omega^1(S^2) \) of 1-forms 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 \( p \):

```python
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 \( \mathbb{C} \):

```python
sage: M = Manifold(1, 'C*', field='complex'); M
1-dimensional complex manifold \( \mathbb{C}^* \)
```

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 \( \mathbb{C} \):

```python
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 \( z \):

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

The origin of the complex plane is the point of coordinate \( z = 0 \):

```python
sage: O = U.point((0,), chart=Z, name='O'); O
Point \( O \) on the 1-dimensional complex manifold \( \mathbb{C}^* \)
```

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

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 $\mathbb{C}^*$
```

Its coordinates with respect to 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.subset_family()
Set {A, U, V, $\mathbb{C}^*$} of open subsets of the 1-dimensional complex manifold $\mathbb{C}^*$
sage: M.atlas()
[Chart (U, (z,)), Chart (V, (w,)), Chart (A, (z,)), Chart (A, (w,))]
```

A constant map $\mathbb{C}^* \to \mathbb{C}$:

```
sage: f = M.constant_scalar_field(3+2*I, name='f'); f
Scalar field f on the 1-dimensional complex manifold $\mathbb{C}^*$
(continues on next page)
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 ∂/∂z
sage: v.display(W.frame())
v = -∂/∂w
```

The vector field $v$ acting on the scalar field $f$:

```
sage: v(f)  
Scalar field zero on the 1-dimensional complex manifold $\mathbb{C}^*$
```

Since $f$ is constant, $v(f)$ is vanishing:

```
sage: v(f).display()
zero: $\mathbb{C}^*$ → $\mathbb{C}$
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 $\mathbb{C}^*$
sage: v.at(inf).display()
v = -∂/∂w
sage: v.at(inf).parent()
Tangent space at Point inf on the 1-dimensional complex manifold $\mathbb{C}^*$
```

AUTHORS:

- Eric Gourgoulhon (2015): initial version
- Travis Scrimshaw (2016): review tweaks
- Michael Jung (2020): tensor bundles and orientability
- Matthias Koeppe (2021): refactoring of subsets code
REFERENCES:
• [Lee2013]
• [KN1963]
• [Huy2005]
• [Ser1992]
• [Ber2008]
• [BG1988]

class sage.manifolds.differentiable.manifold.DifferentiableManifold(n, name, field, structure, base_manifold=None, diff_degree=+Infinity, latex_name=None, start_index=0, category=None, unique_tag=None)

Bases: TopologicalManifold

Differentiable manifold over a topological field $K$.

Given a non-discrete topological field $K$ (in most applications, $K = \mathbb{R}$ or $K = \mathbb{C}$; see however [Ser1992] for $K = \mathbb{Q}_p$ and [Ber2008] for other fields), a differentiable manifold over $K$ is a topological manifold $M$ over $K$ equipped with an atlas whose transitions 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 $K = \mathbb{R}$ is assumed;
• if $K = \mathbb{C}$, any $C^k$-manifold with $k \geq 1$ is actually a $C^\infty$-manifold (even an analytic manifold);
• if $K = \mathbb{R}$, 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 R
  – 'complex' or an object of type ComplexField (e.g., CC) for a manifold over 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)
• base_manifold – (default: None) if not None, must be a differentiable manifold; the created object is then an open subset of base_manifold
• 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 \( \mathbb{R} \)):

\[
\text{sage: } M = \text{Manifold}(4, 'M', latex_name='\mathcal{M}'); M
\]

4-dimensional differentiable manifold \( \mathcal{M} \)

\[
\text{sage: type(M)}
\]

<class 'sage.manifolds.differentiable.manifold.DifferentiableManifold_with_category'>

\[
\text{sage: latex(M)}
\]

\( \mathcal{M} \)

\[
\text{sage: dim(M)}
\]

4

Since the base field has not been specified, \( \mathbb{R} \) has been assumed:

\[
\text{sage: } M\text{.base_field()}
\]

Real Field with 53 bits of precision

Since the degree of differentiability has not been specified, the default value, \( C^\infty \), has been assumed:

\[
\text{sage: } M\text{.diff_degree()}
\]

\(+\infty\)

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

\[
\text{sage: } M = \text{Manifold}(4, 'M', \text{start_index}=0)
\]

\[
\text{[0, 1, 2, 3]}
\]

\[
\text{sage: } M = \text{Manifold}(4, 'M', \text{start_index}=1)
\]

\[
\text{[1, 2, 3, 4]}
\]

\[
\text{sage: } M = \text{Manifold}(4, 'M', \text{start_index}=-2).\text{irange()}
\]

\([-2, -1, 0, 1]\)

A complex manifold:

\[
\text{sage: } N = \text{Manifold}(3, 'N', \text{field='complex'}); N
\]

3-dimensional complex manifold \( N \)

A differentiable manifold over \( \mathbb{Q}_5 \), the field of 5-adic numbers:

\[
\text{sage: } N = \text{Manifold}(2, 'N', \text{field=Qp}(5)); N
\]

2-dimensional differentiable manifold \( N \) over the 5-adic Field with capped relative precision 20
A differentiable manifold is of course a topological manifold:

\[
\text{sage: } \text{isinstance}(M, \text{sage.manifolds.manifold.TopologicalManifold}) \\
\text{True} \\
\text{sage: } \text{isinstance}(N, \text{sage.manifolds.manifold.TopologicalManifold}) \\
\text{True}
\]

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

\[
\text{sage: } \text{isinstance}(M, \text{Parent}) \\
\text{True} \\
\text{sage: } M.\text{category}() \\
\text{Category of smooth manifolds over Real Field with 53 bits of precision} \\
\text{sage: from sage.categories.manifolds import Manifolds} \\
\text{sage: } M.\text{category}() \text{ is Manifolds}(\text{RR}).\text{Smooth}() \\
\text{True} \\
\text{sage: } M.\text{category}() \text{ is Manifolds}(M.\text{base_field}()).\text{Smooth}() \\
\text{True} \\
\text{sage: } M \text{ in } \text{Manifolds}(\text{RR}).\text{Smooth}() \\
\text{True} \\
\text{sage: } N \text{ in } \text{Manifolds}(\text{Qp}(5)).\text{Smooth}() \\
\text{True}
\]

The corresponding Sage elements are points:

\[
\text{sage: } X.<t, x, y, z> = M.\text{chart}() \\
\text{sage: } p = M.\text{an_element}(); p \\
\text{Point on the 4-dimensional differentiable manifold M} \\
\text{sage: } p.\text{parent}() \\
\text{4-dimensional differentiable manifold M} \\
\text{sage: } M.\text{is_parent_of}(p) \\
\text{True} \\
\text{sage: } p \text{ in } M \\
\text{True}
\]

The manifold’s points are instances of class \textit{ManifoldPoint}:

\[
\text{sage: } \text{isinstance}(p, \text{sage.manifolds.point.ManifoldPoint}) \\
\text{True}
\]

Since an open subset of a differentiable manifold \( M \) is itself a differentiable manifold, open subsets of \( M \) have all attributes of manifolds:

\[
\text{sage: } U = M.\text{open_subset}('U', \text{coord_def=}{X: t>0}); U \\
\text{Open subset U of the 4-dimensional differentiable manifold M} \\
\text{sage: } U.\text{category}() \\
\text{Join of Category of subobjects of sets and Category of smooth manifolds over Real Field with 53 bits of precision} \\
\text{sage: } U.\text{base_field}() == M.\text{base_field}() \\
\text{True} \\
\text{sage: } \text{dim}(U) == \text{dim}(M) \\
\text{True}
\]

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

\[
\text{sage: } \text{TestSuite}(M).\text{run}()
\]
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:

\[
\text{sage: } M = \text{Manifold(3, 'M', start_index=1)} \\
\text{sage: } A = M\text{.open_subset('A', latex_name=r'\mathcal{A}')}) \\
\text{sage: } nab = A\text{.affine_connection('nabla', r'\nabla')} ; \text{nab}
\]

Affine connection \(nabla\) on the Open subset \(A\) of the 3-dimensional differentiable manifold \(M\)

See also:

AffineConnection for more examples.

automorphism_field(*comp, **kwargs)

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 \(\Phi : M \to N\) 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

\[
\forall p \in M, t(p) \in \text{GL} \left( T_{\Phi(p)}N \right),
\]

where \(\text{GL} \left( T_{\Phi(p)}N \right)\) 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 \(\mathbb{R}\)).

See also:

AutomorphismField and AutomorphismFieldParal for a complete documentation.

INPUT:

• comp – (optional) either the components of the field of automorphisms with respect to the vector frame specified by the argument frame or a dictionary of components, the keys of which are vector frames or pairs \((f, c)\) where \(f\) is a vector frame and \(c\) the chart in which the components are expressed
• frame – (default: None; unused if comp is not given or is a dictionary) vector frame in which the components are given; if None, the default vector frame of self is assumed
• chart – (default: None; unused if comp is not given or is a dictionary) coordinate chart in which the components are expressed; if None, the default chart on the domain of frame is assumed
• 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 2-dimensional manifold:

```sage
M = Manifold(2, 'M')
X.<x,y> = M.chart()
a = M.automorphism_field([[1+x^2, 0], [0, 1+y^2]], name='A')
a
Field of tangent-space automorphisms A on the 2-dimensional differentiable manifold M
```
```
sage: a.parent()
General linear group of the Free module X(M) of vector fields on the 2-dimensional differentiable manifold M
```
```
sage: a(X.frame()[0]).display()
A(\partial/\partial x) = (x^2 + 1) \partial/\partial x
```
```
sage: a(X.frame()[1]).display()
A(\partial/\partial y) = (y^2 + 1) \partial/\partial y
```

For more examples, see AutomorphismField and AutomorphismFieldParal.

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 \( \text{GL}(X(M, \Phi)) \) of the module \( X(M, \Phi) \) 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 \( \text{GL}(X(U, \Phi)) \)

EXAMPLES:
Group of tangent-space automorphism fields of a 2-dimensional differentiable manifold:

```sage
M = Manifold(2, 'M')
G = M.automorphism_field_group()
G
General linear group of the Module X(M) of vector fields on the 2-dimensional differentiable manifold M
```
```
sage: G.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
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
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, (\partial/\partial x, \partial/\partial y)) (default basis)
- (M, (\partial/\partial u, \partial/\partial v))
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
```

2.1. Differentiable Manifolds 347
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, (∂/∂x,∂/∂y))
```

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, (∂/∂x,∂/∂y)),
  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, (∂/∂x,∂/∂y))): 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 $\mathbb{R}^2$
sage: M.coframes()
```

(continues on next page)
[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))]

cotangent_bundle(dest_map=None)
Return the cotangent bundle possibly along a destination map with base space self.
See also:
TensorBundle for complete documentation.
INPUT:
• dest_map = (default: None) destination map \( \Phi : M \to N \) (type: DiffMap) from which the
cotangent bundle is pulled back; if None, it is assumed that \( N = M \) and \( \Phi \) is the identity map of \( M \)
(case of the standard tangent bundle over \( M \))
EXAMPLES:
sage: M = Manifold(2, 'M')
sage: cTM = M.cotangent_bundle(); cTM
Cotangent bundle T*M over the 2-dimensional differentiable
manifold M
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 \((t, t_{\text{min}}, t_{\text{max}})\), where
  – \( t \) is the curve parameter used in coord_expression;
  – \( t_{\text{min}} \) is its minimal value;
- \( t_{\text{max}} \) its maximal value;

if \( t_{\text{min}} = -\infty \) and \( t_{\text{max}} = +\infty \), they can be omitted and \( t \) can be passed for \( \text{param} \) instead of the tuple \((t, t_{\text{min}}, t_{\text{max}})\)

- \text{chart} – (default: \text{None}) chart on the manifold used for case (ii) above; if \text{None} the default chart of the manifold is assumed
- \text{name} – (default: \text{None}) string; symbol given to the curve
- \text{latex_name} – (default: \text{None}) string; \LaTeX\ symbol to denote the curve; if none is provided, \text{name} will be used

**OUTPUT:**

- \text{DifferentiableCurve}

**EXAMPLES:**

The limniscate of Gerono in the 2-dimensional Euclidean plane:

```sage
M = Manifold(2, 'M')
X.<x,y> = M.chart()
R.<t> = manifolds.RealLine()
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 \( t_{\text{min}} \) and \( t_{\text{max}} \) omitted: a helix in \( \mathbb{R}^3 \):

```sage
R3 = Manifold(3, 'R^3')
X.<x,y,z> = R3.chart()
c = R3.curve([cos(t), sin(t), t], t, name='c') ; c
Curve c in the 3-dimensional differentiable manifold R^3
c.domain() # check that t is unbounded
Real number line \( \mathbb{R} \)
```

See also:

\text{DifferentiableCurve} for more examples, including plots.

**\text{de_rham_complex}(\text{dest_map}=\text{None})**

Return the set of mixed forms defined on \text{self}, possibly with values in another manifold, as a graded algebra.

See also:

\text{MixedFormAlgebra} for complete documentation.

**INPUT:**

- \text{dest_map} – (default: \text{None}) destination map, i.e. a differentiable map \( \Phi : M \to N \), where \( M \) is the current manifold and \( N \) a differentiable manifold; if \text{None}, it is assumed that \( N = M \) and that \( \Phi \) is the identity map (case of mixed forms on \( M \)), otherwise \text{dest_map} must be a \text{DiffMap}

**OUTPUT:**

- a \text{MixedFormAlgebra} representing the graded algebra \( \Omega^*(M, \Phi) \) of mixed forms on \( M \) taking values on \( \Phi(M) \subset N \)
EXAMPLES:

Graded algebra of mixed forms on a 2-dimensional manifold:

\begin{verbatim}
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: M.mixed_form_algebra()
Graded algebra Omega^*(M) of mixed differential forms on the
2-dimensional differentiable manifold M
sage: M.mixed_form_algebra().category()
Join of Category of graded algebras over Symbolic Ring and Category of chain...
˓→complexes over Symbolic Ring
sage: M.mixed_form_algebra().base_ring()
Symbolic Ring
\end{verbatim}

The outcome is cached:

\begin{verbatim}
sage: M.mixed_form_algebra() is M.mixed_form_algebra()
True
\end{verbatim}

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:

\begin{verbatim}
sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart()
sage: M.default_frame()
Coordinate frame (M, (∂/∂x,∂/∂y))
\end{verbatim}

degenerate_metric \( \text{name, latex_name=None, dest_map=None} \)

Define a degenerate (or null or lightlike) metric on the manifold.

A degenerate metric is a field of degenerate symmetric bilinear forms acting in the tangent spaces.

See DegenerateMetric 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 tensor field on \( U \))

OUTPUT:

• instance of DegenerateMetric representing the defined degenerate metric.
EXAMPLES:

Lightlike cone:

```python
sage: M = Manifold(3, 'M'); X.<x,y,z> = M.chart()
sage: g = M.degenerate_metric('g'); g
degenerate metric g on the 3-dimensional differentiable manifold M
sage: det(g)
Scalar field zero on the 3-dimensional differentiable manifold M
sage: g.parent()
Free module T^(0,2)(M) of type-(0,2) tensors fields on the
3-dimensional differentiable manifold M
sage: g[0,0], g[ 0,1], g[ 0,2] = (y^2 + z^2)/(x^2 + y^2 + z^2), 
....: - x*y/(x^2 + y^2 + z^2), - x*z/(x^2 + y^2 + z^2)
 sage: g[1,1], g[ 1,2], g[ 2,2] = (x^2 + z^2)/(x^2 + y^2 + z^2), 
....: - y*z/(x^2 + y^2 + z^2), (x ^2 + y^2)/(x^2 + y^2 + z^2)
 sage: g.disp()
 g = (y^2 + z^2)/(x^2 + y^2 + z^2) dx⊗dx - x*y/(x^2 + y^2 + z^2) dx⊗dy
- x*z/(x^2 + y^2 + z^2) dx⊗dz - x*y/(x^2 + y^2 + z^2) dy⊗dx
+ (x^2 + z^2)/(x^2 + y^2 + z^2) dy⊗dy - y*z/(x^2 + y^2 + z^2) dy⊗dz
- x*z/(x^2 + y^2 + z^2) dz⊗dx - y*z/(x^2 + y^2 + z^2) dz⊗dy
+ (x^2 + y^2)/(x^2 + y^2 + z^2) dz⊗dz
```

See also:

`DegenerateMetric` for more examples.

**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:**

```python
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**

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

$$t : M \rightarrow T^{(0,p)} N$$

($T^{(0,p)} N$ being the tensor bundle of type $(0, p)$ over $N$) such that

$$\forall x \in M, \quad t(x) \in \Lambda^p(T^*_{\Phi(x)} N),$$

where $\Lambda^p(T^*_{\Phi(x)} N)$ is the $p$-th exterior power of the dual of the tangent space $T_{\Phi(x)} N$.

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 $\mathbb{R}$).
For $p = 1$, one can use the method \texttt{one_form()} instead.

See also:

\texttt{DiffForm} and \texttt{DiffFormParal} for a complete documentation.

\textbf{INPUT:}

- \texttt{degree} – the degree $p$ of the differential form (i.e. its tensor rank)
- \texttt{comp} – (optional) either the components of the differential form with respect to the vector frame specified by the argument \texttt{frame} or a dictionary of components, the keys of which are vector frames or pairs \((f, c)\) where $f$ is a vector frame and $c$ the chart in which the components are expressed
- \texttt{frame} – (default: None; unused if \texttt{comp} is not given or is a dictionary) vector frame in which the components are given; if None, the default vector frame of \texttt{self} is assumed
- \texttt{chart} – (default: None; unused if \texttt{comp} is not given or is a dictionary) coordinate chart in which the components are expressed; if None, the default chart on the domain of \texttt{frame} is assumed
- \texttt{name} – (default: None) name given to the differential form
- \texttt{latex_name} – (default: None) LaTeX symbol to denote the differential form; if none is provided, the LaTeX symbol is set to \texttt{name}
- \texttt{dest_map} – (default: None) the destination map $\Phi : M \to N$; if None, it is assumed that $N = M$ and that $\Phi$ is the identity map (case of a differential form \textit{on} $M$), otherwise \texttt{dest_map} must be a \texttt{DiffMap}

\textbf{OUTPUT:}

- the $p$-form as a \texttt{DiffForm} (or if $N$ is parallelizable, a \texttt{DiffFormParal})

\textbf{EXAMPLES:}

A 2-form on a 3-dimensional differentiable manifold:

\begin{verbatim}
sage: M = Manifold(3, 'M')
sage: X.<x,y,z> = M.chart()
sage: f = M.diff_form(2, name='F'); f
2-form F on the 3-dimensional differentiable manifold M
sage: f[0,1], f[1,2] = x+y, x*z
sage: f.display()
F = (x + y) dx∧dy + x*z dy∧dz
\end{verbatim}

For more examples, see \texttt{DiffForm} and \texttt{DiffFormParal}.

\texttt{diff_form_module}(\texttt{degree, dest_map=None})

Return the set of differential forms of a given degree defined on \texttt{self}, possibly with values in another manifold, as a module over the algebra of scalar fields defined on \texttt{self}.

See also:

\texttt{DiffFormModule} for complete documentation.

\textbf{INPUT:}

- \texttt{degree} – positive integer; the degree $p$ of the differential forms
- \texttt{dest_map} – (default: None) destination map, i.e. a differentiable map $\Phi : M \to N$, 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 \textit{on} $M$), otherwise \texttt{dest_map} must be a \texttt{DiffMap}

\textbf{OUTPUT:}
• a `DiffFormModule` (or if $N$ is parallelizable, a `DiffFormFreeModule`) representing the module $\Omega^p(M, \Phi)$ of $p$-forms on $M$ taking values on $\Phi(M) \subset N$.

**EXAMPLES:**

Module of $2$-forms on a $3$-dimensional parallelizable manifold:

```python
sage: M = Manifold(3, 'M')
sage: X.<x,y,z> = M.chart()
sage: M.diff_form_module(2)
Free module $\Omega^2(M)$ 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:

```python
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 \texttt{DiffMap}

EXAMPLES:

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

```python
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'\mathbb{R}^3')
sage: c_cart.<x,y,z> = N.chart()  # Cartesian coord. on $\mathbb{R}^3$
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 $\mathbb{R}^3$
```

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

```python
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:

```python
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 $\mathbb{R}^3$
sage: Phi(p).coord(c_cart)
(-1, 0, 0)
sage: Phi1(p) == Phi(p)
True
```

See the documentation of class \texttt{DiffMap} for more examples.

\texttt{diffeomorphism}(\texttt{codomain}=None, \texttt{coord_functions}=None, \texttt{chart1}=None, \texttt{chart2}=None, \texttt{name}=None, \texttt{latex_name}=None)

Define a diffeomorphism between the current manifold and another one.

See \texttt{DiffMap} for a complete documentation.

INPUT:

• \texttt{codomain} – (default: \texttt{None}) codomain of the diffeomorphism (the arrival manifold or some subset of it). If \texttt{None}, the current manifold is taken.

• \texttt{coord_functions} – (default: \texttt{None}) if not \texttt{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 (\texttt{chart1}, \texttt{chart2}) as keys (\texttt{chart1} being a chart on the current manifold and \texttt{chart2} a chart on \texttt{codomain})
  - (ii) a single coordinate expression in a given pair of charts, the latter being provided by the arguments \texttt{chart1} and \texttt{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 \(\mathbb{R}^2\) and \(\mathbb{R}^2\):

```
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), coord_restrictions=lambda x, y: x^2+y^2<1)
....: # Cartesian coord on M
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 \(\mathbb{R}^2\):
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, (∂/∂x,∂/∂y))]
sage: e = M.vector_frame('e')
sage: M.frames()
[Coordinate frame (R^2, (∂/∂x,∂/∂y)),
 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, (∂/∂x,∂/∂y))]
sage: M.frames()
[Coordinate frame (R^2, (∂/∂x,∂/∂y)),
 Vector frame (R^2, (e_0,e_1)),
 Coordinate frame (U, (∂/∂x,∂/∂y))]

**integrated_autoparallel_curve**(*affine_connection, curve_param, initial_tangent_vector*,
\[chart=None, name=None, latex_name=None, verbose=False, across_charts=False\])

Construct an autoparallel curve on the manifold with respect to a given affine connection.

See also:

*IntegratedAutoparallelCurve* for details.

**INPUT:**

- **affine_connection** - `AffineConnection`; affine connection with respect to which the curve is autoparallel
- **curve_param** - a tuple of the type \((t, t_{\text{min}}, t_{\text{max}})\), where
  - \(t\) is the symbolic variable to be used as the parameter of the curve (the equations defining an instance of *IntegratedAutoparallelCurve* are such that \(t\) will actually be an affine parameter of the curve);
  - \(t_{\text{min}}\) is its minimal (finite) value;
  - \(t_{\text{max}}\) its maximal (finite) value.
- **initial_tangent_vector** - `TangentVector`; initial tangent vector of the curve
- **chart** - (default: `None`) chart on the manifold in which the equations are given; 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:**

- *IntegratedAutoparallelCurve*

**EXAMPLES:**

Autoparallel curves associated with the Mercator projection of the 2-sphere \(S^2\):

sage: S2 = Manifold(2, 'S^2', start_index=1)
sage: polar.<th,ph> = S2.chart('th ph')
sage: epolar = polar.frame()
sage: ch_basis = S2.automorphism_field()
Set the affine connection associated with Mercator projection; it is metric compatible but it has non-vanishing torsion:

```python
sage: nab = S2.affine_connection('nab')
sage: nab.set_coef(epolar_ON)[[]
[[[0, 0], [0, 0]], [[0, 0], [0, 0]]]
sage: g = S2.metric('g')
sage: g[1,1], g[2,2] = 1, (sin(th))^2
sage: nab(g)[[]
[[[0, 0], [0, 0]], [[0, 0], [0, 0]]]
sage: nab.torsion()[[]
[[[0, 0], [0, 0]], [[0, cos(th)/sin(th)], [-cos(th)/sin(th), 0]]]
```

Declare an integrated autoparallel curve with respect to this connection:

```python
sage: p = S2.point((pi/4, 0), name = 'p')
sage: Tp = S2.tangent_space(p)
sage: v = Tp((1,1), basis=epolar_ON.at(p))
sage: t = var('t')
sage: c = S2.integrated_autoparallel_curve(nab, (t, 0, 2.3),
......: v, chart = polar, name = 'c')
sage: sys = c.system(verbose=True)
```

Autoparallel curve c in the 2-dimensional differentiable manifold S^2 equipped with Affine connection nab on the 2-dimensional differentiable manifold S^2, and integrated over the Real interval (0, 2.3) as a solution to the following equations, written with respect to Chart (S^2, (th, ph)):

Initial point: Point p on the 2-dimensional differentiable manifold S^2 with coordinates [1/4*pi, 0] with respect to Chart (S^2, (th, ph))
Initial tangent vector: Tangent vector at Point p on the 2-dimensional differentiable manifold S^2 with components [1, sqrt(2)] with respect to Chart (S^2, (th, ph))

\[
\begin{align*}
\frac{d}{dt}(\text{th}) &= D\text{th} \\
\frac{d}{dt}(\text{ph}) &= D\text{ph} \\
\frac{d}{dt}(D\text{th}) &= 0 \\
\frac{d}{dt}(D\text{ph}) &= -D\text{ph}*D\text{th}*\text{cos}(\text{th})/\text{sin}(\text{th})
\end{align*}
\]

```python
sage: sol = c.solve()
sage: interp = c.interpolate()
sage: p = c(1.3, verbose=True)
```

Evaluating point coordinates from the interpolation associated with the key 'cubic spline- interp- odeint' by default...

```python
sage: p
```

Point on the 2-dimensional differentiable manifold S^2

```python
sage: polar(p)  
(2.0853981633974477, 1.4203177070475606)
sage: tgt_vec = c.tangent_vector_eval_at(1.3, verbose=True)
```

Evaluating tangent vector components from the interpolation
associated with the key 'cubic spline-interp-odeint' by default...

\begin{verbatim}
sage: tgt_vec[:]  # abs tol 1e-12
[1.000000000000011, 1.14879968412235]
\end{verbatim}

\texttt{integrated_curve} (\texttt{equations\_rhs}, \texttt{velocities}, \texttt{curve\_param}, \texttt{initial\_tangent\_vector}, \texttt{chart}=None, \\
\texttt{name}=None, \texttt{latex\_name}=None, \texttt{verbose}=False, across\_charts=False)

Construct a curve defined by a system of second order differential equations in the coordinate functions.

See also:

\texttt{IntegratedCurve} for details.

INPUT:

- \texttt{equations\_rhs} – list of the right-hand sides of the equations on the velocities only
- \texttt{velocities} – list of the symbolic expressions used in \texttt{equations\_rhs} to denote the velocities
- \texttt{curve\_param} – a tuple of the type (\texttt{t}, \texttt{t}\_min, \texttt{t}\_max), where
  - \texttt{t} is the symbolic variable used in \texttt{equations\_rhs} to denote the parameter of the curve;
  - \texttt{t}\_min is its minimal (finite) value;
  - \texttt{t}\_max its maximal (finite) value.
- \texttt{initial\_tangent\_vector} – \texttt{TangentVector}; initial tangent vector of the curve
- \texttt{chart} – (default: None) chart on the manifold in which the equations are given; if None the default chart of the manifold is assumed
- \texttt{name} – (default: None) string; symbol given to the curve
- \texttt{latex\_name} – (default: None) string; LaTeX symbol to denote the curve; if none is provided, \texttt{name} will be used

OUTPUT:

- \texttt{IntegratedCurve}

EXAMPLES:

Trajectory of a particle of unit mass and unit charge in a unit, uniform, stationary magnetic field:

\begin{verbatim}
sage: M = Manifold(3, 'M')
sage: X.<x1,x2,x3> = M.chart()
sage: t = var('t')
sage: D = X.symbolic_velocities()
sage: eqns = [D[1], -D[0], SR(0)]
sage: p = M.point((0,0,0), name='p')
sage: Tp = M.tangent_space(p)
sage: v = Tp((1,0,1))
sage: c = M.integrated_curve(eqns, D, (t,0,6), v, name='c'); c
Integrated curve c in the 3-dimensional differentiable manifold M
sage: sys = c.system(verbose=True)
\end{verbatim}

Curve c in the 3-dimensional differentiable manifold M integrated over the Real interval (0, 6) as a solution to the following system, written with respect to Chart (M, (x1, x2, x3)):
Initial point: Point p on the 3-dimensional differentiable manifold M with coordinates [0, 0, 0] with respect to Chart (M, (x1, x2, x3))

Initial tangent vector: Tangent vector at Point p on the 3-dimensional differentiable manifold M with components [1, 0, 1] with respect to Chart (M, (x1, x2, x3))

\[
\begin{align*}
\frac{d(x1)}{dt} &= Dx1 \\
\frac{d(x2)}{dt} &= Dx2 \\
\frac{d(x3)}{dt} &= Dx3 \\
\frac{d(Dx1)}{dt} &= Dx2 \\
\frac{d(Dx2)}{dt} &= -Dx1 \\
\frac{d(Dx3)}{dt} &= 0
\end{align*}
\]

sage: sol = c.solve()
sage: interp = c.interpolate()
sage: p = c(1.3, verbose=True)
Evaluating point coordinates from the interpolation associated with the key 'cubic spline-interp-odeint' by default...
sage: p
Point on the 3-dimensional differentiable manifold M
sage: p.coordinates()  # abs tol 1e-12
(0.9635581599167499, -0.7325011788437327, 1.3)
sage: tgt_vec = c.tangent_vector_eval_at(3.7, verbose=True)
Evaluating tangent vector components from the interpolation associated with the key 'cubic spline-interp-odeint' by default...
sage: tgt_vec[:]
# abs tol 1e-12
[-0.8481007454066425, 0.5298350137284363, 1.0]

`integrated_geodesic` *(metric, curve_param, initial_tangent_vector, chart=None, name=None, latex_name=None, verbose=False, across_charts=False)*

Construct a geodesic on the manifold with respect to a given metric.

See also: `IntegratedGeodesic` for details.

INPUT:

- **metric** -- *PseudoRiemannianMetric*; metric with respect to which the curve is a geodesic
- **curve_param** -- a tuple of the type *(t, t_min, t_max)*, where
  - *t* is the symbolic variable to be used as the parameter of the curve (the equations defining an instance of `IntegratedGeodesic` are such that *t* will actually be an affine parameter of the curve);
  - *t_min* is its minimal (finite) value;
  - *t_max* is its maximal (finite) value.
- **initial_tangent_vector** -- *TangentVector*; initial tangent vector of the curve
- **chart** -- *(default: None)*; chart on the manifold in which the equations are given; 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:

- IntegratedGeodesic

EXAMPLES:

Geodesics of the unit 2-sphere $S^2$:

```
sage: S2 = Manifold(2, 'S^2', start_index=1)
sage: polar.<th,ph> = S2.chart('th ph')
sage: epolar = polar.frame()
```

Set the standard metric tensor $g$ on $S^2$:

```
sage: g = S2.metric('g')
sage: g[1,1], g[2,2] = 1, (sin(th))^2
```

Declare an integrated geodesic with respect to this metric:

```
sage: p = S2.point((pi/4, 0), name='p')
sage: Tp = S2.tangent_space(p)
sage: v = Tp((1, 1), basis=epolar.at(p))
sage: t = var('t')
sage: c = S2.integrated_geodesic(g, (t, 0, 6), v, ....: chart=polar, name='c')
sage: sys = c.system(verbose=True)
```

```
Geodesic c in the 2-dimensional differentiable manifold $S^2$ equipped with Riemannian metric g on the 2-dimensional differentiable manifold $S^2$, and integrated over the Real interval (0, 6) as a solution to the following geodesic equations, written with respect to Chart (S^2, (th, ph)):

Initial point: Point p on the 2-dimensional differentiable manifold $S^2$ with coordinates [1/4*pi, 0] with respect to Chart (S^2, (th, ph))
Initial tangent vector: Tangent vector at Point p on the 2-dimensional differentiable manifold $S^2$ with components [1, 1] with respect to Chart (S^2, (th, ph))

d(th)/dt = Dth
d(ph)/dt = Dph
d(d(th))/dt = Dph^2*cos(th)*sin(th)
d(d(ph))/dt = -2*Dph*Dth*cos(th)/sin(th)
```

```
sage: sol = c.solve()
sage: interp = c.interpolate()
sage: p = c(1.3, verbose=True)
```

```
Evaluating point coordinates from the interpolation associated with the key 'cubic spline- interp- odeint' by default...
```

```
sage: p.coordinates()  # abs tol 1e-12
(2.2047435672397526, 0.7986602654406825)
sage: tgt_vec = c.tangent_vector_eval_at(3.7, verbose=True)
```

```
Evaluating tangent vector components from the interpolation associated with the key 'cubic spline- interp- odeint' by default...
```

```
sage: tgt_vec[:]  # abs tol 1e-12
(continues on next page)
```
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
```

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 \((-+,\cdot\cdot\cdot,+\) or \((+,-,\cdot\cdot\cdot,-\).

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. \((-,+,\cdot\cdot\cdot,+\)
  - if set to 'negative', the signature is -n+2, i.e. \((+,-,\cdot\cdot\cdot,-\)
- 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 \(\Phi : U \to M\), where \(U\) is the current manifold; if None, the identity map is assumed (case of a metric tensor 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

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

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 $\Phi : U \to M$, where $U$ is the current manifold; if None, the identity map is assumed (case of a metric tensor 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_xy.<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.

mixed_form (comp=None, name=None, latex_name=None, dest_map=None)
Define a mixed form on self.
Via the argument dest_map, it is possible to let the mixed form take its values on another manifold. More precisely, if $M$ is the current manifold, $N$ a differentiable manifold, $\Phi : M \to N$ a differentiable map, a \textit{mixed form along} $\Phi$ can be considered as a differentiable map

$$a : M \to \bigoplus_{k=0}^{n} T^{(0,k)} N$$

($T^{(0,k)} N$ being the tensor bundle of type $(0, k)$ over $N$, $\oplus$ being the Whitney sum and $n$ being the dimension of $N$) such that

$$\forall x \in M, \ a(x) \in \bigoplus_{k=0}^{n} \Lambda^k (T^*_{\Phi(x)} N),$$

where $\Lambda^k (T^*_{\Phi(x)} N)$ is the $k$-th exterior power of the dual of the tangent space $T_{\Phi(x)} N$.

The standard case of a mixed form on $M$ corresponds to $N = M$ and $\Phi = \text{Id}_M$.

See also:

\textit{MixedForm} for complete documentation.

INPUT:

- comp – (default: None) homogeneous components of the mixed form as a list; if none is provided, the components are set to innocent unnamed differential forms
- 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 $\Phi : M \to N$; if None, it is assumed that $N = M$ and that $\Phi$ is the identity map (case of a differential form on $M$), otherwise $\text{dest\_map}$ must be a \texttt{DiffMap}

OUTPUT:

- the mixed form as a \texttt{MixedForm}

EXAMPLES:

A mixed form on an open subset of a 3-dimensional differentiable manifold:

```python
sage: M = Manifold(3, 'M')
sage: U = M.open_subset('U', latex_name=r'\mathcal{U}'); U
Open subset U of the 3-dimensional differentiable manifold M
sage: c_xyz.<x,y,z> = U.chart()
sage: f = U.mixed_form(name='F'); f
Mixed differential form F on the Open subset U of the 3-dimensional differentiable manifold M
```

See the documentation of class \texttt{MixedForm} for more examples.

\texttt{mixed\_form\_algebra}(dest\_map=None)

Return the set of mixed forms defined on \texttt{self}, possibly with values in another manifold, as a graded algebra.

See also:

\texttt{MixedFormAlgebra} for complete documentation.

INPUT:
• dest_map – (default: None) destination map, i.e. a differentiable map \( \Phi : M \to N \), 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 mixed forms on \( M \)), otherwise dest_map must be a DiffMap.

OUTPUT:

• a \( \text{MixedFormAlgebra} \) representing the graded algebra \( \Omega^*(M, \Phi) \) of mixed forms on \( M \) taking values on \( \Phi(M) \subset N \)

EXAMPLES:

Graded algebra of mixed forms on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: M.mixed_form_algebra()
Graded algebra Omega^*(M) of mixed differential forms on the
2-dimensional differentiable manifold M
```

The outcome is cached:

```
sage: M.mixed_form_algebra() is M.mixed_form_algebra()
True
```

\texttt{multivector\_field} (*\texttt{args}, **\texttt{kwargs})

Define a multivector field on self.

Via the argument \texttt{dest\_map}, it is possible to let the multivector field take its values on another manifold. More precisely, if \( M \) is the current manifold, \( N \) a differentiable manifold, \( \Phi : M \to N \) a differentiable map and \( p \) a non-negative integer, a \textit{multivector field of degree} \( p \) (or \( p \)-vector field) \textit{along} \( M \) \textit{with values} \textit{on} \( N \) is a differentiable map

\[ t : M \to T^{(p,0)}N \]

\((T^{(p,0)}N\) being the tensor bundle of type \((p, 0)\) over \( N \)) such that

\[ \forall x \in M, \quad t(x) \in \Lambda^p(T_{\Phi(x)}N), \]

where \( \Lambda^p(T_{\Phi(x)}N) \) is the \( p \)-th exterior power of the tangent vector space \( T_{\Phi(x)}N \).

The standard case of a \( p \)-vector field 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 \( \mathbb{R} \)).

For \( p = 1 \), one can use the method \texttt{vector\_field()} instead.

See also:

\texttt{MultivectorField} and \texttt{MultivectorFieldParal} for a complete documentation.

INPUT:

• degree – the degree \( p \) of the multivector field (i.e. its tensor rank)
• comp – (optional) either the components of the multivector field with respect to the vector frame specified by the argument \texttt{frame} or a dictionary of components, the keys of which are vector frames or pairs \((f, c)\) where \( f \) is a vector frame and \( c \) the chart in which the components are expressed
• frame – (default: None; unused if comp is not given or is a dictionary) vector frame in which the components are given; if None, the default vector frame of self is assumed

• chart – (default: None; unused if comp is not given or is a dictionary) coordinate chart in which the components are expressed; if None, the default chart on the domain of frame is assumed

• name – (default: None) name given to the multivector field

• latex_name – (default: None) LaTeX symbol to denote the multivector field; if none is provided, the LaTeX symbol is set to name

• dest_map – (default: None) the destination map $\Phi : M \to N$; if None, it is assumed that $N = M$ and that $\Phi$ is the identity map (case of a multivector field on $M$), otherwise dest_map must be a DiffMap

OUTPUT:

• the $p$-vector field as a MultivectorField (or if $N$ is parallelizable, a MultivectorFieldParal)

EXAMPLES:

A 2-vector field on a 3-dimensional differentiable manifold:

```
sage: M = Manifold(3, 'M')
sage: X.<x,y,z> = M.chart()
sage: h = M.multivector_field(2, name='H'); h
2-vector field H on the 3-dimensional differentiable manifold M
sage: h[0,1], h[0,2], h[1,2] = x+y, x*z, -3
sage: h.display()
H = (x + y) \partial/\partial x \wedge \partial/\partial y + x*z \partial/\partial x \wedge \partial/\partial z - 3 \partial/\partial y \wedge \partial/\partial z
```

For more examples, see MultivectorField and MultivectorFieldParal.

multivector_module (degree, dest_map=None)

Return the set of multivector fields 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:

MultivectorModule for complete documentation.

INPUT:

• degree – positive integer; the degree $p$ of the multivector fields

• dest_map – (default: None) destination map, i.e. a differentiable map $\Phi : M \to N$, 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 multivector fields on $M$), otherwise dest_map must be a DiffMap

OUTPUT:

• a MultivectorModule (or if $N$ is parallelizable, a MultivectorFreeModule) representing the module $\Omega^p(M, \Phi)$ of $p$-forms on $M$ taking values on $\Phi(M) \subset N$

EXAMPLES:

Module of 2-vector fields on a 3-dimensional parallelizable manifold:

```
sage: M = Manifold(3, 'M')
sage: X.<x,y,z> = M.chart()
sage: M.multivector_module(2)
Free module A^2(M) of 2-vector fields on the 3-dimensional
```

differentiable manifold M
sage: M.multivector_module(2).category()
Category of finite dimensional modules over Algebra of
differentiable scalar fields on the 3-dimensional
differentiable manifold M
sage: M.multivector_module(2).base_ring()
Algebra of differentiable scalar fields on the 3-dimensional
differentiable manifold M
sage: M.multivector_module(2).rank()
3

The outcome is cached:

sage: M.multivector_module(2) is M.multivector_module(2)
True

one_form(*comp, **kwargs)

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

$$t : M \rightarrow \mathcal{T}^* N$$

($\mathcal{T}^* N$ being the cotangent bundle of $N$) such that

$$\forall p \in M, \quad t(p) \in T_{\Phi(p)}^* N,$$

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 $\mathbb{R}$).

See also:

DiffForm and DiffFormParal for a complete documentation.

INPUT:

- comp – (optional) either the components of 1-form with respect to the vector frame specified by the
  argument frame or a dictionary of components, the keys of which are vector frames or pairs $(f, c)$
  where $f$ is a vector frame and $c$ the chart in which the components are expressed
- frame – (default: None; unused if comp is not given or is a dictionary) vector frame in which the
  components are given; if None, the default vector frame of self is assumed
- chart – (default: None; unused if comp is not given or is a dictionary) coordinate chart in which the
  components are expressed; if None, the default chart on the domain of frame is assumed
- 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 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: om = M.one_form(-y, 2+x, name='omega', latex_name=r'\omega')
sage: om
1-form omega on the 2-dimensional differentiable manifold M
sage: om.display()
omega = -y dx + (x + 2) dy
sage: om.parent()
Free module Omega^1(M) of 1-forms on the 2-dimensional differentiable manifold M
```

For more examples, see `DiffForm` and `DiffFormParal`.

`open_subset (name, latex_name=None, coord_def={}, supersets=None)`

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.
- `supersets` – (default: only `self`) list of sets that the new open subset is a subset of

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`:
We have then:

\begin{verbatim}
sage: A.subset_family()
Set {A, B} of open subsets of the 2-dimensional differentiable manifold M
sage: B.is_subset(A)
True
sage: B.is_subset(M)
True
\end{verbatim}

Defining an open subset by some coordinate restrictions: the open unit disk in of the Euclidean plane:

\begin{verbatim}
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
\end{verbatim}

Since the argument \texttt{coord\_def} has been set, \texttt{U} is automatically endowed with a chart, which is the restriction of \texttt{X} to \texttt{U}:

\begin{verbatim}
sage: U.atlas()
[Chart (U, (x, y))]
sage: U.default_chart()
Chart (U, (x, y))
sage: U.default_chart() is X.restrict(U)
True
\end{verbatim}

A point in \texttt{U}:

\begin{verbatim}
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
\end{verbatim}

Checking whether various points, defined by their coordinates with respect to chart \texttt{X}, are in \texttt{U}:

\begin{verbatim}
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
\end{verbatim}

\texttt{orientation()}  
Get the preferred orientation of \texttt{self} if available.

An \textit{orientation} on a differentiable manifold is an atlas of charts whose transition maps are pairwise orientation preserving, i.e. whose Jacobian determinants are pairwise positive.

A differentiable manifold with an orientation is called \textit{orientable}.

A differentiable manifold is orientable if and only if the tangent bundle is orientable in terms of a vector bundle, see \texttt{orientation()}.  

2.1. Differentiable Manifolds
Note: In contrast to topological manifolds, see `orientation()`, differentiable manifolds preferably use the notion of orientability in terms of the tangent bundle.

The trivial case corresponds to the manifold being parallelizable, i.e. admitting a frame covering the whole manifold. In that case, if no preferred orientation has been manually set before, one of those frames (usually the default frame) is set to the preferred orientation on self and returned here.

**EXAMPLES:**

In case one frame already covers the manifold, an orientation is readily obtained:

```python
sage: M = Manifold(3, 'M')
sage: c.<x,y,z> = M.chart()
sage: M.orientation()
[Coordinate frame (M, (∂/∂x,∂/∂y,∂/∂z))]
```

However, orientations are usually not easy to obtain:

```python
sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U'); V = M.open_subset('V')
sage: M.declare_union(U, V)
sage: c_xy.<x,y> = U.chart(); c_uv.<u,v> = V.chart()
sage: M.orientation()
[]
```

In that case, the orientation can be set by the user; either in terms of charts or in terms of frames:

```python
sage: M.set_orientation([c_xy, c_uv])
sage: M.orientation()
[Coordinate frame (U, (∂/∂x,∂/∂y)),
 Coordinate frame (V, (∂/∂u,∂/∂v))]
```

The orientation on submanifolds are inherited from the ambient manifold:

```python
sage: W = U.intersection(V, name='W')
sage: W.orientation()
[Vector frame (W, (∂/∂x,∂/∂y))]
```

**poisson_tensor**(name=None, latex_name=None)

Construct a Poisson tensor on the current manifold.

**OUTPUT:**

- instance of `PoissonTensorField`

**EXAMPLES:**

Standard Poisson tensor on $\mathbb{R}^2$:

```python
sage: M.<q, p> = EuclideanSpace(2)
sage: poisson = M.poisson_tensor('varpi')
sage: poisson.set_comp()[1,2] = -1
sage: poisson.display()
varpi = -e_q∧e_p
```
riemannian_metric\( (\text{name}, \text{latex\_name}=\text{None}, \text{dest\_map}=\text{None}) \)

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:

- \text{name} – name given to the metric
- \text{latex\_name} – (default: None) LaTeX symbol to denote the metric; if None, it is formed from name
- \text{dest\_map} – (default: None) instance of class DiffMap representing the destination map \( \Phi : U \to M \), where \( U \) is the current manifold; if None, the identity map is assumed (case of a metric tensor field on \( U \))

OUTPUT:

- instance of PseudoRiemannianMetric representing the defined Riemannian metric.

EXAMPLES:

Metric of the hyperbolic plane \( H^2 \):

\begin{verbatim}
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 \otimes dx + y^(-2) \, dy \otimes dy
sage: g.signature()
2
\end{verbatim}

See also:

PseudoRiemannianMetric for more examples.

set_change_of_frame\( (\text{frame1}, \text{frame2}, \text{change\_of\_frame}, \text{compute\_inverse}=\text{True}) \)

Relate two vector frames by an automorphism.

This updates the internal dictionary self._frame_changes.

INPUT:

- \text{frame1} – frame 1, denoted \((e_i)\) below
- \text{frame2} – frame 2, denoted \((f_i)\) below
- \text{change\_of\_frame} – instance of class AutomorphismFieldParal describing the automorphism \( P \) that relates the basis \((e_i)\) to the basis \((f_i)\) according to \( f_i = P(e_i) \)
- \text{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:

\begin{verbatim}
sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart()
sage: e = M.vector_frame('e')
\end{verbatim}
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)
\text{f}_0 = \text{e}_0
sage: f[1].display(e)
\text{f}_1 = 2 \text{e}_0 + 3 \text{e}_1
sage: e[0].display(f)
\text{e}_0 = \text{f}_0
sage: e[1].display(f)
\text{e}_1 = -2/3 \text{f}_0 + 1/3 \text{f}_1
sage: M.change_of_frame(e, f)[e, :]
\begin{bmatrix}
1 & 2 \\
0 & 3 \\
\end{bmatrix}

\textbf{set_default_frame}(\text{frame})

Changing the default vector frame on \textit{self}.

\textbf{INPUT}:

\begin{itemize}
\item frame – \text{VectorFrame} a vector frame defined on some subset of \textit{self}
\end{itemize}

\textbf{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, (∂/∂x,∂/∂y))
sage: M.set_default_frame(e)
sage: M.default_frame()
Vector frame (M, (\text{e}_0,\text{e}_1))

\textbf{set_orientation}(\text{orientation})

Set the preferred orientation of \textit{self}.

\textbf{INPUT}:

\begin{itemize}
\item orientation – either a chart / list of charts, or a vector frame / list of vector frames, covering \textit{self}
\end{itemize}

\textbf{Warning}: It is the user's responsibility that the orientation set here is indeed an orientation. There is no check going on in the background. See \text{orientation()} for the definition of an orientation.

\textbf{EXAMPLES}:

Set an orientation on a manifold:

sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart(); c_uv.<u,v> = M.chart()
sage: M.set_orientation(c_uv)
sage: M.orientation()
[Coordinate frame (M, (\text{∂}/\text{∂u},\text{∂}/\text{∂v}))]

Instead of a chart, a vector frame can be given, too:
Set an orientation in the non-trivial case:

```python
sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U'); V = M.open_subset('V')
sage: M.declare_union(U, V)
sage: c_xy.<x,y> = U.chart(); c_uv.<u,v> = V.chart()
sage: M.set_orientation([c_xy, c_uv])
sage: M.orientation()
[Coordinate frame (U, (∂/∂x,∂/∂y)),
 Coordinate frame (V, (∂/∂u,∂/∂v))]
Again, the vector frame notion can be used instead:

```python
sage: M.set_orientation([c_xy.frame(), c_uv.frame()])
sage: M.orientation()
[Coordinate frame (U, (∂/∂x,∂/∂y)),
 Coordinate frame (V, (∂/∂u,∂/∂v))]
```

**sym_bilin_form_field** (*comp, **kwargs*)

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 *Φ* : *M* → *N* a differentiable map, a *field of symmetric bilinear forms along *M* with values on *N* is a differentiable map

\[ t : M \rightarrow T^{(0,2)} N \]

(*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*(0,2)*Φ(p)*N*) is the space of symmetric bilinear forms on the tangent space *T*(0,2)*Φ(p)*N*.

The standard case of fields of symmetric bilinear forms on *M* corresponds to *N* = *M* and *Φ* = *Id* *M*. Other common cases are *Φ* being an immersion and *Φ* being a curve in *N* (*M* is then an open interval of *R*).

**INPUT:**

- **comp** – (optional) either the components of the field of symmetric bilinear forms with respect to the vector frame specified by the argument *frame* or a dictionary of components, the keys of which are vector frames or pairs (*f*, *c*) where *f* is a vector frame and *c* the chart in which the components are expressed
- **frame** – (default: None; unused if *comp* is not given or is a dictionary) vector frame in which the components are given; if None, the default vector frame of *self* is assumed
- **chart** – (default: None; unused if *comp* is not given or is a dictionary) coordinate chart in which the components are expressed; if None, the default chart on the domain of *frame* is assumed
- **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 *Φ* : *M* → *N*; if None, it is assumed that *N* = *M* and that *Φ* is the identity map (case of a field on *M*), otherwise *dest_map* must be an instance of instance of class *DiffMap*
OUTPUT:

- a $\text{TensorField}$ (or if $N$ is parallelizable, a $\text{TensorFieldParal}$) of tensor type $(0, 2)$ and symmetric representing the defined field of symmetric bilinear forms

EXAMPLES:

A field of symmetric bilinear forms on a 2-dimensional manifold:

```python
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: t = M.sym_bilin_form_field(name='T'); t
Field of symmetric bilinear forms T on the 2-dimensional differentiable manifold M
```

Such a object is a tensor field of rank 2 and type $(0, 2)$:

```python
sage: t.parent()
Free module $T^{(0,2)}(M)$ of type-$(-0,2)$ tensors fields on the 2-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:

```python
sage: latex(t)
T
sage: om = M.sym_bilin_form_field(name='Omega', latex_name=r'\Omega')
sage: latex(om)
\Omega
```

Setting the components in the manifold’s default vector frame:

```python
sage: t[0,0], t[0,1], t[1,1] = -1, x, x*y
```

The unset components are either zero or deduced by symmetry:

```python
sage: t[1, 0]
x
sage: t[:]
[ -1 x]
[ x x*y]
```

One can also set the components while defining the field of symmetric bilinear forms:

```python
sage: t = M.sym_bilin_form_field([[[-1, x], [x, x*y]], name='T'])
```

A symmetric bilinear form acts on vector pairs:

```python
sage: v1 = M.vector_field(y, x, name='V_1')
sage: v2 = M.vector_field(x+y, 2, name='V_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
```

(continues on next page)
Adding two symmetric bilinear forms results in another symmetric bilinear form:

```python
sage: a = M.sym_bilin_form_field([[1, 2], [2, 3]])
sage: b = M.sym_bilin_form_field([[-1, 4], [4, 5]])
sage: s = a + b; s
Field of symmetric bilinear forms on the 2-dimensional differentiable manifold M
```

```latex
T\left(V_1, V_2\right)
```

But adding a symmetric bilinear form with a non-symmetric bilinear form results in a generic type $(0,2)$ tensor:

```python
sage: c = M.tensor_field(0, 2, [[-2, -3], [1,7]])
sage: s1 = a + c; s1
Tensor field of type $(0,2)$ on the 2-dimensional differentiable manifold M
```

```latex
\begin{bmatrix}
0 & 6 \\
6 & 8 \\
\end{bmatrix}
```

**symplectic_form** *(name=None, latex_name=None)*

Construct a symplectic form on the current manifold.

**OUTPUT:**

- instance of *SymplecticForm*

**EXAMPLES:**

Standard symplectic form on $\mathbb{R}^2$:

```python
sage: M.<q, p> = EuclideanSpace(2)
sage: omega = M.symplectic_form('omega', r'\omega')
sage: omega.set_comp()[1,2] = -1
sage: omega.display()
omega = -dq ∧ dp
```

**tangent_bundle** *(dest_map=None)*

Return the tangent bundle possibly along a destination map with base space *self*.

**See also:**

*TensorBundle* for complete documentation.

**INPUT:**
• dest_map – (default: None) destination map $\Phi : M \to N$ (type: \texttt{DiffMap}) from which the tangent bundle is pulled back; if None, it is assumed that $N = M$ and $\Phi$ is the identity map of $M$ (case of the standard tangent bundle over $M$)

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: TM = M.tangent_bundle(); TM
Tangent bundle TM over the 2-dimensional differentiable manifold M
```

\textbf{tangent\_identity\_field} (dest_map=None)

Return the field of identity maps in the tangent spaces on \texttt{self}.

Via the argument \texttt{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 $\Phi : M \to N$ 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

$$\forall p \in M, \ t(p) = Id_{T_{\Phi(p)} N},$$

where $Id_{T_{\Phi(p)} N}$ 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 $\Phi = Id_M$. Other common cases are $\Phi$ being an immersion and $\Phi$ being a curve in $N$ ($M$ is then an open interval of $\mathbb{R}$).

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 ‘\texttt{mathrm{Id}}’ if name is ‘Id’ and to name otherwise

• dest_map – (default: None) the destination map $\Phi : M \to N$; 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 \texttt{dest\_map} must be a \texttt{DiffMap}

OUTPUT:

• a \texttt{AutomorphismField} (or if $N$ is parallelizable, a \texttt{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
```

For more examples, see \texttt{AutomorphismField}.

\textbf{tangent\_space} (point, base\_ring=None)

Tangent space to \texttt{self} at a given point.

INPUT:
• point – ManifoldPoint; point \( p \) on the manifold
• base_ring – (default: the symbolic ring) the base ring

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:

```python
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.

```python
tangent_vector(*args, **kwargs)
```

Define a tangent vector at a given point of \( \text{self} \).

INPUT:
• point – ManifoldPoint; point \( p \) on \( \text{self} \)
• comp – components of the vector with respect to the basis specified by the argument basis, either as an iterable or as a sequence of \( n \) components, \( n \) being the dimension of \( \text{self} \) (see examples below)
• basis – (default: None) FreeModuleBasis; basis of the tangent space at \( p \) with respect to which the components are defined; if None, the default basis of the tangent space is used
• 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

OUTPUT:
• **TangentVector** representing the tangent vector at point \( p \)

EXAMPLES:
Vector at a point \( p \) of the Euclidean plane:

```python
sage: E.<x,y>= EuclideanSpace()
sage: p = E((1, 2), name='p')
sage: v = E.tangent_vector(p, -1, 3, name='v'); v
Vector v at Point p on the Euclidean plane E^2
sage: v.display()
v = -e_x + 3 e_y
sage: v.parent()
Tangent space at Point p on the Euclidean plane E^2
sage: v in E.tangent_space(p)
True
```
An alias of `tangent_vector` is `vector`:

```
sage: v = E.vector(p, -1, 3, name='v'); v
Vector v at Point p on the Euclidean plane E^2
```

The components can be passed as a tuple or a list:

```
sage: v1 = E.vector(p, (-1, 3)); v1
Vector at Point p on the Euclidean plane E^2
sage: v1 == v
True
```

or as an object created by the `vector` function:

```
sage: v2 = E.vector(p, vector([-1, 3])); v2
Vector at Point p on the Euclidean plane E^2
sage: v2 == v
True
```

Example of use with the options `basis` and `latex_name`:

```
sage: polar_basis = E.polar_frame().at(p)
sage: v = E.vector(p, 2, -1, basis=polar_basis, name='v', latex_name=r'\vec{v}')
sage: v
Vector v at Point p on the Euclidean plane E^2
sage: v.display(polar_basis)
v = 2 e_r - e_ph
sage: latex(v)
\vec{v}
```

tensor_bundle\( (k, l, \text{dest\_map=\text{None}}) \)

Return a tensor bundle of type \((k, l)\) defined over `self`, possibly along a destination map.

**INPUT:**

- `k` – the contravariant rank of the tensor bundle
- `l` – the covariant rank of the tensor bundle
- `dest_map` – (default: `None`) destination map \(\Phi : M \to N\) (type: `DiffMap`) from which the tensor bundle is pulled back; if `None`, it is assumed that \(N = M\) and \(\Phi\) is the identity map of \(M\) (case of the standard tangent bundle over \(M\))

**OUTPUT:**

- a `TensorBundle` representing a tensor bundle of type-\((k, l)\) over `self`

**EXAMPLES:**

A tensor bundle over a parallelizable 2-dimensional differentiable manifold:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()  # makes M parallelizable
sage: M.tensor_bundle(1, 2)
Tensor bundle T^(1,2)M over the 2-dimensional differentiable manifold M
```
The special case of the tangent bundle as tensor bundle of type (1,0):

```
 sage: M.tensor_bundle(1,0)
 Tangent bundle TM over the 2-dimensional differentiable manifold M
```

The result is cached:

```
 sage: M.tensor_bundle(1, 2) is M.tensor_bundle(1, 2)
 True
```

See also:

`TensorBundle` for more examples and documentation.

**tensor_field(***args, **kwargs)**

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, `Φ : M → 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

\[ t : M → T^{(k,l)}N \]

\((T^{(k,l)}N\) being the tensor bundle of type \((k,l)\) over \(N\)) such that

\[ ∀p ∈ M, t(p) ∈ T^{(k,l)}(T_{Φ(p)}N), \]

where \(T^{(k,l)}(T_{Φ(p)}N)\) is the space of tensors of type \((k,l)\) on the tangent space \(T_{Φ(p)}N\).

The standard case of tensor fields on `M` corresponds to `N = M` and `Φ = Id_M`. Other common cases are `Φ` being an immersion and `Φ` being a curve in `N` (`M` is then an open interval of `ℝ`).

See also:

`TensorField` and `TensorFieldParal` 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 \((k,l)\)
- `comp` – (optional) either the components of the tensor field with respect to the vector frame specified by the argument `frame` or a dictionary of components, the keys of which are vector frames or pairs \((f, c)\) where `f` is a vector frame and `c` the chart in which the components are expressed
- `frame` – (default: `None`; unused if `comp` is not given or is a dictionary) vector frame in which the components are given; if `None`, the default vector frame of `self` is assumed
- `chart` – (default: `None`; unused if `comp` is not given or is a dictionary) coordinate chart in which the components are expressed; if `None`, the default chart on the domain of `frame` is assumed
- `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
\texttt{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

- \texttt{antisym} – (default: \texttt{None}) antisymmetry or list of antisymmetries among the arguments, with the same convention as for \texttt{sym}

- \texttt{dest_map} – (default: \texttt{None}) the destination map \( \Phi : M \rightarrow N \); if \texttt{None}, it is assumed that \( N = M \) and that \( \Phi \) is the identity map (case of a tensor field \texttt{on M}), otherwise \texttt{dest_map} must be a \texttt{DiffMap}

\textbf{OUTPUT:}

- a \texttt{TensorField} (or if \( N \) is parallelizable, a \texttt{TensorFieldParal}) representing the defined tensor field

\textbf{EXAMPLES:}

A tensor field of type \((2,0)\) on a 2-dimensional differentiable manifold:

\begin{verbatim}
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: t = M.tensor_field(2, 0, [[1+x, -y], [0, x*y]], name='T'); t
Tensor field T of type (2,0) on the 2-dimensional differentiable
manifold M
sage: t.display()
T = (x + 1) \partial/\partial x \otimes \partial/\partial x - y \partial/\partial x \otimes \partial/\partial y + x*y \partial/\partial y \otimes \partial/\partial y
\end{verbatim}

The type \((2,0)\) tensor fields on \( M \) form the set \( T^{(2,0)}(M) \), which is a module over the algebra \( C^k(M) \) of differentiable scalar fields on \( M \):

\begin{verbatim}
sage: t.parent() Free module T^(2,0)(M) of type-(2,0) tensors fields on the
2-dimensional differentiable manifold M
sage: t in M.tensor_field_module((2,0)) True
\end{verbatim}

For more examples, see \texttt{TensorField} and \texttt{TensorFieldParal}.

\texttt{tensor_field_module} (\texttt{tensor_type, dest_map=None})

Return the set of tensor fields of a given type defined on \texttt{self}, possibly with values in another manifold, as a module over the algebra of scalar fields defined on \texttt{self}.

\textbf{See also:}
\texttt{TensorFieldModule} for a complete documentation.

\textbf{INPUT:}

- \texttt{tensor_type} – pair \((k,l)\) with \( k \) being the contravariant rank and \( l \) the covariant rank

- \texttt{dest_map} – (default: \texttt{None}) destination map, i.e. a differentiable map \( \Phi : M \rightarrow N \), where \( M \) is the current manifold and \( N \) a differentiable manifold; if \texttt{None}, it is assumed that \( N = M \) and that \( \Phi \) is the identity map (case of tensor fields \texttt{on M}), otherwise \texttt{dest_map} must be a \texttt{DiffMap}

\textbf{OUTPUT:}

- a \texttt{TensorFieldModule} (or if \( N \) is parallelizable, a \texttt{TensorFieldFreeModule}) representing the module \( T^{(k,l)}(M, \Phi) \) of type-\((k,l)\) tensor fields on \( M \) taking values on \( \Phi(M) \subset N \)

\textbf{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 tensor products 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()
\(2 \partial/\partial x \otimes \partial/\partial x \otimes dx\)

vector (*args, **kwargs)

Define a tangent vector at a given point of self.

INPUT:

- **point** – *ManifoldPoint*: point \(p\) on self
- **comp** – components of the vector with respect to the basis specified by the argument *basis*, either as an iterable or as a sequence of \(n\) components, \(n\) being the dimension of self (see examples below)
- **basis** – (default: None)*FreeModuleBasis*: basis of the tangent space at \(p\) with respect to which the components are defined; if None, the default basis of the tangent space is used
- **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

OUTPUT:

- **TangentVector** representing the tangent vector at point \(p\)

EXAMPLES:

Vector at a point \(p\) of the Euclidean plane:

sage: E.<x,y> = EuclideanSpace()
sage: p = E((1, 2), name='p')
sage: v = E.tangent_vector(p, -1, 3, name='v'); v
Vector v at Point p on the Euclidean plane \(E^2\)
sage: v.display()
v = -e_x + 3 e_y
sage: v.parent()
Tangent space at Point p on the Euclidean plane \(E^2\)
sage: v in E.tangent_space(p)
True

An alias of tangent_vector is vector:
Manifolds, Release 10.3

```sage
v = E.vector(p, -1, 3, name=v); v
Vector v at Point p on the Euclidean plane E^2
```

The components can be passed as a tuple or a list:

```sage
v1 = E.vector(p, (-1, 3)); v1
Vector at Point p on the Euclidean plane E^2
sage: v1 == v
True
```

or as an object created by the `vector` function:

```sage
v2 = E.vector(p, vector([-1, 3])); v2
Vector at Point p on the Euclidean plane E^2
sage: v2 == v
True
```

Example of use with the options `basis` and `latex_name`:

```sage
polar_basis = E.polar_frame().at(p)
sage: polar_basis
Basis (e_r,e_ph) on the Tangent space at Point p on the Euclidean plane E^2
sage: v = E.vector(p, 2, -1, basis=polar_basis, name=v, latex_name=r'\vec{v}'); v
Vector v at Point p on the Euclidean plane E^2
sage: v.display(polar_basis)
v = 2 e_r - e_ph
sage: v.display()
v = 4/5*sqrt(5) e_x + 3/5*sqrt(5) e_y
sage: latex(v)
\vec{v}
```

The `vector_bundle` function:

```sage
vector_bundle(rank, name='E', field='real', latex_name=None)
```

Return a differentiable vector bundle over the given field with given rank over this differentiable manifold of the same differentiability class as the manifold.

**INPUT:**

- `rank` – rank of the vector bundle
- `name` – name given to the total space
- `field` – (default: `'real'`) topological field giving the vector space structure to the fibers
- `latex_name` – optional LaTeX name for the total space

**OUTPUT:**

- a differentiable vector bundle as an instance of `DifferentiableVectorBundle`

**EXAMPLES:**

```sage
M = Manifold(2, 'M')
sage: M.vector_bundle(2, 'E')
Differentiable real vector bundle E -> M of rank 2 over the base space 2-dimensional differentiable manifold M
```
**vector_field**(*comp, **kwargs*)

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 $\Phi : M \to N$ a differentiable map, a vector field along $M$ with values on $N$ is a differentiable map

$$v : M \to TN$$

($TN$ 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 $\mathbb{R}$).

See also:

*VectorField* and *VectorFieldParal* for a complete documentation.

**INPUT:**

- `comp` – (optional) either the components of the vector field with respect to the vector frame specified by the argument `frame` or a dictionary of components, the keys of which are vector frames or pairs $(f, c)$ where $f$ is a vector frame and $c$ the chart in which the components are expressed
- `frame` – (default: None; unused if `comp` is not given or is a dictionary) vector frame in which the components are given; if None, the default vector frame of self is assumed
- `chart` – (default: None; unused if `comp` is not given or is a dictionary) coordinate chart in which the components are expressed; if None, the default chart on the domain of `frame` is assumed
- `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 $\Phi : M \to N$; 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(y, -x*z, 1+y, name='v'); v
Vector field v on the Open subset U of the 3-dimensional differentiable manifold M
sage: v.display()
v = y ∂/∂x - x*z ∂/∂y + (y + 1) ∂/∂z
```

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 $U$.

---

2.1. Differentiable Manifolds
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 \( \Phi : M \to N \), 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 \( C^k(M, \Phi) \)-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 \( S^2 \):

\[
\text{sage: } S2 = \text{Manifold}(2, 'S^2') \\
\text{sage: } U = S2.open_subset('U') \quad \# \text{the complement of the two poles} \\
\text{sage: } \text{spher_coord.<th,ph> = U.chart(r'\theta:(0,pi)\phi:(0,2*pi)')} \quad \# \text{spherical coordinates} \\
\text{sage: } XU = U.vector_field_module() ; XU \\
\text{Free module } X(U) \text{ of vector fields on the Open subset } U \text{ of the 2-dimensional differentiable manifold } S^2 \\
\text{sage: } XU.category() \\
\text{Category of finite dimensional modules over Algebra of differentiable scalar fields on the Open subset } U \text{ of the 2-dimensional differentiable manifold } S^2 \\
\text{sage: } XU.base_ring() \\
\text{Algebra of differentiable scalar fields on the Open subset } U \text{ of the 2-dimensional differentiable manifold } S^2 \\
\text{sage: } XU.base_ring() \text{ is } U\text{.scalar_field_algebra()} \\
\text{True}
\]

\( \mathcal{X}(U) \) is a free module because \( U \) is parallelizable (being a chart domain):

\[
\text{sage: } U\text{.is_manifestly_parallelizable()} \\
\text{True}
\]

Its rank is the manifold's dimension:

\[
\text{sage: } XU\text{.rank()} \\
2
\]

The elements of \( \mathcal{X}(U) \) are vector fields on \( U \):
Vector field module $\mathfrak{X}(U, \Phi)$ of the $\mathbb{R}^3$-valued vector fields along $U$, associated with the embedding $\Phi$ of $S^2$ into $\mathbb{R}^3$:

$sage$: 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 $\mathbb{R}^3$

$sage$: XU_R3.base_ring()
Algebra of differentiable scalar fields on the Open subset $U$ of the 2-dimensional differentiable manifold $S^2$

$\mathfrak{X}(U, \Phi)$ is a free module because $\mathbb{R}^3$ is parallelizable and its rank is 3:

$sage$: XU_R3.rank()
3

Without any information on the manifold, the vector field module is not free by default:

$sage$: M = Manifold(2, 'M')
sage: XM = M.vector_field_module()
sage: isinstance(XM, FiniteRankFreeModule)
False

In particular, declaring a coordinate chart on $M$ would yield an error:

$sage$: X.<x,y> = M.chart()
Traceback (most recent call last):
...
ValueError: the Module $X(M)$ of vector fields on the 2-dimensional differentiable manifold $M$ has already been constructed as a non-free module, which implies that the 2-dimensional differentiable manifold $M$ is not parallelizable and hence cannot be the domain of a coordinate chart

Similarly, one cannot declare a vector frame on $M$:

$sage$: e = M.vector_frame('e')
Traceback (most recent call last):
...
ValueError: the Module $X(M)$ of vector fields on the 2-dimensional differentiable manifold $M$ has already been constructed as a non-free module and therefore cannot have a basis

One shall use the keyword `force_free=True` to construct a free module before declaring the chart:
If one declares the chart or the vector frame before asking for the vector field module, the latter is initialized as a free module, without the need to specify `force_free=True`. Indeed, the information that $M$ is the domain of a chart or a vector frame implies that $M$ is parallelizable and is therefore sufficient to assert that $\mathcal{X}(M)$ is a free module over $C^\infty(M)$:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: XM = M.vector_field_module()
sage: isinstance(XM, FiniteRankFreeModule)
True
sage: M.is_manifestly_parallelizable()
True
```

`vector_frame(*args, **kwargs)`

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 $p$ (or at $\Phi(p)$ when `dest_map` is not `None`, see below).

The vector frame can be defined from a set of $n$ linearly independent vector fields, $n$ being the dimension of `self`.

See also:

`VectorFrame` for complete documentation.

INPUT:

- `symbol` – either a string, to be used as a common base for the symbols of the vector fields constituting the vector frame, or a list/tuple of strings, representing the individual symbols of the vector fields; can be omitted only if `from_frame` is not `None` (see below)
- `vector_fields` – tuple or list of $n$ linearly independent vector fields on the manifold `self` ($n$ being the dimension of `self`) defining the vector frame; can be omitted if the vector frame is created from scratch or if `from_frame` is not `None`
- `latex_symbol` – (default: None) either a string, to be used as a common base for the LaTeX symbols of the vector fields constituting the vector frame, or a list/tuple of strings, representing the individual LaTeX symbols of the vector fields; if `None`, `symbol` is used in place of `latex_symbol`
- `dest_map` – (default: `None`) `DiffMap`; destination map $\Phi : U \to M$, where $U$ is `self` and $M$ is a differentiable manifold; for each $p \in U$, the vector frame evaluated at $p$ is a basis of the tangent space $T_{\Phi(p)}M$; if `dest_map` is `None`, the identity map is assumed (case of a vector frame on $U$)
- `from_frame` – (default: `None`) vector frame $\tilde{e}$ on the codomain $M$ of the destination map $\Phi$; the returned frame $e$ is then such that for all $p \in U$, we have $e(p) = \tilde{e}(\Phi(p))$
- `indices` – (default: `None`; used only if `symbol` is a single string) tuple of strings representing the indices labelling the vector fields of the frame; if `None`, the indices will be generated as integers within the range declared on `self`
- `latex_indices` – (default: `None`) tuple of strings representing the indices for the LaTeX symbols of the vector fields; if `None`, `indices` is used instead
- `symbol_dual` – (default: `None`) same as `symbol` but for the dual coframe; if `None`, `symbol` must be a string and is used for the common base of the symbols of the elements of the dual coframe
• \texttt{latex\_symbol\_dual} \texttt{\text{\texttt{(default: None) same as latex\_symbol but for the dual coframe\text}}} \\

\textbf{OUTPUT:}

• a \texttt{VectorFrame} representing the defined vector frame

\textbf{EXAMPLES:}

Defining a vector frame from two linearly independent vector fields on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: e0 = M.vector_field(1+x^2, 1+y^2)
sage: e1 = M.vector_field(2, -x*y)
sage: e = M.vector_frame('e', (e0, e1)); e
Vector frame (M, (e_0,e_1))
sage: e[0].display()
e_0 = (x^2 + 1) \partial/\partial x + (y^2 + 1) \partial/\partial y
sage: e[1].display()
e_1 = 2 \partial/\partial x - x*y \partial/\partial y
sage: (e[0], e[1]) == (e0, e1)
True
```

If the vector fields are not linearly independent, an error is raised:

```
sage: z = M.vector_frame('z', (e0, -e0))
Traceback (most recent call last):
  ... ValueError: the provided vector fields are not linearly independent
```

Another example, involving a pair vector fields along a curve:

```
sage: R.<t> = manifolds.RealLine()
sage: c = M.curve([sin(t), sin(2*t)/2], (t, 0, 2*pi), name='c')
sage: I = c.domain(); I
Real interval (0, 2*pi)
sage: v = c.tangent_vector_field()
sage: v.display()
c' = cos(t) \partial/\partial x + (2*cos(t)^2 - 1) \partial/\partial y
sage: w = I.vector_field(1-2*cos(t)^2, cos(t), dest_map=c)
sage: u = I.vector_frame('u', (v, w))
sage: u[0].display()
u_0 = cos(t) \partial/\partial x + (2*cos(t)^2 - 1) \partial/\partial y
sage: u[1].display()
u_1 = (-2*cos(t)^2 + 1) \partial/\partial x + cos(t) \partial/\partial y
sage: (u[0], u[1]) == (v, w)
True
```

It is also possible to create a vector frame from scratch, without connecting it to previously defined vector frames or vector fields (this can still be performed later via the method \texttt{set\_change\_of\_frame()}):

```
sage: f = M.vector_frame('f'); f
Vector frame (M, (f_0,f_1))
sage: f[0]
Vector field f_0 on the 2-dimensional differentiable manifold M
```

Thanks to the keywords \texttt{dest\_map} and \texttt{from\_frame}, one can also define a vector frame from one pre-existing on another manifold, via a differentiable map (here provided by the curve \texttt{c}):
Note that the symbol for $fc$, namely $f$, is inherited from $f$, the original vector frame.

See also:

For more options, in particular for the choice of symbols and indices, 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, $K = \mathbb{R}$ or $K = \mathbb{C}$).

The subclass `RealDiffChart` is devoted to the case $K = \mathbb{R}$, 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:**

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

**REFERENCES:**

• Chap. 1 of [Lee2013]

```python
sage: fc = I.vector_frame(dest_map=c, from_frame=f); fc
Vector frame ((0, 2*pi), (f_0,f_1)) with values on the 2-dimensional differentiable manifold M
sage: fc[0]
Vector field f_0 along the Real interval (0, 2*pi) with values on the 2-dimensional differentiable manifold M
```

The components $(x^1, \ldots, x^n)$ of $\varphi$, defined by $\varphi(p) = (x^1(p), \ldots, x^n(p)) \in K^n$ 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 three fields, separated by a colon (`:`):
  1. the coordinate symbol (a letter or a few letters)
  2. (optional) the period of the coordinate if the coordinate is periodic; the period field must be written as `period=T`, where $T$ is the period (see examples below)
  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 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’s backslash character (see examples below). If no period and no LaTeX spelling are to be set for any coordinate, the argument coordinates can be omitted when the shortcut operator <, > is used to declare the chart (see examples below).

- **calc_method** – (default: None) string defining the calculus method for computations involving coordinates of the chart; must be one of
  - ‘SR’: Sage’s default symbolic engine (Symbolic Ring)
  - ‘sympy’: SymPy
  - None: the default of CalculusMethod will be used

- **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).

- **coord_restrictions**: Additional restrictions on the coordinates. 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 (or set or frozenset) coord_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 (or set or frozenset) coord_restrictions. For example:

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

  means $(x > y)$ and $(x \neq 0$ or $y \neq 0)$ and $(z^2 < x)$. If the list coord_restrictions contains only one item, this item can be passed as such, i.e. writing $x > y$ instead of the single element list $[x > y]$. If the chart variables have not been declared as variables yet, coord_restrictions must be lambda-quoted.

EXAMPLES:

A chart on a complex 2-dimensional differentiable manifold:

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

The coordinates are then immediately accessible:

```sage
y
is
x is X[0] and y is X[1]
True
```
The trick is performed by Sage preparser:

```
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.

Periodic coordinates are declared through the keyword $\text{period}$ in the coordinate field:

```
sage: N = Manifold(2, 'N', field='complex')
sage: XN.<Z1,Z2> = N.chart('Z1: period=1+2*I Z2')
sage: XN.periods()
(2*I + 1, None)
```

Coordinates are Sage symbolic variables (see $\text{sage.symbolic.expression}$):
sage: type(z1)
<class '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 a 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, (∂/∂x,∂/∂y))
sage: Y.frame()
Coordinate frame (U, (∂/∂z1,∂/∂z2))
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 \( \mathbb{R} \).

**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_with_category'>
```

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 \partial/\partial x on the 2-dimensional differentiable manifold M
sage: ey = c_xy.frame()[1] ; ey
Vector field \partial/\partial y on the 2-dimensional differentiable manifold M
sage: dx(ex).display()
dx(∂/∂x): M → ℝ
  (x, y) ↦ 1
sage: dx(ey).display()
dx(∂/∂y): M → ℝ
  (x, y) ↦ 0
sage: dy(ex).display()
dy(∂/∂x): M → ℝ
  (x, y) ↦ 0
sage: dy(ey).display()
dy(∂/∂y): 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: frame()
```
Manifolds, Release 10.3

sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart()
sage: c_xy.frame()
Coordinate frame (M, (\partial/\partial x, \partial/\partial y))
sage: type(c_xy.frame())
<class 'sage.manifolds.differentiable.vectorframe.CoordFrame_with_category'>

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 \(\partial/\partial x\) on the 2-dimensional differentiable manifold \(M\)
sage: ey = c_xy.frame()[1] ; ey
Vector field \(\partial/\partial y\) on the 2-dimensional differentiable manifold \(M\)
sage: ex(M.scalar_field(x)).display()
1: M → ℝ
(x, y) ↦ 1
sage: ex(M.scalar_field(y)).display()
zero: M → ℝ
(x, y) ↦ 0
sage: ey(M.scalar_field(x)).display()
zero: M → ℝ
(x, y) ↦ 0
sage: ey(M.scalar_field(y)).display()
1: 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))

symbolic_velocities (left='D', right=None)

Return a list of symbolic variables ready to be used by the user as the derivatives of the coordinate functions with respect to a curve parameter (i.e. the velocities along the curve). It may actually serve to denote anything else than velocities, with a name including the coordinate functions. The choice of strings provided as ‘left’ and ‘right’ arguments is not entirely free since it must comply with Python prescriptions.

INPUT:

• left – (default: D) string to concatenate to the left of each coordinate functions of the chart
• right – (default: None) string to concatenate to the right of each coordinate functions of the chart

OUTPUT:

• a list of symbolic expressions with the desired names

EXAMPLES:

Symbolic derivatives of the Cartesian coordinates of the 3-dimensional Euclidean space:

sage: R3 = Manifold(3, 'R3', start_index=1)
sage: cart.<X,Y,Z> = R3.chart()
sage: D = cart.symbolic_velocities(); D
[DX, DY, DZ]
sage: D = cart.symbolic_velocities(left='d', right='/dt'); D
[dx/dt, dy/dt, dz/dt]
sage: D = cart.symbolic_velocities(left='', right='''''); D
[dx, dy, dz]

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

\[
\psi \circ \varphi^{-1} : \varphi(U \cap V) \subset K^n \to \psi(U \cap V) \subset K^n,
\]

where \(K\) is the manifold’s base field. In other words, the transition map expresses the coordinates \((y^1, \ldots, y^n)\) of \((V, \psi)\) in terms of the coordinates \((x^1, \ldots, x^n)\) 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. $\text{diff\_degree()}$).

**INPUT:**

- **other** – the chart $(V, \psi)$
- **transformations** – tuple (or list) $(Y_1, \ldots, Y_2)$, where $Y_i$ is the symbolic expression of the coordinate $y^i$ in terms of the coordinates $(x^1, \ldots, x^n)$
- **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 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!=0) or (y!=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 $\text{DiffCoordChange}$.

**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 $\text{transition\_map}()$:

```
sage: F = M.subset\_family(); F
```

```
Set (S^1, U, V, W) of open subsets of the 1-dimensional differentiable...
```

```
\_manifold S^1
```

```
sage: W = F['W']
```

```
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 $\mathbb{R}^2$:

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

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

(continues on next page)
sage: U = M.open_subset('U') # the complement of the half line (y=0, x >= 0)
sage: c_spher.<r,phi> = U.chart(rr:(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.subset_family()
Set {R^2, U} of open subsets 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: 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

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

In other words, the transition map $\psi \circ \varphi^{-1}$ expresses the coordinates $(y^1, \ldots, y^n)$ of $(V, \psi)$ in terms of the coordinates $(x^1, \ldots, x^n)$ 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) $(Y_1, \ldots, Y_2)$, where $Y_i$ is the symbolic expression of the coordinate $y^i$ in terms of the coordinates $(x^1, \ldots, x^n)$

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

\[ y^i = Y^i(x^1, \ldots, x^n) \quad 1 \leq i \leq n \]

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. `ChartFunction`)

**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 chart 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. `ChartFunction`)

**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:
class sage.manifolds.differentiable.chart.RealDiffChart(domain, coordinates=None, calc_method=None, bounds=None, periods=None, coord_restrictions=None)

Bases: DiffChart, RealChart

Chart on a differentiable manifold over \( \mathbb{R} \).

Given a differentiable manifold \( M \) of dimension \( n \) over \( \mathbb{R} \), a chart is a member \((U, \varphi)\) of the manifold's differentiable atlas; \( U \) is then an open subset of \( M \) and \( \varphi : U \rightarrow V \subseteq \mathbb{R}^n \) is a homeomorphism from \( U \) to an open subset \( V \) of \( \mathbb{R}^n \).

The components \((x^1, \ldots, x^n)\) of \( \varphi \), defined by \( \varphi(p) = (x^1(p), \ldots, x^n(p)) \in \mathbb{R}^n \) 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, with ’ ’ (whitespace) as a separator; each item has at most four 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 \( \mathbb{R} \); otherwise \( I \) must be provided in the form \((a, b)\) (or equivalently \( \left[a, b\right] \)); the bounds \( a \) and \( b \) can be +/- Infinity, Inf, infinity, inf or oo; for singular coordinates, non-open intervals such as \( \left[a, b\right] \) and \( (a, b) \) (or equivalently \( \left[a, b\right] \)) are allowed; note that the interval declaration must not contain any whitespace
  3. (optional) indicator of the periodic character of the coordinate, either as period=T, where \( T \) is the period, or as the keyword periodic (the value of the period is then deduced from the interval \( I \) declared in field 2; see examples below)
  4. (optional) the \LaTeX\ spelling of the coordinate; if not provided the coordinate symbol given in the first field will be used

The order of fields 2 to 4 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\’s backslash character (see examples below). If interval range, no period and no \LaTeX\ spelling are to be set for any coordinate, the argument coordinates can be omitted when the shortcut operator <,> is used to declare the chart (see examples below).

- calc_method – (default: None) string defining the calculus method for computations involving coordinates of the chart; must be one of
  - ‘SR’: Sage’s default symbolic engine (Symbolic Ring)
  - ‘sympy’: SymPy
  - None: the default of CalculusMethod will be used
- 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).
- coord_restrictions: Additional restrictions on the coordinates. 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 (or set or frozenset) coord_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 (or set or frozenset) coord_restrictions. For example:

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

means \((x > y)\) and \((x != 0, y != 0)\) and \((z^2 < x)\). If the list coord_restrictions contains only one item, this item can be passed as such, i.e. writing \(x > y\) instead of the single element list \([x > y]\). If the chart variables have not been declared as variables yet, coord_restrictions must be lambda-quoted.

EXAMPLES:

Cartesian coordinates on \(\mathbb{R}^3\):

```python
sage: M = Manifold(3, 'R^3', r'\mathbb{R}^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:

```python
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()):

```python
sage: M = Manifold(3, 'R^3', r'\mathbb{R}^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:

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

The trick is performed by Sage preparser:

```python
sage: preparse("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 \(<x,y,z>\) are mere Python variable names and do not have to coincide with the coordinate symbols; for instance, one may write:

```python
sage: M = Manifold(3, 'R^3', r'\mathbb{R}^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 \(y1\):

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

```python
sage: U = M.open_subset('U')
sage: c_spher.<r,th,ph> = U.chart(\(r:(0,+\infty)\ th:(0,\pi)\ ph:(0,2\pi)\))
```

Note the prefix ‘r’ for the string defining the coordinates in the arguments of `chart`.

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

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

Coordinate are also accessible by their indices:

```python
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 `[]`:

```python
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:

```python
sage: c_cart.coord_range()
x: (-\infty, +\infty); y: (-\infty, +\infty); z: (-\infty, +\infty)
sage: c_spher.coord_range()
r: (0, +\infty); 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:

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

A coordinate can be declared periodic by adding the keyword `periodic` to its range:
sage: V = M.open_subset('V')
sage: c_spher1.<r,th,ph1> = V.chart(rr:(0,+oo) th:(0,pi): \theta ph1:(0,2*pi):periodic: \phi_1)
sage: c_spher1.periods()
(\text{None}, \text{None}, 2\pi)
sage: c_spher1.coord_range()
r: (0, +\infty); th: (0, \pi); ph1: \{0, 2\pi\} (periodic)

It is equivalent to give the period as \text{period}=2\pi, skipping the coordinate range:

sage: c_spher2.<r,th,ph2> = V.chart(rr:(0,+oo) th:(0,pi): \theta ph2:period=2*pi: \phi_2)
sage: c_spher2.periods()
(\text{None}, \text{None}, 2\pi)
sage: c_spher2.coord_range()
r: (0, +\infty); th: (0, \pi); ph2: \{0, 2\pi\} (periodic)

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)), Chart (V, (r, th, ph1)), Chart (V, (r, th, ph2))]

and to the atlas of its domain:

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

Manifold subsets have a \textit{default chart}, which, unless changed via the method \texttt{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 \texttt{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 \(\mathbb{R}^3\)
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...
\rightarrow \text{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...
\rightarrow \text{ones}
sage: c_spher(a)
(1, 1/2\pi, \pi)
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:

```python
sage: c_cartU.<x,y,z> = U.chart(coord_restrictions=lambda x,y,z: (y!=0, x<0))
   # the tuple (y!=0, x<0) means y!=0 OR x<0
   # [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))]
```

A vector frame is naturally associated to each chart:

```python
sage: c_cart.frame()
Coordinate frame (R^3, (∂/∂x,∂/∂y,∂/∂z))
sage: c_spher.frame()
Coordinate frame (U, (∂/∂r,∂/∂th,∂/∂ph))
```

as well as a dual frame (basis of 1-forms):

```python
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 != 0) or (y != 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, ψ)

EXAMPLES:

Cartesian coordinates on the unit open disc in \( \mathbb{R}^2 \) as a subchart of the global Cartesian coordinates:

```python
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 \( 1 < \sqrt{x^2+y^2} < 2 \):

```python
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 \( \mathbb{R} \). The derived class RealLine is devoted to \( \mathbb{R} \) itself, as the open interval \((-\infty,+\infty)\).

AUTHORS:

• Eric Gourgoulhon (2015): initial version

• Travis Scrimshaw (2016): review tweaks

REFERENCES:

• [Lee2013]
class sage.manifolds.differentiable.examples.real_line.OpenInterval(lower, upper, ambient_interval=None, name=None, latex_name=None, coordinate=None, names=None, start_index=0)

Bases: DifferentiableManifold

Open interval as a 1-dimensional differentiable manifold over \( \mathbb{R} \).

INPUT:

- `lower` – lower bound of the interval (possibly \(-\infty\))
- `upper` – upper bound of the interval (possibly \(+\infty\))
- `ambient_interval` – (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 = manifolds.OpenInterval(0, pi); I
Real interval (0, \pi)
sage: latex(I)
\left(0, \pi\right)
```

\(I\) is a 1-dimensional smooth manifold over \( \mathbb{R} \):

```
sage: I.category()
Category of smooth connected 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 manifolds.OpenInterval(0, pi)
True
sage: I is manifolds.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 = manifolds.OpenInterval(0, pi, name='I'); I1
Real interval I
sage: latex(I1)
I
sage: I2 = manifolds.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)
<class '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> = manifolds.OpenInterval(0, pi)
sage: t
<class 'sage.symbolic.expression.Expression'>
```

The trick is performed by the Sage preparser:

```
sage: preparse("I.<t> = manifolds.OpenInterval(0, pi)")
"I = manifolds.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> = manifolds.OpenInterval(0, pi)
sage: J.canonical_chart()
Chart ((0, canonical_chart()
Chart ((0, pi), (x,))
```

(continues on next page)
The LaTeX symbol of the canonical coordinate can be adjusted via the same syntax as a chart declaration (see `RealChart`):

```sage
J.<x> = manifolds.OpenInterval(0, pi, coordinate=r'x:\xi')
latex(x)
\xi
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)
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
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)
x.coord()
(2,)
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
I.upper_bound()
pi
```

One of the endpoint can be infinite:

```sage
J = manifolds.OpenInterval(1, +oo); J
Real interval (1, +Infinity)
J.an_element().coord()
(2,)
```

The construction of a subinterval can be performed via the argument `ambient_interval` of `OpenInterval`:
sage: J = manifolds.OpenInterval(0, 1, ambient_interval=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: list(I.subset_family())
[Real interval (0, 1), Real interval (0, pi), Real interval (1/2, 1)]

sage: list(J.subset_family())
[Real interval (0, 1), Real interval (1/2, 1)]

sage: list(K.subset_family())
[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 and Category of connected
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 and Category of connected
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 connected manifolds over Real Field with 53 bits of
precision

and we have:

sage: J.category() is I.category().Subobjects()
True

All intervals are parents:

2.3. The Real Line and Open Intervals
We have:

We have:

The canonical chart of subintervals is inherited from the canonical chart of the parent interval:

The canonical chart of subintervals is inherited from the canonical chart of the parent interval:

canonical_chart()

Return the canonical chart defined on self.

OUTPUT:

\cdot \text{RealDiffChart}

EXAMPLES:

Canonical chart on the interval \((0, \pi)\):

Canonical chart on the interval \((0, \pi)\):

The symbol used for the coordinate of the canonical chart is that defined during the construction of the interval:

The symbol used for the coordinate of the canonical chart is that defined during the construction of the interval:

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 = manifolds.OpenInterval(0, pi)
sage: I.canonical_coordinate()
t
sage: type(I.canonical_coordinate())
<class '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 = manifolds.OpenInterval(0, pi, coordinate='x')
sage: I.canonical_coordinate()
x
sage: I.<x> = manifolds.OpenInterval(0, pi)
sage: I.canonical_coordinate()
x
```

`inf()`

Return the lower bound (infimum) of the interval.

EXAMPLES:

```
sage: I = manifolds.OpenInterval(1/4, 3)
sage: I.lower_bound()
1/4
sage: J = manifolds.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 = manifolds.OpenInterval(1/4, 3)
sage: I.lower_bound()
1/4
sage: J = manifolds.OpenInterval(-oo, 2)
sage: J.lower_bound()
-Infinity
```
An alias of \texttt{lower_bound()} is \texttt{inf()}:

\begin{verbatim}
sage: I.inf()
1/4
sage: J.inf()
-Infinity
\end{verbatim}

\texttt{open_interval (lower, upper, name=None, latex_name=None)}

Define an open subinterval of \texttt{self}.

\textbf{INPUT}:

\begin{itemize}
  \item \texttt{lower} – lower bound of the subinterval (possibly \texttt{-Infinity})
  \item \texttt{upper} – upper bound of the subinterval (possibly \texttt{+Infinity})
  \item \texttt{name} – (default: \texttt{None}) string; name (symbol) given to the subinterval; if \texttt{None}, the name is constructed from \texttt{lower} and \texttt{upper}
  \item \texttt{latex_name} – (default: \texttt{None}) string; \LaTeX{} symbol to denote the subinterval; if \texttt{None}, the \LaTeX{} symbol is constructed from \texttt{lower} and \texttt{upper} if \texttt{name} is \texttt{None}, otherwise, it is set to \texttt{name}
\end{itemize}

\textbf{OUTPUT}:

\begin{itemize}
  \item \texttt{OpenInterval} representing the open interval (\texttt{lower}, \texttt{upper})
\end{itemize}

\textbf{EXAMPLES}:

The interval \((0, \pi)\) as a subinterval of \((-4, 4)\):

\begin{verbatim}
sage: I = manifolds.OpenInterval(-4, 4)
sage: J = I.open_interval(0, pi); J
Real interval (0, pi)
sage: J.is_subset(I)
True
sage: list(I.subset_family())
[Real interval (-4, 4), Real interval (0, pi)]
\end{verbatim}

\texttt{J} is considered as an open submanifold of \texttt{I}:

\begin{verbatim}
sage: J.manifold() is I
True
\end{verbatim}

The subinterval \((-4, 4)\) is \texttt{I} itself:

\begin{verbatim}
sage: I.open_interval(-4, 4) is I
True
\end{verbatim}

\texttt{sup()} 

Return the upper bound (supremum) of the interval.

\textbf{EXAMPLES}:

\begin{verbatim}
sage: I = manifolds.OpenInterval(1/4, 3)
sage: I.upper_bound()
3
sage: J = manifolds.OpenInterval(1, +oo)
sage: J.upper_bound() +Infinity
\end{verbatim}

An alias of \texttt{upper_bound()} is \texttt{sup()}:
upper_bound()

Return the upper bound (supremum) of the interval.

EXAMPLES:

```python
sage: I = manifolds.OpenInterval(1/4, 3)
sage: I.upper_bound()
3
sage: J = manifolds.OpenInterval(1, +oo)
sage: J.upper_bound()
+Infinity
```

An alias of `upper_bound()` is `sup()`:

```python
sage: I.sup()
3
sage: J.sup()
+Infinity
```

class `sage.manifolds.differentiable.examples.real_line.RealLine`

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:

```python
sage: R = manifolds.RealLine(); R
Real number line R
sage: latex(R)
\Bold{R}
```

R is a 1-dimensional real smooth manifold:
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 manifolds.RealLine()
True
sage: R is manifolds.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)
<class '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> = manifolds.RealLine()
sage: t
t
sage: type(t)
<class 'sage.symbolic.expression.Expression'>
```

The trick is performed by Sage preparser:

```
sage: preparse("R.<t> = manifolds.RealLine()")
"R = manifolds.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> = manifolds.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`):
The LaTeX symbol of the real line itself can also be customized:

```sage
R.<x> = manifolds.RealLine(latex_name=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 \( \mathbb{R} \)
```

```sage
p.coord()
(2,)
```

```sage
p = R(1.742) ; p
Point on the Real number line \( \mathbb{R} \)
```

```sage
p.coord()
(1.74200000000000,)
```

Symbolic variables can also be used:

```sage
p = R(pi, name =pi) ; p
Point \( \pi \) on the Real number line \( \mathbb{R} \)
```

```sage
p.coord()
(pi,)
```

```sage
a = var('a')
```

```sage
p = R(a) ; p
Point on the Real number line \( \mathbb{R} \)
```

```sage
p.coord()
(a,)
```

The real line is considered as the open interval \((-\infty, +\infty)\):

```sage
isinstance(R, sage.manifolds.differentiable.examples.real_line.OpenInterval)
True
```

```sage
R.lower_bound()
-Infinity
```

```sage
R.upper_bound()
+Infinity
```

A real interval can be created from \( \mathbb{R} \) means of the method `open_interval()`:

```sage
I = R.open_interval(0, 1); I
Real interval (0, 1)
```

```sage
I.manifold()
Real number line \( \mathbb{R} \)
```

```sage
list(R.subset_family())
[Real interval (0, 1), Real number line \( \mathbb{R} \)]
```
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, $K = \mathbb{R}$ or $K = \mathbb{C}$). By differentiable scalar field, it is meant a function $M \to K$ 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:


REFERENCES:

- [KN1963]
- [Lee2013]
- [ONe1983]

class sage.manifolds.differentiable.scalarfield_algebra.DiffScalarFieldAlgebra (domain)

Bases: `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 $M \to K$. 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 $K = \mathbb{R}$ or $K = \mathbb{C}$, 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 $\mathbb{R}$ nor $\mathbb{C}$ 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:

```python
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: 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()
```

(continues on next page)
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 \( \mathbb{R} \) (represented here by Sage's Symbolic Ring):

```python
sage: CM.category()
Join of Category of commutative algebras over Symbolic Ring and Category of \( \rightarrow \) homsets of topological spaces
sage: CM.base_ring()
Symbolic Ring
sage: CW.category()
Join of Category of commutative algebras over Symbolic Ring and Category of \( \rightarrow \) homsets of topological spaces
sage: CW.base_ring()
Symbolic Ring
```

The elements of \( C^k(M) \) are scalar fields on \( M \):

```python
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 → \( \mathbb{R} \)
on U: \((x, y) \mapsto 2
on V: \((u, v) \mapsto 2
```

Those of \( C^k(W) \) are scalar fields on \( W \):

```python
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 → \( \mathbb{R} \)
(x, y) \mapsto 2
(u, v) \mapsto 2
```

The zero element:

```python
sage: CM.zero()
Scalar field zero on the 2-dimensional differentiable manifold M
sage: CM.zero().display()
zero: M → \( \mathbb{R} \)
on U: \((x, y) \mapsto 0
on V: \((u, v) \mapsto 0
```

```python
sage: CW.zero()
Scalar field zero on the Open subset W of the 2-dimensional differentiable manifold M
sage: CW.zero().display()
zero: W → \( \mathbb{R} \)
(x, y) \mapsto 0
(u, v) \mapsto 0
```

The unit element:
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()
$W \rightarrow \mathbb{R}$
$(x, y) \mapsto \arctan(x^2 + y^2)$
$(u, v) \mapsto 1/2\pi - \arctan(u^2 + v^2)$
```

```
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()
$W \rightarrow \mathbb{R}$
$(x, y) \mapsto 2\arctan(x^2 + y^2)$
$(u, v) \mapsto \pi - 2\arctan(u^2 + v^2)$
```

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._rmul_()`. 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()
$M \rightarrow \mathbb{R}$
on $U$: $(x, y) \mapsto \sqrt{2}\pi$
on $V$: $(u, v) \mapsto \sqrt{2}\pi$
```

```
sage: a = var(a)
sage: h = CM(a); h .display()
$M \rightarrow \mathbb{R}$
on $U$: $(x, y) \mapsto a$
on $V$: $(u, v) \mapsto a$
```

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()
$M \rightarrow \mathbb{R}$
on $U$: $(x, y) \mapsto a + x$
on $W$: $(u, v) \mapsto (a*u^2 + a*v^2 + u)/(u^2 + v^2)$
```

```
sage: h = CM(a+u); h.display()
$M \rightarrow \mathbb{R}$
```

(continues on next page)
on W: \((x, y) \mapsto (a*x^2 + a*y^2 + x)/(x^2 + y^2)\)
on V: \((u, v) \mapsto a + u\)

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:

\begin{verbatim}
sage: h = CM(x+u); h.display()
M → ℝ
\end{verbatim}

**TESTS OF THE ALGEBRA LAWS:**

**Ring laws:**

\begin{verbatim}
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: \((x, y) \mapsto \sqrt(2)\pi + \arctan(x^2 + y^2)\)
on V: \((u, v) \mapsto 1/2\pi*(2*\sqrt(2) + 1) - \arctan(u^2 + v^2)\)
\end{verbatim}

\begin{verbatim}
sage: s = f - h ; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M → ℝ
on U: \((x, y) \mapsto -\sqrt(2)\pi + \arctan(x^2 + y^2)\)
on V: \((u, v) \mapsto -1/2\pi*(2*\sqrt(2) - 1) - \arctan(u^2 + v^2)\)
\end{verbatim}

\begin{verbatim}
sage: s = f*h ; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M → ℝ
on U: \((x, y) \mapsto \sqrt(2)\pi*\arctan(x^2 + y^2)\)
on V: \((u, v) \mapsto 1/2*\sqrt(2)*(\pi^2 - 2*\pi*\arctan(u^2 + v^2))\)
\end{verbatim}

\begin{verbatim}
sage: s = f/h ; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M → ℝ
on U: \((x, y) \mapsto 1/2*\sqrt(2)*\pi*\arctan(x^2 + y^2)/\pi\)
on V: \((u, v) \mapsto 1/4*\sqrt(2)*(\pi - 2*\arctan(u^2 + v^2))/\pi\)
\end{verbatim}

\begin{verbatim}
sage: f*(h+f) == f*h + f*f
True
\end{verbatim}

**Ring laws with coercion:**

\begin{verbatim}
sage: f - f_W == CW.zero()
True
sage: f/f_W == CW.one()
True
sage: s = f*f_W ; s
Scalar field on the Open subset \(W\) of the 2-dimensional differentiable
Multiplication by a number:

\[
sage: \ s = 2*f \ ; \ s
\]

Scalar field on the 2-dimensional differentiable manifold \( M \)
\[
sage: \ s.display()
\]

\[
M \rightarrow \mathbb{R}
\)
on \( U \): \((x, y) \mapsto 2*\arctan(x^2 + y^2) \)
on \( V \): \((u, v) \mapsto \pi - 2*\arctan(u^2 + v^2) \)

\[
sage: \ 0*f == CM.zero()
\]

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 \texttt{DiffScalarField}

### 2.4.2 Differentiable Scalar Fields

Given a differentiable manifold \( M \) of class \( C^k \) over a topological field \( K \) (in most applications, \( K = \mathbb{R} \) or \( K = \mathbb{C} \)), a \textit{differentiable scalar field} on \( M \) is a map

\[
f : M \rightarrow K
\]

of class \( C^k \).

Differentiable scalar fields are implemented by the class \texttt{DiffScalarField}.

AUTHORS:

- Eric Gourgoulhon (2018): operators gradient, Laplacian and d’Alembertian

REFERENCES:
class sage.manifolds.differentiable.scalarfield.DiffScalarField(parent, coord_expression=None, chart=None, name=None, latex_name=None)

Bases: 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, $K = \mathbb{R}$ or $K = \mathbb{C}$), a *differentiable scalar field* defined on $M$ is a map

$$f : M \rightarrow K$$

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
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)),
(continues on next page)
```
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 ∩ 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:
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))
<class 'sage.symbolic.expression.Expression'>
```

The method `coord_function()` returns instead a function of the chart coordinates, i.e. an instance of `ChartFunction`:

```
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.chart_func.ChartFunctionRing_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:
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
```

(continues on next page)
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. $C^k(U)$):

```python
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:

```python
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:

```python
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:

```python
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:

```python
sage: s = f + 1 ; s
Scalar field f+1 on the 2-dimensional differentiable manifold M
sage: s.display()
f+1: 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)
```

The number can 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)
on W: (u, v) ↦ (u^4 + v^4 + u^3 + (2*u^2 + u)*v^2 + u)/(u^4 + v^4 + (2*u^2 + 1)*v^2 + u)

sage: s = f + u; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M → ℝ
on U: (x, y) ↦ (x^3 + (x + 1)*y^2 + x^2 + x)/(x^4 + y^4 + (2*x^2 + 1)*y^2 + x^2)
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 f+g 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()
f+g: 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 :class:`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 $\mathbb{R}$, 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)
on W: (u, v) ↦ u/(u^2 + v^2 + 1)
sage: s = u*f; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M → ℝ
on W: (x, y) ↦ x/(x^4 + y^4 + (2*x^2 + 1)*y^2 + x^2)
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()
```

(continues on next page)
\( g \ast h : U \to \mathbb{R} \)
\((x, y) \mapsto x^y \ast H(x, y) \)
on \( W : (u, v) \mapsto u^y \ast H(u/(u^2 + v^2), v/(u^2 + v^2))/(u^4 + 2*u^2*v^2 + v^4) \)

Thanks to the coercion \( C^k(M) \to C^k(U) \) 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 \( U \):

\[
\text{sage: } f \cdot \text{domain()}, g \cdot \text{domain()}
\]
(2-dimensional differentiable manifold \( M \),
Open subset \( U \) of the 2-dimensional differentiable manifold \( M \))
\[
\text{sage: } s = f \ast g ; s
\]
Scalar field \( f \ast g \) on the Open subset \( U \) of the 2-dimensional differentiable manifold \( M \)
\[
\text{sage: } s \cdot \text{display()}
\]
f\( \ast g : U \to \mathbb{R} \)
\((x, y) \mapsto x^y/(x^2 + y^2 + 1) \)
on \( W : (u, v) \mapsto u^v/(u^2 + v^2 + (2*u^2 + 1)*v^2 + u^2) \)
\[
\text{sage: } s == f \cdot \text{restrict}(U) \ast g
\]
True

Scalar fields can be divided (pointwise division):

\[
\text{sage: } s = f/c ; s
\]
Scalar field \( f/c \) on the 2-dimensional differentiable manifold \( M \)
\[
\text{sage: } s \cdot \text{display()}
\]
f\( / c : M \to \mathbb{R} \)
on \( U : (x, y) \mapsto 1/(a*x^2 + a*y^2 + a) \)
on \( V : (u, v) \mapsto (u^2 + v^2)/(a*u^2 + a*v^2 + a) \)
\[
\text{sage: } s \cdot g \cdot \text{display()}
\]
g\( / h : U \to \mathbb{R} \)
\((x, y) \mapsto x^y / H(x, y) \)
on \( W : (u, v) \mapsto u^v/((u^4 + 2*u^2*v^2 + v^4)*H(u/(u^2 + v^2), v/(u^2 + v^2))) \)
\[
\text{sage: } s == f/g
\]
True

For scalar fields defined on a single chart domain, we may perform some arithmetics with symbolic expressions involving the chart coordinates:

\[
\text{sage: } s = g + x^2 - y ; s
\]
Scalar field on the Open subset \( U \) of the 2-dimensional differentiable manifold \( M \)
\[
\text{sage: } s \cdot \text{display()}
\]
\( U \to \mathbb{R} \)
\((x, y) \mapsto x^2 + (x - 1)*y \)
on \( W : (u, v) \mapsto -(v^3 - u^2 + (u^2 - u)*v)/(u^4 + 2*u^2*v^2 + v^4) \)
The test suite is passed:

```sage```
TestSuite(f).run()
TestSuite(zer).run()
```

**bracket**(other)

Return the Schouten-Nijenhuis bracket of self, considered as a multivector field of degree 0, with a multivector field.

See **bracket()** for details.

**INPUT:**

- other – a multivector field of degree \( p \)

**OUTPUT:**

- if \( p = 0 \), a zero scalar field
- if \( p = 1 \), an instance of **DiffScalarField** representing the Schouten-Nijenhuis bracket [self, other]
- if \( p \geq 2 \), an instance of **MultivectorField** representing the Schouten-Nijenhuis bracket [self, other]

**EXAMPLES:**

The Schouten-Nijenhuis bracket of two scalar fields is identically zero:

```sage```
M = Manifold(2, 'M')
x, y = M.chart()
f = M.scalar_field({x: x+y^2}, name='f')
g = M.scalar_field({y: y-x}, name='g')
s = f.bracket(g); s Scalar field zero on the 2-dimensional differentiable manifold M
```

(continues on next page)
while the Schouten-Nijenhuis bracket of a scalar field \( f \) with a multivector field \( a \) is equal to minus the interior product of the differential of \( f \) with \( a \):

\[
sage: a = M.multivector_field(2, name =a) \\
sage: a[0,1] = x*y ; a.display() \\
a = x*y \partial/\partial x \wedge \partial/\partial y \\
sage: s = f.bracket(a); s \\
Vector field -i_df a on the 2-dimensional differentiable manifold M \\
sage: s.display() \\
-i_df a = 2*x*y^2 \partial/\partial x - x*y \partial/\partial y
\]

See \texttt{bracket()} for other examples.

\textbf{dalembertian (metric=None)}

Return the d'Alembertian of \texttt{self} with respect to a given Lorentzian metric.

The \textit{d'Alembertian} of a scalar field \( f \) with respect to a Lorentzian metric \( g \) is nothing but the Laplacian (see \texttt{laplacian()}) of \( f \) with respect to that metric:

\[
\Box f = g^{ij} \nabla_i \nabla_j f = \nabla_i \nabla^i f
\]

where \( \nabla \) is the Levi-Civita connection of \( g \).

**Note:** If the metric \( g \) is not Lorentzian, the name \textit{d'Alembertian} is not appropriate and one should use \texttt{laplacian()} instead.

**INPUT:**

- \texttt{metric} – (default: None) the Lorentzian metric \( g \) involved in the definition of the d'Alembertian: if none is provided, the domain of \texttt{self} is supposed to be endowed with a default Lorentzian metric (i.e. is supposed to be Lorentzian manifold, see \textit{PseudoRiemannianManifold}) and the latter is used to define the d'Alembertian

**OUTPUT:**

- instance of \textit{DiffScalarField} representing the d'Alembertian of \texttt{self}

**EXAMPLES:**

d'Alembertian of a scalar field in Minkowski spacetime:

\[
sage: M = Manifold(4, 'M', structure='Lorentzian') \\
sage: X.<t,x,y,z> = M.chart() \\
sage: g = M.metric() \\
sage: g[0,0], g[1,1], g[2,2], g[3,3] = -1, 1, 1, 1 \\
sage: f = M.scalar_field(t + x^2 + t^2*y^3 - x*z^4, name='f') \\
sage: s = f.dalembertian(); s \\
Scalar field Box(f) on the 4-dimensional Lorentzian manifold M \\
sage: s.display() \\
Box(f): M \to \mathbb{R} \\
\( (t, x, y, z) \mapsto 6*t^2*y - 2*y^3 - 12*x*z^2 + 2 \)
\]

The function \texttt{dalembertian()} from the \textit{operators} module can be used instead of the method \texttt{dalembertian()}.  

\textbf{See also:} \texttt{bracket()}, \texttt{laplacian()}. 

430 Chapter 2. Differentiable Manifolds
Manifolds, Release 10.3

```python
sage: from sage.manifolds.operators import dalembertian
dalembertian(f) == s
```

**degree()**

Return the degree of `self`, considered as a differential form or a multivector field, i.e. zero.

This trivial method is provided for consistency with the exterior calculus scheme, cf. the methods `degree()` (differential forms) and `degree()` (multivector fields).

**OUTPUT:**

- 0

**EXAMPLES:**

```python
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x+y^2})
sage: f.degree()
0
```

**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:

```python
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
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 Omega^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:

```python
sage: f.differential() is df
True
```

Instead of invoking the method `differential()`, one may apply the function `diff` to the scalar field:

```python
sage: diff(f) is f.differential()
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()`:

2.4. Scalar Fields 431
```
sage: df = f.exterior_derivative() ; df
definition 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
```

**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
definition on the 3-dimensional differentiable manifold M
sage: dg._components
{(Coordinate frame (M, (∂/∂u,∂/∂v,∂/∂w)): 1-index components w.r.t.
  Coordinate frame (M, (∂/∂u,∂/∂v,∂/∂w))):

  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)
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
definition on the 3-dimensional differentiable manifold M
sage: ddf == 0
True
sage: ddf[:]
[0 0 0]
[0 0 0]
[0 0 0]
sage: ddg = dg.exterior_derivative() ; ddg
definition on the 3-dimensional differentiable manifold M
sage: ddg == 0
True
```

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
definition 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
```

(continues on next page)
Free module $\Omega^1(M)$ 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
```

Instead of invoking the method `differential()`, one may apply the function `diff` to the scalar field:

```
sage: diff(f) is f.differential()
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 $M$
```

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 $M$
```

The exterior derivative is nilpotent:

```
sage: ddf = df.exterior_derivative() ; ddf
2-form ddf on the 3-dimensional differentiable manifold $M$
```

```
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:**

2.4. Scalar Fields 433
• a :math:`\text{DiffForm}` (or of :math:`\text{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:

```python
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 Omega^1(M) of 1-forms on the 3-dimensional differentiable manifold M
```

The result is cached, i.e. is not recomputed unless :math:`f` is changed:

```python
sage: f.differential() is df
True
```

Instead of invoking the method `differential()`, one may apply the function `diff` to the scalar field:

```python
sage: diff(f) is f.differential()
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()`:

```python
sage: df = f.exterior_derivative() ; df
1-form df on the 3-dimensional differentiable manifold M
sage: df._components
{Coordinate frame (M, (∂/∂u,∂/∂v,∂/∂w)): 1-index components w.r.t. Coordinate frame (M, (∂/∂u,∂/∂v,∂/∂w))}
```

Differential computed on a chart that is not the default one:

```python
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, (∂/∂u,∂/∂v,∂/∂w)): 1-index components w.r.t. Coordinate frame (M, (∂/∂u,∂/∂v,∂/∂w))}
sage: dg.comp(c_uvw.frame())[v^2*w^3, 2*u*v*w^3, 3*u*v^2*w^2]
sage: dg.display(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:

```python
sage: ddf = df.exterior_derivative() ; ddf
2-form ddf on the 3-dimensional differentiable manifold M
```
gradient (metric=None)

Return the gradient of self (with respect to a given metric).

The gradient of a scalar field \( f \) with respect to a metric \( g \) is the vector field \( \nabla f \) whose components in any coordinate frame are

\[
(\nabla f)^i = g^{ij} \frac{\partial F}{\partial x^j}
\]

where the \( x^j \)'s are the coordinates with respect to which the frame is defined and \( F \) is the chart function representing \( f \) in these coordinates: \( f(p) = F(x^1(p), \ldots, x^n(p)) \) for any point \( p \) in the chart domain. In other words, the gradient of \( f \) is the vector field that is the \( g \)-dual of the differential of \( f \).

INPUT:

- metric - (default: None) the pseudo-Riemannian metric \( g \) involved in the definition of the gradient; if none is provided, the domain of self is supposed to be endowed with a default metric (i.e. is supposed to be pseudo-Riemannian manifold, see \texttt{PseudoRiemannianManifold}) and the latter is used to define the gradient

OUTPUT:

- instance of \texttt{VectorField} representing the gradient of \texttt{self}

EXAMPLES:

Gradient of a scalar field in the Euclidean plane:

```python
sage: M.<x,y> = EuclideanSpace()
sage: f = M.scalar_field(cos(x*y), name='f')
sage: v = f.gradient(); v
Vector field grad(f) on the Euclidean plane E^2
sage: v.display()
grad(f) = -y*sin(x*y) e_x - x*sin(x*y) e_y
sage: v[:]
[-y*sin(x*y), -x*sin(x*y)]
```

Gradient in polar coordinates:

```python
sage: M.<r,phi> = EuclideanSpace(coordinates='polar')
sage: f = M.scalar_field(r*cos(phi), name='f')
sage: f.gradient().display()
grad(f) = cos(phi) e_r - sin(phi) e_phi
sage: f.gradient()[:]
[cos(phi), -sin(phi)]
```

Note that \((e_r, e_\phi)\) is the orthonormal vector frame associated with polar coordinates (see \texttt{polar_frame()}); the gradient expressed in the coordinate frame is:
The function `grad()` from the `operators` module can be used instead of the method `gradient()`:

```python
sage: from sage.manifolds.operators import grad
sage: grad(f) == f.gradient()
True
```

The gradient can be taken with respect to a metric tensor that is not the default one:

```python
sage: h = M.lorentzian_metric('h')
sage: h[1,1], h[2,2] = -1, 1/(1+r^2)
sage: h.display(M.polar_coordinates().frame())
h = -dr⊗dr + r^2/(r^2 + 1) dphi⊗dphi
sage: v = f.gradient(h); v
Vector field grad_h(f) on the Euclidean plane E^2
sage: v.display()
grad_h(f) = -cos(phi) e_r + (-r^2*sin(phi) - sin(phi)) e_phi
```

**hodge_dual**(nondegenerate_tensor)

Compute the Hodge dual of the scalar field with respect to some non-degenerate bilinear form (Riemannian metric or symplectic form).

If $M$ is the domain of the scalar field (denoted by $f$), $n$ is the dimension of $M$ and $g$ is a non-degenerate bilinear form on $M$, the Hodge dual of $f$ w.r.t. $g$ is the $n$-form $\ast f$ defined by

$$\ast f = f \epsilon,$$

where $\epsilon$ is the volume $n$-form associated with $g$ (see `volume_form()`).

**INPUT:**

- nondegenerate_tensor: a non-degenerate bilinear form defined on the same manifold as the current differential form; must be an instance of `PseudoRiemannianMetric` or `SymplecticForm`.

**OUTPUT:**

- the $n$-form $\ast f$

**EXAMPLES:**

Hodge dual of a scalar field in the Euclidean space $R^3$:

```python
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:

```python
sage: f.hodge_dual(g) == g.hodge_star(f)
True
```

**`laplacian(metric=None)`**

Return the Laplacian of `self` with respect to a given metric (Laplace-Beltrami operator).

The Laplacian of a scalar field `f` with respect to a metric `g` is the scalar field

\[
\Delta f = g^{ij} \nabla_i \nabla_j f = \nabla_i \nabla^i f
\]

where \(\nabla\) is the Levi-Civita connection of `g`. \(\Delta\) is also called the Laplace-Beltrami operator.

**INPUT:**

- `metric` (default: `None`) the pseudo-Riemannian metric `g` involved in the definition of the Laplacian: if none is provided, the domain of `self` is supposed to be endowed with a default metric (i.e. is supposed to be pseudo-Riemannian manifold, see `PseudoRiemannianManifold`) and the latter is used to define the Laplacian

**OUTPUT:**

- instance of `DiffScalarField` representing the Laplacian of `self`

**EXAMPLES:**

Laplacian of a scalar field on the Euclidean plane:

```python
sage: M.<x,y> = EuclideanSpace()
sage: f = M.scalar_field(function('F')(x,y), name='f')
sage: s = f.laplacian(); s
Scalar field Delta(f) on the Euclidean plane E^2
sage: s.display()
Delta(f): E^2 → ℝ
(x, y) ↦ d^2(F)/dx^2 + d^2(F)/dy^2
```

The function `laplacian()` from the `operators` module can be used instead of the method `laplacian()`:

```python
sage: from sage.manifolds.operators import laplacian
sage: laplacian(f) == s
True
```

The Laplacian can be taken with respect to a metric tensor that is not the default one:

```python
sage: h = M.lorentzian_metric('h')
sage: h[1,1], h[2,2] = -1, 1/(1+x^2+y^2)
sage: s = f.laplacian(h); s
Scalar field Delta_h(f) on the Euclidean plane E^2
sage: s.display()
Delta_h(f): E^2 → ℝ
(x, y) ↦ (y^4*d^2(F)/dy^2 + y^3*d(F)/dy + (2*(x^2 + 1)*d^2(F)/dy^2 - d^2(F)/dx^2)*y^2 + (x^2 + 1)*y*d(F)/dy + x*d(F)/dx - (x^2 + 1)*d^2(F)/dx^2 + (x^4 + 2*x^2 + 1)*d^2(F)/dy^2)/(x^2 + y^2 + 1)
```

The Laplacian of `f` is equal to the divergence of the gradient of `f`:

\[
\Delta f = \text{div(} \text{grad} f \text{)}
\]
Let us check this formula:

```python
sage: s == f.gradient(h).div(h)
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:

```python
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:

```python
sage: f.lie_derivative(v) is f.lie_derivative(v)
True
```

An alias is `lie_der`:

```python
sage: f.lie_der(v) is f.lie_derivative(v)
True
```

Alternative expressions of the Lie derivative of a scalar field:

```python
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:

```python
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:

```python
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:

```python
sage: f.lie_derivative(v) is f.lie_derivative(v)
True
```

An alias is lie_der:

```python
sage: f.lie_der(v) is f.lie_derivative(v)
True
```

Alternative expressions of the Lie derivative of a scalar field:

```python
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:

```python
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:**

```python
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)
```

**wedge**(other)

Return the exterior product of `self`, considered as a differential form of degree 0 or a multivector field of degree 0, with `other`.

See `wedge()` (exterior product of differential forms) or `wedge()` (exterior product of multivector fields) for details.

For a scalar field `f` and a `p`-form (or `p`-vector field) `a`, the exterior product reduces to the standard product on the left by an element of the base ring of the module of `p`-forms (or `p`-vector fields): \( f \wedge a = fa \).

**INPUT:**

- `other` – a differential form or a multivector field `a`

**OUTPUT:**

- the product `fa`, where `f` is `self`

**EXAMPLES:**

```python
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x+y^2}, name='f')
sage: a = M.diff_form(2, name='a')
sage: a[0,1] = x*y
sage: s = f.wedge(a); s
2-form f*a on the 2-dimensional differentiable manifold M
sage: s.display()
f*a = (x*y^3 + x^2*y) dx\wedge dy
```

## 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, \( K = \mathbb{R} \) or \( K = \mathbb{C} \)), 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 \( \mathbb{R} \).

The subclass `IntegratedCurveSet` is devoted to differentiable curves that are defined as a solution to a system of second order differential equations.

The subclass `IntegratedAutoparallelCurveSet` is devoted to differentiable curves that are defined as autoparallel curves with respect to a certain affine connection.

The subclass `IntegratedGeodesicSet` is devoted to differentiable curves that are defined as geodesics with respect to a certain metric.

**AUTHORS:**

440 Chapter 2. Differentiable Manifolds
• Eric Gourgoulhon (2015): initial version
• Travis Scrimshaw (2016): review tweaks
• Karim Van Aelst (2017): sets of integrated curves

REFERENCES:
• [Lee2013]
• [KN1963]

class sage.manifolds.differentiable.manifold_homset.DifferentiableCurveSet(domain, codomain, name=None, latex_name=None)

Bases: DifferentiableManifoldHomset

Set of differentiable curves in a differentiable manifold.

Given an open interval \( I \) of \( \mathbb{R} \) (possibly \( I = \mathbb{R} \)) and a differentiable manifold \( M \) over \( \mathbb{R} \), this is the set \( \text{Hom}(I, M) \) of morphisms (i.e. differentiable curves) \( I \to M \).

INPUT:
• domain – \texttt{OpenInterval} if an open interval \( I \subset \mathbb{R} \) (domain of the morphisms), or \texttt{RealLine} if \( I = \mathbb{R} \)
• codomain – \texttt{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 \( \mathbb{R} \longrightarrow M \), where \( M \) is a 2-dimensional manifold:

```python
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: R.<t> = manifolds.RealLine() ; R
Real number line \( \mathbb{R} \)
sage: H = Hom(R, M) ; H
Set of Morphisms from Real number line \( \mathbb{R} \) 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)
\text{\textbackslash mathrm{Hom}}(\text{\textbackslash Bold{R}},\text{\textbackslash Bold{M}})
```

An element of \( H \) is a curve in \( M \):

```python
sage: c = H.an_element(); c
Curve in the 2-dimensional differentiable manifold \( M \)
sage: c.display()
```

(continues on next page)
The test suite is passed:

\[
\text{sage: } \text{TestSuite}(\mathcal{H}).\text{run()}
\]

The set of curves \((0, 1) \rightarrow U\), where \(U\) is an open subset of \(M\):

\[
\text{sage: } I = R.\text{open_interval}(0, 1) \ ; I
\]
Real interval \((0, 1)\)

\[
\text{sage: } U = M.\text{open_subset}('U', \text{coord_def}=(X: x^2+y^2<1)) \ ; U
\]
Open subset \(U\) of the 2-dimensional differentiable manifold \(M\)

\[
\text{sage: } H = \text{Hom}(I, U) \ ; H
\]
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 \(U\):

\[
\text{sage: } c = H.\text{an_element()} \ ; c
\]
Curve in the Open subset \(U\) of the 2-dimensional differentiable
manifold \(M\)

\[
\text{sage: } c.\text{display()}
\]
\((0, 1) \rightarrow U \quad t \mapsto (x, y) = (1/(t^2 + 1) - 1/2, 0)
\]

The set of curves \(\mathbb{R} \rightarrow \mathbb{R}\) is a set of (manifold) endomorphisms:

\[
\text{sage: } E = \text{Hom}(R, R) \ ; E
\]
Set of Morphisms from Real number line \(\mathbb{R}\) to Real number line \(\mathbb{R}\) in
Category of smooth connected manifolds over Real Field with 53 bits of
precision

\[
\text{sage: } E.\text{category()}
\]
Category of endsets of topological spaces

\[
\text{sage: } E.\text{is_endomorphism_set()}
\]
True

\[
\text{sage: } E \text{ is End}(R)
\]
True

It is a monoid for the law of morphism composition:

\[
\text{sage: } E \text{ in Monoids()}
\]
True

The identity element of the monoid is the identity map of \(\mathbb{R}\):

\[
\text{sage: } E.\text{one()}
\]
Identity map \(\text{Id}_\mathbb{R}\) of the Real number line \(\mathbb{R}\)

\[
\text{sage: } E.\text{one()} \text{ is R.\text{identity_map()}}
\]
True

\[
\text{sage: } E.\text{one()}.\text{display()}
\]
\(\text{Id}_\mathbb{R}: \mathbb{R} \rightarrow \mathbb{R} \quad t \mapsto t\)

A “typical” element of the monoid:
The testsuite 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 and Category of connected
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, latex_name=None)

Bases: `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:

```python
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)

An element of $H$ is a differentiable map from $M$ to $N$:

```python
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()
$M \rightarrow N$
$(x, y) \mapsto (u, v, w) = (0, 0, 0)$
```

The test suite is passed:

```python
sage: TestSuite(H).run()
```

When the codomain coincides with the domain, the homset is a set of *endomorphisms* in the category of differentiable manifolds:

```python
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
  \( (x, y) \mapsto (x, y) \)
```

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`

```python
class sage.manifolds.differentiable.manifold_homset.IntegratedAutoparallelCurveSet (domain, codomain, name=None, latex_name=None)
```

**Bases:** `IntegratedCurveSet`

Set of integrated autoparallel curves in a differentiable manifold.

**INPUT:**

- `domain` – `OpenInterval` open interval \( I \subset \mathbb{R} \) with finite boundaries (domain of the morphisms)
- `codomain` – `DifferentiableManifold`; differentiable manifold \( M \) (codomain of the morphisms)
- `name` – (default: None) string; name given to the set of integrated autoparallel curves; if None, `Hom_autoparallel(I, M)` will be used
- `latex_name` – (default: None) string; LaTeX symbol to denote the set of integrated autoparallel curves; if None, `\text{Hom}_{\text{autoparallel}}(I, M)` will be used

**EXAMPLES:**

This parent class needs to be imported:

```
sage: from sage.manifolds.differentiable.manifold_homset import...
    → IntegratedAutoparallelCurveSet
```

Integrated autoparallel curves are only allowed to be defined on an interval with finite bounds. This forbids to define an instance of this parent class whose domain has infinite bounds:
Manifolds, Release 10.3

An instance whose domain is an interval with finite bounds allows to build a curve that is autoparallel with respect to a connection defined on the codomain:

```
sage: I = R.open_interval(-1, 2)
sage: H = IntegratedAutoparallelCurveSet(I, M) ; H
Set of Morphisms from Real interval (-1, 2) to 2-dimensional
differentiable manifold M in Category of homsets of topological spaces
which actually are integrated autoparallel curves with respect to a
certain affine connection
sage: nab = M.affine_connection('nabla')
sage: nab[0,1,0], nab[0,0,1] = 1,2
sage: nab.torsion()[:]
[[[0, -1], [1, 0]], [[0, 0], [0, 0]]]
sage: t = var('t')
sage: p = M.point((3,4))
sage: Tp = M.tangent_space(p)
sage: v = Tp((1,2))
sage: c = H(nab, t, v, name='c') ; c
Integrated autoparallel curve c in the 2-dimensional
differentiable manifold M
```

A “typical” element of $H$ is an autoparallel curve in $M$:

```
sage: d = H.an_element(); d
Integrated autoparallel curve in the 2-dimensional
differentiable manifold M
sage: sys = d.system(\text{verbose=True})
Autoparallel curve in the 2-dimensional differentiable manifold
M equipped with Affine connection nab on the 2-dimensional
differentiable manifold M, and integrated over the Real
interval $(-1, 2)$ as a solution to the following equations,
written with respect to Chart $(M, (x, y))$:

Initial point: Point on the 2-dimensional differentiable
manifold $M$ with coordinates $[0, -1/2]$ with respect to
Chart $(M, (x, y))$
Initial tangent vector: Tangent vector at Point on the
2-dimensional differentiable manifold $M$ with components
$[-1/6/(e^{-1} - 1), 1/3]$ with respect to Chart $(M, (x, y))$

\[
\begin{align*}
\frac{dx}{dt} & = Dx \\
\frac{dy}{dt} & = Dy \\
\frac{d(Dx)}{dt} & = -Dx* Dy \\
\frac{d(Dy)}{dt} & = 0
\end{align*}
\]

The test suite is passed:

```
sage: TestSuite(H).run()
```
For any open interval \( J \) with finite bounds \((a, b)\), all curves are autoparallel with respect to any connection. Therefore, the set of autoparallel curves \( J \rightarrow J \) is a set of numerical (manifold) endomorphisms that is a monoid for the law of morphism composition:

```sage
[a,b] = var('a b')
J = R.open_interval(a, b)
H = IntegratedAutoparallelCurveSet(J, J); H
Set of Morphisms from Real interval (a, b) to Real interval (a, b) in Category of endsets of subobjects of sets and topological spaces which actually are integrated autoparallel curves with respect to a certain affine connection
```

Although it is a monoid, no identity map is implemented via the `one` method of this class or its subclass devoted to geodesics. This is justified by the lack of relevance of the identity map within the framework of this parent class and its subclass, whose purpose is mainly devoted to numerical issues (therefore, the user is left free to set a numerical version of the identity if needed):

```sage
H.category()
Category of endsets of subobjects of sets and topological spaces
H
in Monoids()
True
```

A “typical” element of the monoid:

```sage
g = H.an_element() ; g
Integrated autoparallel curve in the Real interval (a, b)
sys = g.system(verbose=True)
```

The test suite is passed, tests `_test_one` and `_test_prod` being skipped for reasons mentioned above:

```sage
TestSuite(H).run(skip=['_test_one', '_test_prod'])
```

Element

alias of `IntegratedAutoparallelCurve`

```sage
sage.manifolds.differentiable.manifold_homset.IntegratedCurveSet(domain, codomain, name=None, latex_name=None)
```

2.5. Differentiable Maps and Curves 447
Bases: DifferentiableCurveSet

Set of integrated curves in a differentiable manifold.

INPUT:

- domain – OpenInterval open interval \( I \subseteq \mathbb{R} \) with finite boundaries (domain of the morphisms)
- codomain – DifferentiableManifold; differentiable manifold \( M \) (codomain of the morphisms)
- name – (default: None) string; name given to the set of integrated curves; if None, \( \text{Hom}_{\text{integrated}}(I, M) \) will be used
- latex_name – (default: None) string; LaTeX symbol to denote the set of integrated curves; if None, \( \text{Hom}_{\text{integrated}}(I, M) \) will be used

EXAMPLES:

This parent class needs to be imported:

```
sage: from sage.manifolds.differentiable.manifold_homset import IntegratedCurveSet
```

Integrated curves are only allowed to be defined on an interval with finite bounds. This forbids to define an instance of this parent class whose domain has infinite bounds:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: R.<t> = manifolds.RealLine()
sage: H = IntegratedCurveSet(R, M)
Traceback (most recent call last):
... ValueError: both boundaries of the interval defining the domain
of a Homset of integrated curves need to be finite
```

An instance whose domain is an interval with finite bounds allows to build an integrated curve defined on the interval:

```
sage: I = R.open_interval(-1, 2)
sage: H = IntegratedCurveSet(I, M) ; H
Set of Morphisms from Real interval (-1, 2) to 2-dimensional
differentiable manifold M in Category of homsets of topological spaces
which actually are integrated curves
sage: eqns_rhs = [1,1]
sage: vels = X.symbolic_velocities()
sage: t = var('t')
sage: p = M.point((3,4))
sage: Tp = M.tangent_space(p)
sage: v = Tp((1,2))
sage: c = H(eqns_rhs, vels, t, v, name='c') ; c
Integrated curve c in the 2-dimensional differentiable
manifold M
```

A “typical” element of \( H \) is a curve in \( M \):

```
sage: d = H.an_element(); d
Integrated curve in the 2-dimensional differentiable manifold M
sage: sys = d.system(\text{verbose}=True)
Curve in the 2-dimensional differentiable manifold M integrated
over the Real interval (-1, 2) as a solution to the following
```

(continues on next page)
Initial point: Point on the 2-dimensional differentiable manifold M with coordinates [0, 0] with respect to Chart (M, (x, y))
Initial tangent vector: Tangent vector at Point on the 2-dimensional differentiable manifold M with components [1/4, 0] with respect to Chart (M, (x, y))

d(x)/dt = Dx
d(y)/dt = Dy
d(Dx)/dt = -1/4*sin(t + 1)
d(Dy)/dt = 0

The test suite is passed:

```
sage: TestSuite(H).run()
```

More generally, an instance of this class may be defined with abstract bounds (a, b):

```
sage: [a,b] = var('a b')
sage: J = R.open_interval(a, b)
sage: H = IntegratedCurveSet(J, M) ; H
Set of Morphisms from Real interval (a, b) to 2-dimensional differentiable manifold M in Category of homsets of topological spaces which actually are integrated curves
```

A “typical” element of $H$ is a curve in $M$:

```
sage: f = H.an_element(); f
Integrated curve in the 2-dimensional differentiable manifold M
sage: sys = f.system(verbose=True)
Curve in the 2-dimensional differentiable manifold M integrated over the Real interval (a, b) as a solution to the following system, written with respect to Chart (M, (x, y)):

Initial point: Point on the 2-dimensional differentiable manifold M with coordinates [0, 0] with respect to Chart (M, (x, y))
Initial tangent vector: Tangent vector at Point on the 2-dimensional differentiable manifold M with components [1/4, 0] with respect to Chart (M, (x, y))

d(x)/dt = Dx
d(y)/dt = Dy
d(Dx)/dt = -1/4*sin(-a + t)
d(Dy)/dt = 0
```

Yet, even in the case of abstract bounds, considering any of them to be infinite is still prohibited since no numerical integration could be performed:

```
sage: f.solve(parameters_values={a:-1, b:+oo})
Traceback (most recent call last):
... ValueError: both boundaries of the interval need to be finite
```

The set of integrated curves $J \longrightarrow J$ is a set of numerical (manifold) endomorphisms:
It is a monoid for the law of morphism composition:

```
sage: H in Monoids()
True
```

Although it is a monoid, no identity map is implemented via the `one` method of this class or any of its subclasses. This is justified by the lack of relevance of the identity map within the framework of this parent class and its subclasses, whose purpose is mainly devoted to numerical issues (therefore, the user is left free to set a numerical version of the identity if needed):

```
sage: H.one()
Traceback (most recent call last):
  ...
ValueError: the identity is not implemented for integrated curves and associated subclasses
```

A “typical” element of the monoid:

```
sage: g = H.an_element() ; g
Integrated curve in the Real interval (a, b)
sage: sys = g.system(\text{verbose=}True)
Curve in the Real interval (a, b) integrated over the Real interval (a, b) as a solution to the following system, written with respect to Chart ((a, b), (t,)):

Initial point: Point on the Real number line \(\mathbb{R}\) with coordinates \([0]\) with respect to Chart ((a, b), (t,))
Initial tangent vector: Tangent vector at Point on the Real number line \(\mathbb{R}\) with components \([1/4]\) with respect to Chart ((a, b), (t,))

\[
d(t)/ds = Dt \\
d(Dt)/ds = -1/4*\sin(-a + s)
\]
```

The test suite is passed, tests `_test_one` and `_test_prod` being skipped for reasons mentioned above:

```
sage: TestSuite(H).run(skip=['_test_one', '_test_prod'])
```

**Element**

- alias of `IntegratedCurve`

- `one()`
  - Raise an error refusing to provide the identity element. This overrides the `one` method of class `TopologicalManifoldHomset`, which would actually raise an error as well, due to lack of option `is_identity` in `element_constructor` method of self.
class sage.manifolds.differentiable.manifold_homset.IntegratedGeodesicSet(domain, codomain, name=None, latex_name=None)

Bases: IntegratedAutoparallelCurveSet

Set of integrated geodesics in a differentiable manifold.

INPUT:

- **domain** – OpenInterval open interval \( I \subset \mathbb{R} \) with finite boundaries (domain of the morphisms)
- **codomain** – DifferentiableManifold; differentiable manifold \( M \) (codomain of the morphisms)
- **name** – (default: None) string; name given to the set of integrated geodesics; if None, \( \text{Hom}_{\text{geodesic}}(I, M) \) will be used
- **latex_name** – (default: None) string; LaTeX symbol to denote the set of integrated geodesics; if None, \( \text{Hom}_{\text{geodesic}}(I, M) \) will be used

EXAMPLES:

This parent class needs to be imported:

```
sage: from sage.manifolds.differentiable.manifold_homset import...
----IntegratedGeodesicSet
```

Integrated geodesics are only allowed to be defined on an interval with finite bounds. This forbids to define an instance of this parent class whose domain has infinite bounds:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: R.<t> = manifolds.RealLine()
sage: H = IntegratedGeodesicSet(R, M)
Traceback (most recent call last):
  ... ValueError: both boundaries of the interval defining the domain of a Homset of integrated geodesics need to be finite
```

An instance whose domain is an interval with finite bounds allows to build a geodesic with respect to a metric defined on the codomain:

```
sage: I = R.open_interval(-1, 2)
sage: H = IntegratedGeodesicSet(I, M) ; H
Set of Morphisms from Real interval (-1, 2) to 2-dimensional differentiable manifold M in Category of homsets of topological spaces which actually are integrated geodesics with respect to a certain metric
sage: g = M.metric('g')
sage: g[0,0], g[1,1], g[0,1] = 1, 1, 2
sage: t = var('t')
sage: p = M.point((3,4))
sage: Tp = M.tangent_space(p)
sage: v = Tp((1,2))
sage: c = H(g, t, v, name='c') ; c
Integrated geodesic c in the 2-dimensional differentiable manifold M
```

2.5. Differentiable Maps and Curves

A “typical” element of $H$ is a geodesic in $M$:

```
sage: d = H.an_element(); d
Integrated geodesic in the 2-dimensional differentiable manifold $M$
sage: sys = d.system( verbose=True )
Geodesic in the 2-dimensional differentiable manifold $M$ equipped with Riemannian metric $g$ on the 2-dimensional differentiable manifold $M$, and integrated over the Real interval $(-1, 2)$ as a solution to the following geodesic equations, written with respect to Chart $(M, (x, y))$:

Initial point: Point on the 2-dimensional differentiable manifold $M$ with coordinates $[0, 0]$ with respect to Chart $(M, (x, y))$
Initial tangent vector: Tangent vector at Point on the 2-dimensional differentiable manifold $M$ with components $[1/3*e^{(1/2)} - 1/3, 0]$ with respect to Chart $(M, (x, y))$

\[
\begin{align*}
\frac{d(x)}{dt} &= Dx \\
\frac{d(y)}{dt} &= Dy \\
\frac{d(Dx)}{dt} &= -Dx^2 \\
\frac{d(Dy)}{dt} &= 0
\end{align*}
\]

The test suite is passed:

```
sage: TestSuite(H).run()
```

For any open interval $J$ with finite bounds $(a, b)$, all curves are geodesics with respect to any metric. Therefore, the set of geodesics $J \rightarrow J$ is a set of numerical (manifold) endomorphisms that is a monoid for the law of morphism composition:

```
sage: [a,b] = var('a b')
sage: J = R.open_interval(a, b)
sage: H = IntegratedGeodesicSet( J, J ); H
Set of Morphisms from Real interval $(a, b)$ to Real interval $(a, b)$ in Category of endsets of subobjects of sets and topological spaces which actually are integrated geodesics with respect to a certain metric
sage: H.category()
Category of endsets of subobjects of sets and topological spaces
sage: H in Monoids()
True
```

Although it is a monoid, no identity map is implemented via the `one` method of this class. This is justified by the lack of relevance of the identity map within the framework of this parent class, whose purpose is mainly devoted to numerical issues (therefore, the user is left free to set a numerical version of the identity if needed):

```
sage: H.one()
Traceback (most recent call last):
...
ValueError: the identity is not implemented for integrated curves and associated subclasses
```

A “typical” element of the monoid:

```
sage: g = H.an_element(); g
Integrated geodesic in the Real interval $(a, b)$
```

(continues on next page)
sage: sys = g.system(\texttt{verbose=True})

Geodesic in the Real interval \((a, b)\) equipped with Riemannian metric \(g\) on the Real interval \((a, b)\), and integrated over the Real interval \((a, b)\) as a solution to the following geodesic equations, written with respect to Chart \(((a, b), (t,))\):

Initial point: Point on the Real number line \(\mathbb{R}\) with coordinates \([0]\) with respect to Chart \(((a, b), (t,))\)
Initial tangent vector: Tangent vector at Point on the Real number line \(\mathbb{R}\) with components \([-e^{(1/2)} - 1]/(a - b)\]
with respect to Chart \(((a, b), (t,))\)

\[
\begin{align*}
    d(t)/ds &= Dt \\
    d(Dt)/ds &= -Dt^2
\end{align*}
\]

The test suite is passed, tests \_test\_one and \_test\_prod being skipped for reasons mentioned above:

```
sage: TestSuite(H).run(skip=["_test_one", "_test_prods"])
```

Element

alias of \texttt{IntegratedGeodesic}

### 2.5.2 Differentiable Maps between Differentiable Manifolds

The class \texttt{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, \(K = \mathbb{R}\) or \(K = \mathbb{C}\)):

\[
\Phi : M \longrightarrow N
\]

AUTHORS:

- Marco Mancini (2018): pullback parallelization

REFERENCES:

- Chap. 1 of [KN1963]
- Chaps. 2 and 3 of [Lee2013]

**class** \texttt{sage.manifolds.differentiable.diff_map.DifMap}\(\texttt{(parent, coord\_functions=\texttt{None}, name=\texttt{None}, latex\_name=\texttt{None}, is\_isomorphism=\texttt{False}, is\_identity=\texttt{False})}\)

**Bases:** \texttt{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, \(K = \mathbb{R}\) or \(K = \mathbb{C}\)).

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 an 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 $\mathbb{R}^3$:

```python
sage: M = Manifold(2, 'S^2') # the 2-dimensional sphere S^2
gsage: U = M.open_subset('U') # complement of the North pole
gsage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
gsage: V = M.open_subset('V') # complement of the South pole
gsage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
gsage: M.declare_union(U,V) # S^2 is the union of U and V
gsage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
gsage: ....: intersection_name='W', restrictions1 = x^2+y^2!=0,
gsage: ....: restrictions2= u^2+v^2!=0)
gsage: uv_to_xy = xy_to_uv.inverse()
gsage: 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
Diffe
tiable 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
```
It is possible to create the map via the method `diff_map()` only in a single pair of charts: the argument `co-ord_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:

```python
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:

```python
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:

```python
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()`:

```python
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:

```python
sage: Phi1 == Phi
True
```

The test suite is passed:

```python
sage: TestSuite(Phi).run()
sage: TestSuite(Phi1).run()
```

The map acts on points:

```
```
Manifolds, Release 10.3

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 $T_{\Phi(N)} R^3$:

sage: Phi.differential(np).matrix()
[2 0]
[0 2]
[0 0]

the default bases being:

sage: Phi.differential(np).domain().default_basis()
Basis (∂/∂u,∂/∂v) on the Tangent space at Point N on the 2-dimensional differentiable manifold S^2
sage: Phi.differential(np).codomain().default_basis()
Basis (∂/∂X,∂/∂Y,∂/∂Z) 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 $\mathbb{R}^3 \rightarrow \mathbb{R}^2$ 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'\mathbb{R}^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 $S^2 \rightarrow \mathbb{R}^2$:

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 \texttt{ster} is indeed the composite of \texttt{Psi} and \texttt{Phi}:

```
sage: ster(sp) == Psi(Phi(sp))
True
```

Actually \texttt{ster} is the stereographic projection from the North pole, as its coordinate expression reveals:

```
sage: ster.display()
S^2 \rightarrow \mathbb{R}^2
on U: (x, y) \mapsto (xP, yP) = (x, y)
on V: (u, v) \mapsto (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 \( \mathbb{R} \rightarrow \mathbb{R}^2 \):

```
sage: R = Manifold(1, 'R')  # field \mathbb{R}
sage: T.<t> = R.chart()  # canonical chart on \mathbb{R}
sage: R2 = Manifold(2, 'R^2')  # \mathbb{R}^2
sage: c_xy.<x,y> = R2.chart()  # Cartesian coordinates on \mathbb{R}^2
sage: Phi = R.diff_map(R2, [cos(t), sin(t)], name='Phi') ; Phi
Differentiable map Phi from the 1-dimensional differentiable manifold \mathbb{R} to the 2-dimensional differentiable manifold \mathbb{R}^2
sage: Phi.parent()
Set of Morphisms from 1-dimensional differentiable manifold \mathbb{R} to 2-dimensional differentiable manifold \mathbb{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 \rightarrow \mathbb{R}^2
t ↦ (x, y) = (\cos(t), \sin(t))
```

An example of diffeomorphism between the unit open disk and the Euclidean plane \( \mathbb{R}^2 \):

```
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 \mathbb{R}^2 to the 2-dimensional differentiable manifold \mathbb{R}^2
sage: Phi.parent()
Set of Morphisms from Open subset D of the 2-dimensional differentiable manifold \mathbb{R}^2 to 2-dimensional differentiable manifold \mathbb{R}^2 in 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 \rightarrow \mathbb{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:
The inverse diffeomorphism is computed by means of the method `inverse()`:

```sage
sage: Phi.inverse()
Diffeomorphism \(\Phi^{-1}\) from the 2-dimensional differentiable manifold \(\mathbb{R}^2\) to the Open subset \(D\) of the 2-dimensional differentiable manifold \(\mathbb{R}^2\)
```

Equivalently, one may use the notations ^(-1) or ~ to get the inverse:

```sage
sage: Phi^(-1) is Phi.inverse()
True
sage: -Phi is Phi.inverse()
True
```

Check that ~\(\Phi\) is indeed the inverse of \(\Phi\):

```sage
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
sage: (~Phi).display()
\Phi^{-1}: \mathbb{R}^2 \to D
(x, y) \mapsto (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
sage: id = D.identity_map() ; id
Identity map Id_D of the Open subset D of the 2-dimensional differentiable manifold \(\mathbb{R}^2\)

sage: latex(id)
\mathrm{Id}_D

sage: id.parent()
Set of Morphisms from Open subset D of the 2-dimensional differentiable manifold \(\mathbb{R}^2\) to Open subset D of the 2-dimensional differentiable manifold \(\mathbb{R}^2\) 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:
The coordinate expression of the identity map:

\[
\text{sage: id.display()}
\]

\[
\text{Id}_D: D \rightarrow D
\]

\[
(x, y) \mapsto (x, y)
\]

The identity map is its own inverse:

\[
\text{sage: id^(-1) is id}
\]

True

\[
\text{sage: ~id is id}
\]

True

**differential** (**point**)

Return the differential of **self** at a given point.

If the differentiable map **self** is

\[
\Phi: M \rightarrow N,
\]

where \(M\) and \(N\) are differentiable manifolds, the **differential** of \(\Phi\) at a point \(p \in M\) is the tangent space linear map:

\[
d\Phi_p: T_pM \rightarrow T_{\Phi(p)}N
\]

defined by

\[
\forall v \in T_pM, \quad d\Phi_p(v): C^k(N) \rightarrow \mathbb{R}
\]

\[
f \mapsto v(f \circ \Phi)
\]

**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:
To: Tangent space at Point $\Phi(p)$ on the 3-dimensional differentiable manifold $N$

$sage$: latex(dPhip)

\[
\text{\{mathrm{d}\Phi\}_p}
\]

$sage$: 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 $T_p M$ and $T_{\Phi(p)} N$:

$sage$: dPhip.matrix()

\[
\begin{bmatrix}
1 & -2 \\
-1 & 2 \\
4 & -3
\end{bmatrix}
\]

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:

$$d\Phi_p : T_p M \longrightarrow T_{\Phi(p)} N$$

defined by

$$\forall v \in T_p M, \quad d\Phi_p(v) : C^k(N) \longrightarrow \mathbb{R}, \quad f \mapsto v(f \circ \Phi).$$

If the coordinate expression of $\Phi$ is

$$y^i = Y^i(x^1, \ldots, x^n), \quad 1 \leq i \leq m,$$

where $(x^1, \ldots, x^n)$ are coordinates of a chart on $M$ and $(y^1, \ldots, y^m)$ 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 $T_p M$ and $T_{\Phi(p)} N$ associated to the above charts:

$$d\Phi_p \left( \frac{\partial}{\partial x^j} \right)_p = J_{ij}|_p \frac{\partial}{\partial y^i}\Big|_{\Phi(p)}.$$

INPUT:

- **chart1** – (default: None) chart on the domain $M$ of $\Phi$ (coordinates denoted by $(x^i)$ 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 ChartFunction

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.chart_func.ChartFunctionRing_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])
<class '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

$$y^i = Y^i(x^1, \ldots, x^n), \quad 1 \leq i \leq m,$$

where $(x^1, \ldots, x^n)$ are coordinates of a chart $X$ on the domain of $\Phi$ and $(y^1, \ldots, y^m)$ 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:

```python
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_or_codomain_subset, name=None, latex_name=None)

Pullback operator associated with self.

In what follows, let $\Phi$ denote a differentiable map $self$, $M$ its domain and $N$ its codomain.

**INPUT:**

One of the following:
- tensor_or_codomain_subset – one of the following:
  - a 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
  - a ManifoldSubset $S$

**OUTPUT:**
- (if the input is a tensor field $T$) a TensorField representing a fully covariant tensor field on $M$ that is the pullback of $T$ by $\Phi$
- (if the input is a manifold subset $S$) a ManifoldSubset that is the preimage $\Phi^{-1}(S)$; same as preimage()
EXAMPLES:

Pullback on $S^2$ of a scalar field defined on $R^3$:

```sage
M = Manifold(2, 'S^2', start_index=1)
U = M.open_subset('U') # the complement of a meridian (domain of...
# spherical coordinates)
c_spher.<th,ph> = U.chart(r'\theta:(0,pi):\theta ph:(0,2*pi):\phi') #...
# spherical coord. on U
N = Manifold(3, 'R^3', start_index=1)
c_cart.<x,y,z> = N.chart() # Cartesian coord. on R^3
Phi = U.diff_map(N, (sin(th)*cos(ph), sin(th)*sin(ph), cos(th)),
.....: name='Phi', latex_name=r'\Phi')
f = N.scalar_field(x*y*z, name='f'); f
Scalar field f on the 3-dimensional differentiable manifold R^3
f.display()
f: R^3 → R
(x, y, z) ↦ x*y*z
pf = Phi.pullback(f); pf
Scalar field Phi^*(f) on the Open subset U of the 2-dimensional differentiable manifold S^2
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(name='g')
g[1,1], g[2,2], g[3,3] = 1, 1, 1
g.display()
g = dx⊗dx + dy⊗dy + dz⊗dz
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
pg.display()
Phi^*(g) = dth⊗dth + sin(th)^2 dph⊗dph
```

Parallel computation:

```sage
Parallelism().set('tensor', nproc=2)
pq = Phi.pullback(g); pq
Field of symmetric bilinear forms Phi^*(g) on the Open subset U of the 2-dimensional differentiable manifold S^2
pq.display()
Phi^*(g) = dth⊗dth + sin(th)^2 dph⊗dph
Parallelism().set('tensor', nproc=1) # switch off parallelization
```

Pullback on $S^2$ of a 3-form on $R^3$:

```sage
a = N.diff_form(3, name='A')
a[1,2,3] = f
a.display()
A = x*y*z dx∧dy∧dz
pa = Phi.pullback(a); pa
3-form Phi^*(A) on the Open subset U of the 2-dimensional differentiable manifold S^2
pa.display() # should be zero (as any 3-form on a 2-dimensional...
# manifold)
Phi^*(A) = 0
```

2.5. Differentiable Maps and Curves
**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 contravariant 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 $\mathbb{R}^3$, via the standard embedding of $S^2$:

```python
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'\theta:(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')
v[1] = 0, 1
sage: v.display()
v = \partial/\partial \phi
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 $\mathbb{R}^3$
sage: pv.display()
$\Phi_*(v) = -\sin(\phi)\sin(\theta) \partial/\partial x + \cos(\phi)\sin(\theta) \partial/\partial y$
```

Pushforward of a vector field on the real line to the $\mathbb{R}^3$, via a helix embedding:

```python
sage: R.<t> = manifolds.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')
u[0] = 1
sage: u.display()
u = \partial/\partial t
sage: pu = Psi.pushforward(u); pu
Vector field $\Psi_*(u)$ along the Real number line $\mathbb{R}$ with values on the 3-dimensional differentiable manifold $\mathbb{R}^3$
sage: pu.display()
$\Psi_*(u) = -\sin(t) \partial/\partial x + \cos(t) \partial/\partial y + \partial/\partial z$
```
2.5.3 Curves in Manifolds

Given a differentiable manifold $M$, a differentiable curve in $M$ is a differentiable mapping

$$\gamma : I \rightarrow M,$$

where $I$ is an interval of $\mathbb{R}$.

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]

```python
class sage.manifolds.differentiable.curve.DifferentiableCurve(parent, coord_expression=None, name=None, latex_name=None, is_isomorphism=False, is_identity=False)
```

**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 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 Gerono in the 2-dimensional Euclidean plane:
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> = manifolds.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)  # needs sage.plot
```

Curves are considered as (manifold) morphisms from real intervals to differentiable manifolds:
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 $I \to M$:

```
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)
\text{Curve in the 2-dimensional differentiable manifold M}
```

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) \partial/\partial x + (2\cos(t)^2 - 1) \partial/\partial y
```

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' = -\partial/\partial x + \partial/\partial y
```

Curves $\mathbb{R} \to \mathbb{R}$ 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 $\mathbb{R}$ to itself
sage: s.display()
$\mathbb{R} \to \mathbb{R}$
t ↦ \cos(t^2)
sage: s = f*g; s
Differentiable map from the Real number line $\mathbb{R}$ to itself
sage: s.display()
```

(continues on next page)
2.5. Differentiable Maps and Curves
Curvature and torsion of a curve in a Riemannian manifold

Let us consider a helix $C$ in the Euclidean space $\mathbb{E}^3$ parametrized by its arc length $s$:

```
sage: E.<x,y,z> = EuclideanSpace()
sage: R.<s> = manifolds.RealLine()
sage: C = E.curve((2*cos(s/3), 2*sin(s/3), sqrt(5)*s/3), (s, 0, 6*pi),
             ....:     name='C')
```

Its tangent vector field is:

```
sage: T = C.tangent_vector_field()
sage: T.display()
C' = -2/3*sin(1/3*s) e_x + 2/3*cos(1/3*s) e_y + 1/3*sqrt(5) e_z
```

Since $C$ is parametrized by its arc length $s$, $T$ is a unit vector (with respect to the Euclidean metric of $\mathbb{E}^3$):

```
sage: norm(T)
Scalar field $|C'|$ on the Real interval (0, 6*pi)
sage: norm(T).display()
|C'|: (0, 6*pi) → ℝ
s ↦ 1
```

Vector fields along $C$ are defined by the method `vector_field()` of the domain of $C$ with the keyword argument `dest_map` set to $C$. For instance the derivative vector $T' = dT/ds$ is:

```
sage: I = C.domain(); I
Real interval (0, 6*pi)
sage: Tp = I.vector_field([diff(T[i], s) for i in E.irange()], dest_map=C,
             ....:     name="T")
sage: Tp.display()
T' = -2/9*cos(1/3*s) e_x - 2/9*sin(1/3*s) e_y
```

The norm of $T'$ is the curvature of the helix:

```
sage: kappa = norm(Tp)
sage: kappa
Scalar field $|T'|$ on the Real interval (0, 6*pi)
sage: kappa.expr()
2/9
```

The unit normal vector along $C$ is:

```
sage: N = Tp / kappa
sage: N.display()
-cos(1/3*s) e_x - sin(1/3*s) e_y
```

while the binormal vector along $C$ is $B = T \times N$:

```
sage: B = T.cross_product(N)
sage: B
Vector field along the Real interval (0, 6*pi) with values on the
```
Euclidean space $\mathbb{E}^3$
\[
\text{sage: } B\text{.display()}
\]
\[
\frac{1}{3}\sqrt{5}\sin\left(\frac{1}{3}s\right) e_x - \frac{1}{3}\sqrt{5}\cos\left(\frac{1}{3}s\right) e_y + \frac{2}{3} e_z
\]

The three vector fields $(T, N, B)$ form the **Frenet-Serret frame** along $C$:

\[
\text{sage: } \text{FS} = \text{I}\text{.vector_frame}\left(\left[\text{\textquoteleft}T\text{	extquoteright}, \text{\textquoteleft}N\text{	extquoteright}, \text{\textquoteleft}B\text{	extquoteright}\right], \left[\text{\textquoteleft}T\text{	extquoteright}, \text{\textquoteleft}N\text{	extquoteright}, \text{\textquoteleft}B\text{	extquoteright}\right],
\text{\textquoteright}\text{symbol\_dual}=(\text{\textquoteleft}t\text{	extquoteright}, \text{\textquoteleft}n\text{	extquoteright}, \text{\textquoteleft}b\text{	extquoteright})\right)\right)
\]
\[
\text{sage: } \text{FS}
\]

Vector frame $((0, 6\pi), (T,N,B))$ with values on the Euclidean space $\mathbb{E}^3$

The Frenet-Serret frame is orthonormal:

\[
\text{sage: } \text{matrix}([\left[u\text{.dot}(v)\text{.expr()} \text{ for } v \text{ in } \text{FS}] \text{ for } u \text{ in } \text{FS}])
\]
\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 
\end{bmatrix}
\]

The derivative vectors $N'$ and $B'$:

\[
\text{sage: } \text{Np} = \text{I}\text{.vector_field}\left(\left[\text{diff}(\text{N}[i], s) \text{ for } i \text{ in } \text{E}\text{.irange()}\right],
\text{\textquoteright}\text{dest\_map}=`\text{C}, \text{\textquoteleft}name\text{\textquoteleft}=`\text{N}`\text{\textquoteright})\right)
\]
\[
\text{sage: } \text{Np}\text{.display()}
\]

\[
N = \frac{1}{3}\sin\left(\frac{1}{3}s\right) e_x - \frac{1}{3}\cos\left(\frac{1}{3}s\right) e_y
\]

\[
\text{sage: } \text{Bp} = \text{I}\text{.vector_field}\left(\left[\text{diff}(\text{B}[i], s) \text{ for } i \text{ in } \text{E}\text{.irange()}\right],
\text{\textquoteright}\text{dest\_map}=`\text{C}, \text{\textquoteleft}name\text{\textquoteleft}=`\text{B}`\text{\textquoteright})\right)
\]
\[
\text{sage: } \text{Bp}\text{.display()}
\]

\[
B = \frac{1}{9}\sqrt{5}\cos\left(\frac{1}{3}s\right) e_x + \frac{1}{9}\sqrt{5}\sin\left(\frac{1}{3}s\right) e_y
\]

The Frenet-Serret formulas:

\[
\text{sage: } \text{for } v \text{ in } (\text{Tp, Np, Bp}): 
\]
\[
\text{...... } v\text{.display}(\text{FS})
\]
\[
\text{...... } T' = \frac{2}{9} N
\]
\[
N' = -\frac{2}{9} T + \frac{1}{9}\sqrt{5} B
\]
\[
B' = -\frac{1}{9}\sqrt{5} N
\]

The torsion of $C$ is obtained as the third component of $N'$ in the Frenet-Serret frame:

\[
\text{sage: } \text{tau} = \text{Np}[\text{FS, 3}]
\]
\[
\text{sage: } \text{tau}
\]
\[
\frac{1}{9}\sqrt{5}
\]

`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:
```python
sage: M = Manifold(2, 'R^2', r'\mathbb{R}^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(rr:(0,+oo) ph:(0,2*pi): \phi) # spherical...
˓→coordinates on U
```

Links between spherical coordinates and Cartesian ones:

```python
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))
Check of the inverse coordinate transformation:
  x == x *passed*
  y == y *passed*
  r == r *passed*
  ph == arctan2(r*sin(ph), r*cos(ph)) **failed**
NB: a failed report can reflect a mere lack of simplification.
sage: R.<t> = manifolds.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:

```python
sage: c.coord_expr()
(cos(t), sin(t))
```

Cartesian expression of a cardioid:

```python
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))
```

```python
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, thickness=1,
˓→plot_points=75, max_range=8, aspect_ratio='automatic', **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.

472 Chapter 2. Differentiable Manifolds
• **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 -Infinity replaced by -\(\text{max\_range}\) and +Infinity by \(\text{max\_range}\))

• **include_end_point** — (default: (True, True)) determines whether the end points of \(\text{prange}\) 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 \(\text{include\_end\_point}[0] == \text{False}\), the minimal value of the curve parameter used for the plot is \(\text{prange}[0] + \text{end\_point\_offset}[0]\), while if \(\text{include\_end\_point}[1] == \text{False}\), the maximal value is \(\text{prange}[1] - \text{end\_point\_offset}[1]\).

• **max\_range** — (default: 8) numerical value substituted to +Infinity if the latter is the upper bound of the parameter range; similarly -\(\text{max\_range}\) is the numerical value substituted for -Infinity

• **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
sage: R2 = Manifold(2, 'R^2')
sage: X.<x,y> = R2.chart()
sage: R.<t> = manifolds.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
sage: c.plot(prange=(0,pi))
Graphics object consisting of 1 graphics primitive
```

Plot with various options:

```sage
sage: c.plot(color='green', style='-', thickness=3, aspect_ratio=1)
Graphics object consisting of 1 graphics primitive
```

Cardioid defined in terms of polar coordinates and plotted with respect to Cartesian coordinates, as an example of use of the optional argument chart:

```sage
sage: c.plot()  # 2D plot
```

Cardioid defined in terms of polar coordinates and plotted with respect to Cartesian coordinates, as an example of use of the optional argument chart:
Plot via a mapping to another manifold: loxodrome of a sphere viewed in $\mathbb{R}^3$:

\begin{verbatim}
sage: S2 = Manifold(2, 'S^2')
sage: U = S2.open_subset('U')
sage: XS.<th,ph> = U.chart(r'\theta:(0,\pi):\theta \ ph:(0,2\pi):\phi')
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: (θ, ph) ↦ (x, y, z) = (cos(ph) * sin(θ), sin(ph) * sin(θ), cos(θ))
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
\end{verbatim}
Example of use of the argument parameters: we define a curve with some symbolic parameters \(a\) and \(b\):

```python
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:

```python
sage: c.plot(parameters={a: 2, b: -3}, aspect_ratio=1)
```

\[
\text{tangent_vector_field}(\text{name}=\text{None}, \text{latex_name}=\text{None})
\]

Return the tangent vector field to the curve (velocity vector).

**INPUT:**

- \text{name} – (default: \text{None}) string; symbol given to the tangent vector field; if none is provided, the primed curve symbol (if any) will be used
- \text{latex_name} – (default: \text{None}) string; LaTeX symbol to denote the tangent vector field; if \text{None} then (i) if \text{name} is \text{None} as well, the primed curve LaTeX symbol (if any) will be used or (ii) if \text{name} is not \text{None}, \text{name} will be used

**OUTPUT:**

- the tangent vector field, as an instance of \text{VectorField}

**EXAMPLES:**
Tangent vector field to a circle curve in $\mathbb{R}^2$:

```
sage: M = Manifold(2, 'R^2')
sage: X.<x,y> = M.chart()
sage: R.<t> = manifolds.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) \partial/\partial x + cos(t) \partial/\partial y
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 = -\partial/\partial y
```

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='V', restrictions1 = x^2+y^2!=0,
    restrictions2= u^2+v^2!=0)
```

Then we define a curve (a loxodrome) by its expression in terms of spherical coordinates and evaluate the tangent vector field:

```
sage: R.<t> = manifolds.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
```
2.5.4 Integrated Curves and Geodesics in Manifolds

Given a differentiable manifold $M$, an integrated curve in $M$ is a differentiable curve constructed as a solution to a system of second order differential equations.

Integrated curves are implemented by the class `IntegratedCurve`, from which the classes `IntegratedAutoparallelCurve` and `IntegratedGeodesic` inherit.

**Example: a geodesic in the hyperbolic plane**

First declare the hyperbolic plane as a 2-dimensional Riemannian manifold $\mathbb{M}$ and introduce the chart $X$ corresponding to the Poincaré half-plane model:

```
sage: M = Manifold(2, 'M', structure='Riemannian')
sage: X.<x,y> = M.chart('x y:(0,+\infty)')
```

Then set the metric to be the hyperbolic one:

```
sage: g = M.metric()
sage: g[0,0], g[1,1] = 1/y^2, 1/y^2
sage: g.display()
g = y^{-2}\,dx\otimes dx + y^{-2}\,dy\otimes dy
```

Pick an initial point and an initial tangent vector:

```
sage: p = M((0,1), name='p')
sage: v = M.tangent_space(p)((1,3/2), name='v')
sage: v.display()
v = \partial/\partial x + 3/2 \partial/\partial y
```

Declare a geodesic with such initial conditions, denoting by $t$ the corresponding affine parameter:

```
sage: t = var('t')
sage: c = M.integrated_geodesic(g, (t, 0, 10), v, name='c')
```

Numerically integrate the geodesic (see `solve()`) for all possible options, including the choice of the numerical algorithm:

```
sage: sol = c.solve() #... needs scipy
```

Plot the geodesic after interpolating the solution `sol`:
\( c \) is a differentiable curve in \( M \) and inherits from the properties of \textit{DifferentiableCurve}:

\[
\begin{align*}
\text{sage: } & \quad \text{c.domain()} \\
& \quad \text{Real interval} \ (0, 10) \\
\text{sage: } & \quad \text{c.codomain()} \\
& \quad 2\text{-dimensional Riemannian manifold} \ M \\
\text{sage: } & \quad \text{c.display()} \\
& \quad \text{c: } (0, 10) \rightarrow M
\end{align*}
\]

In particular, its value at \( t = 1 \) is:

\[
\begin{align*}
\text{sage: } & \quad \text{c(1)} \\
& \quad \text{Point on the 2\text{-dimensional Riemannian manifold} } M
\end{align*}
\]

which corresponds to the following \((x, y)\) coordinates:
AUTHORS:

- Karim Van Aelst (2017): initial version
- Florentin Jaffredo (2018): integration over multiple charts, use of fast_callable to improve the computation speed

```python
sage: X(c(1))  # abs tol 1e-12
(2.4784140715580136, 1.5141683866138937)
```

class `sage.manifolds.differentiable.integrated_curve.IntegratedAutoparallelCurve`

Bases: `IntegratedCurve`

Autoparallel curve on the manifold with respect to a given affine connection.

INPUT:

- **parent** -- `IntegratedAutoparallelCurveSet` the set of curves $\text{Hom}_\text{autoparallel}(I, M)$ to which the curve belongs
- **affine_connection** -- `AffineConnection` affine connection with respect to which the curve is autoparallel
- **curve_parameter** -- symbolic expression to be used as the parameter of the curve (the equations defining an instance of `IntegratedAutoparallelCurve` are such that $t$ will actually be an affine parameter of the curve
- **initial_tangent_vector** -- `TangentVector` initial tangent vector of the curve
- **chart** -- (default: `None`) chart on the manifold in terms of which the equations are expressed; 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

EXAMPLES:

Autoparallel curves associated with the Mercator projection of the unit 2-sphere $S^2$. 

2.5. Differentiable Maps and Curves
See also:

On the Mercator projection, the lines of longitude all appear vertical and then all parallel with respect to each other. Likewise, all the lines of latitude appear horizontal and parallel with respect to each other. These curves may be recovered as autoparallel curves of a certain connection $\nabla$ to be made explicit.

Start with declaring the standard polar coordinates $(\theta, \phi)$ on $S^2$ and the corresponding coordinate frame $(e_\theta, e_\phi)$:

```python
sage: S2 = Manifold(2, 'S^2', start_index=1)
sage: polar.<th,ph>=S2.chart()
sage: epolar = polar.frame()
```

Normalizing $e_\phi$ provides an orthonormal basis:

```python
sage: ch_basis = S2.automorphism_field()
sage: ch_basis[1,1], ch_basis[2,2] = 1, 1/sin(th)
sage: epolar_ON = epolar.new_frame(ch_basis,'epolar_ON')
```

Denote $(\hat{e}_\theta, \hat{e}_\phi)$ such an orthonormal frame field. In any point, the vector field $\hat{e}_\theta$ is normalized and tangent to the line of longitude through the point. Likewise, $\hat{e}_\phi$ is normalized and tangent to the line of latitude.

Now, set an affine connection with respect to such fields that are parallelly transported in all directions, that is: $\nabla \hat{e}_\theta = \nabla \hat{e}_\phi = 0$. This is equivalent to setting all the connection coefficients to zero with respect to this frame:

```python
sage: nab = S2.affine_connection('nab')
sage: nab.set_coef(frame=epolar_ON)[:] = [[[0, 0], [0, 0]], [[0, 0], [0, 0]]]
```

This connection is such that two vectors are parallel if their angles to a given meridian are the same. Check that this connection is compatible with the Euclidean metric tensor $g$ induced on $S^2$:

```python
sage: g = S2.metric('g')
sage: g[1,1], g[2,2] = 1, (sin(th))^2
sage: nab(g)[]
```

Yet, this connection is not the Levi-Civita connection, which implies that it has non-vanishing torsion:

```python
sage: nab.torsion()[:]
```

Set generic initial conditions for the autoparallel curves to compute:

```python
sage: [th0, ph0, v_th0, v_ph0] = var('th0 ph0 v_th0 v_ph0')
sage: p = S2.point((th0, ph0), name='p')
sage: Tp = S2.tangent_space(p)
sage: v = Tp((v_th0, v_ph0), basis=epolar_ON.at(p))
```

Note here that the components $(v_{th0}, v_{ph0})$ of the initial tangent vector $v$ refer to the basis $\text{epolar\_ON} = (\hat{e}_\theta, \hat{e}_\phi)$ and not the coordinate basis $\text{epolar} = (e_\theta, e_\phi)$. This is merely to help picture the aspect of the tangent vector in the usual embedding of $S^2$ in $\mathbb{R}^3$ thanks to using an orthonormal frame, since providing the components with respect to the coordinate basis would require multiplying the second component (i.e. the $\phi$ component) in order to picture the vector in the same way. This subtlety will need to be taken into account later when the numerical curve will be compared to the analytical solution.

Now, declare the corresponding integrated autoparallel curve and display the differential system it satisfies:
sage: [t, tmin, tmax] = var('t tmin tmax')
sage: c = S2.integrated_autoparallel_curve(nab, (t, tmin, tmax),
......: v, chart=polar, name='c')
sage: sys = c.system(verbos=True)
Autoparallel curve c in the 2-dimensional differentiable
manifold S^2 equipped with Affine connection nab on the
2-dimensional differentiable manifold S^2, and integrated over
the Real interval (tmin, tmax) as a solution to the following
equations, written with respect to Chart (S^2, (th, ph)):

Initial point: Point p on the 2-dimensional differentiable
manifold S^2 with coordinates [th0, ph0] with respect to
Chart (S^2, (th, ph))
Initial tangent vector: Tangent vector at Point p on the
2-dimensional differentiable manifold S^2 with
components [v_th0, v_ph0/sin(th0)] with respect to Chart (S^2, (th, ph))

d(th)/dt = Dth
d(ph)/dt = Dph
d(Dth)/dt = 0
d(Dph)/dt = -Dph*Dth*cos(th)/sin(th)

Set a dictionary providing the parameter range and the initial conditions for a line of latitude and a line of longitude:

sage: dict_params={'latit':{tmin:0,tmax:3,th0:pi/4,ph0:0.1,v_th0:0,v_ph0:1},
......: 'longi':{tmin:0,tmax:3,th0:0.1,ph0:0.1,v_th0:1,v_ph0:0}}

Declare the Mercator coordinates (ξ, ζ) and the corresponding coordinate change from the polar coordinates:

sage: mercator.<xi,ze> = S2.chart(r'xi:(-oo,oo):xi ze:(0,2*pi): \zeta')
sage: polar.transition_map(mercator, (log(tan(th /2)), ph))
Change of coordinates from Chart (S^2, (th, ph)) to Chart
(S^2, (xi, ze))

Ask for the identity map in terms of these charts in order to add this coordinate change to its dictionary of expres-
sions. This is required to plot the curve with respect to the Mercator chart:

sage: identity = S2.identity_map()
sage: identity.coord_functions(polar, mercator)
Coordinate functions (log(sin(1/2*th)/cos(1/2*th)), ph) on the
Chart (S^2, (th, ph))

Solve, interpolate and prepare the plot for the solutions corresponding to the two initial conditions previously set:

sage: graph2D_mercator = Graphics()
sage: for key in dict_params:
......: sol = c.solve(solution_key='sol'+key,
......: parameters_values=dict_params[key])
......: interp = c.interpolate(solution_key='sol'+key,
......: interpolation_key='interp'+key)
......: graph2D_mercator+=c.plot_integrated(interpolation_key='interp'+key,
......: chart=mercator, thickness=2)

Prepare a grid of Mercator coordinates lines, and plot the curves over it:

sage: graph2D_mercator_coords=mercator.plot(chart=mercator,
......: number_values=8,color='yellow')
The resulting curves are horizontal and vertical as expected. It is easier to check that these are latitude and longitude lines respectively when plotting them on $S^2$. To do so, use $\mathbb{R}^3$ as the codomain of the standard map embedding $(S^2, (\theta, \phi))$ in the 3-dimensional Euclidean space:

```
sage: R3 = Manifold(3, 'R3', start_index=1)
sage: cart.<X,Y,Z> = R3.chart()
sage: euclid_embedding = S2.diff_map(R3, 
    ....:   {(polar, cart):[sin(th)*cos(ph),sin(th)*sin(ph),cos(th)])})
```

Plot the resulting curves on the grid of polar coordinates lines on $S^2$:

```
sage: graph3D_embedded_curves = Graphics()
sage: for key in dict_params:
    ....:     graph3D_embedded_curves += c.plot_integrated(interpolation_key='interp-
    ....:     \rightarrow'+key,
    ....:        mapping=euclid_embedding, thickness=5,
    ....:        display_tangent=True, scale=0.4, width_tangent=0.5)
sage: graph3D_embedded_polar_coords = polar.plot(chart=cart, 
    ....:        mapping=euclid_embedding, 
    ....:        number_values=15, color='yellow')
```
Finally, one may plot a general autoparallel curve with respect to $\nabla$ that is neither a line of latitude or longitude. The vectors tangent to such a curve make an angle different from 0 or $\pi/2$ with the lines of latitude and longitude. Then, compute a curve such that both components of its initial tangent vectors are non-zero:

```sage
sage: sol = c.solve(solution_key='sol-angle',
...: parameters_values={tmin:0,tmax:2,th0:pi/4,ph0:0.1,v_th0:1,v_ph0:8})
sage: interp = c.interpolate(solution_key='sol-angle',
...: interpolation_key='interp-angle')
```

Plot the resulting curve in the Mercator plane. This generates a straight line, as expected:

```sage
sage: c.plot_integrated(interpolation_key='interp-angle',
...: chart=mercator, thickness=1, display_tangent=True,
...: scale=0.2, width_tangent=0.2)
```

One may eventually plot such a curve on $S^2$:

```sage
sage: graph3D_embedded_angle_curve=c.plot_integrated(interpolation_key='interp-angle',
```

(continues on next page)
All the curves presented are loxodromes, and the differential system defining them (displayed above) may be solved analytically, providing the following expressions:

\[
\begin{align*}
\theta(t) &= \theta_0 + \dot{\theta}_0 (t - t_0), \\
\phi(t) &= \phi_0 - \frac{1}{\tan \alpha} \left( \ln \tan \frac{\theta_0 + \dot{\theta}_0 (t - t_0)}{2} - \ln \tan \frac{\theta_0}{2} \right),
\end{align*}
\]

where \(\alpha\) is the angle between the curve and any latitude line it crosses; then, one finds \(\tan \alpha = -\dot{\theta}_0 / (\dot{\phi}_0 \sin \theta_0)\) (then \(\tan \alpha \leq 0\) when the initial tangent vector points towards the southeast).

In order to use these expressions to compare with the result provided by the numerical integration, remember that the components \((v_{\text{th}0}, v_{\text{ph}0})\) of the initial tangent vector \(v\) refer to the basis \(\text{epolar\_ON} = (\hat{\epsilon}_\theta, \hat{\epsilon}_\phi)\) and not the coordinate basis \(\text{epolar} = (\epsilon_\theta, \epsilon_\phi)\). Therefore, the following relations hold: \(v_{\text{ph}0} = \dot{\phi}_0 \sin \theta_0\) (and not merely \(\dot{\phi}_0\)), while \(v_{\text{th}0}\) clearly is \(\dot{\theta}_0\).

With this in mind, plot an analytical curve to compare with a numerical solution:
Ask for the expression of the loxodrome in terms of the Mercator chart in order to add it to its dictionary of expressions. It is a particularly long expression, and there is no particular need to display it, which is why it may simply be affected to an arbitrary variable `expr_mercator`, which will never be used again. But adding the expression to the dictionary is required to plot the curve with respect to the Mercator chart:

```sage```
```
expr_mercator = c_loxo.expression(chart2=mercator)
```

Plot the curves (for clarity, set a 2 degrees shift in the initial value of $\theta_0$ so that the curves do not overlap):

```sage```
```
graph2D_mercator_loxo = c_loxo.plot(chart=mercator,
....:     parameters={(th0:pi/4+2*pi/180, ph0:0.1, v_th0:1, v_ph0:8),
....:         thickness=1, color='blue')
```

Graphics object consisting of 2 graphics primitives

Both curves do have the same aspect. One may eventually compare these curves on $S^2$: 

![Graph showing two curves on a 2D plot with ξ on the x-axis and ζ on the y-axis. The curves are plotted in red and blue, with the red curve slightly shifted towards the left compared to the blue curve.]
system (verbose=False)

Provide a detailed description of the system defining the autoparallel curve and returns the system defining it: chart, equations and initial conditions.

INPUT:

- **verbose** *(default: False)* prints a detailed description of the curve

OUTPUT:

- list containing the
  - the equations
  - the initial conditions
  - the chart

2.5. Differentiable Maps and Curves 491
EXAMPLES:

System defining an autoparallel curve:

```python
sage: M = Manifold(3, 'M')
sage: X.<x1,x2,x3> = M.chart()
sage: [t, A, B] = var('t A B')
sage: nab = M.affine_connection('nabla', r'nabla')
sage: nab[X.frame(),0,0,1],nab[X.frame(),2,1,2]=A*x1^2,B*x2*x3
sage: p = M.point((0,0,0), name='p')
sage: Tp = M.tangent_space(p)
sage: v = Tp((1,0,1))
sage: c = M.integrated_autoparallel_curve(nab, (t, 0, 5), v)
sage: sys = c.system(verbose=True)
Autoparallel curve in the 3-dimensional differentiable
manifold M equipped with Affine connection nabla on the
3-dimensional differentiable manifold M, and integrated
over the Real interval (0, 5) as a solution to the
following equations, written with respect to
Chart (M, (x1, x2, x3)):

Initial point: Point p on the 3-dimensional differentiable
manifold M with coordinates [0, 0, 0] with respect to
Chart (M, (x1, x2, x3))
Initial tangent vector: Tangent vector at Point p on the
3-dimensional differentiable manifold M with
components [1, 0, 1] with respect to Chart (M, (x1, x2, x3))

d(x1)/dt = Dx1
d(x2)/dt = Dx2
d(x3)/dt = Dx3
d(Dx1)/dt = -A*Dx1*Dx2*x1^2
\sage: sys_bis = c.system()
\sage: sys_bis == sys
True
```

```python
class sage.manifolds.differentiable.integrated_curve.IntegratedCurve (parent, equations_rhs, velocities, curve_parameter, initial_tangent_vector, chart=None, name=None, latex_name=None, verbose=False, across_charts=False)
```

Bases: DifferentiableCurve

Given a chart with coordinates denoted \((x_1,\ldots,x_n)\), an instance of `IntegratedCurve` is a curve \(t \mapsto (x_1(t),\ldots,x_n(t))\) constructed as a solution to a system of second order differential equations satisfied by the coordinate curves \(t \mapsto x_i(t)\).
INPUT:

- **parent** — *IntegratedCurveSet* the set of curves $\text{Hom}_{\text{integrated}}(I, M)$ to which the curve belongs
- **equations_rhs** — list of the right-hand sides of the equations on the velocities only (the term *velocity* referring to the derivatives $dx_i/dt$ of the coordinate curves)
- **velocities** — list of the symbolic expressions used in `equations_rhs` to denote the velocities
- **curve_parameter** — symbolic expression used in `equations_rhs` to denote the parameter of the curve (denoted $t$ in the descriptions above)
- **initial_tangent_vector** — *TangentVector* initial tangent vector of the curve
- **chart** — (default: None) chart on the manifold in which the equations are given; 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

**EXAMPLES:**

Motion of a charged particle in an axial magnetic field linearly increasing in time and exponentially decreasing in space:

$$ B(t, x) = \frac{B_0 t}{T} \exp\left(-\frac{x_1^2 + x_2^2}{L^2}\right) e_3. $$

Equations of motion are:

\[
\begin{align*}
\ddot{x}_1(t) &= \frac{qB(t, x(t))}{m} \dot{x}_2(t), \\
\ddot{x}_2(t) &= -\frac{qB(t, x(t))}{m} \dot{x}_1(t), \\
\ddot{x}_3(t) &= 0.
\end{align*}
\]

Start with declaring a chart on a 3-dimensional manifold and the symbolic expressions denoting the velocities and the various parameters:

```
sage: M = Manifold(3, 'M', start_index=1)
sage: X.<x1,x2,x3> = M.chart()
sage: var('t B_0 m q L T')
(t, B_0, m, q, L, T)
sage: B = B_0*t/T*exp(-(x1^2 + x2^2)/L^2)
sage: D = X.symbolic_velocities(); D
[Dx1, Dx2, Dx3]
sage: eqns = [q*B/m*D[1], -q*B/m*D[0], 0]
```

Set the initial conditions:

```
sage: p = M.point((0,0,0), name='p')
sage: Tp = M.tangent_space(p)
sage: v = Tp((1,0,1))
```

Declare an integrated curve and display information relative to it:

```
sage: c = M.integrated_curve(eqns, D, (t, 0, 5), v, name='c', verbose=True)
```

The curve was correctly set.
Parameters appearing in the differential system defining the curve are $[B_0, L, T, m, q]$.

Integrated curve $c$ in the 3-dimensional differentiable manifold $M$

```
sage: c
```

Curve $c$ in the 3-dimensional differentiable manifold $M$
integrated over the Real interval $(0, 5)$ as a solution to the following system, written with respect to Chart $(M, (x_1, x_2, x_3))$:

Initial point: Point $p$ on the 3-dimensional differentiable manifold $M$ with coordinates $[0, 0, 0]$ with respect to Chart $(M, (x_1, x_2, x_3))$

Initial tangent vector: Tangent vector at Point $p$ on the 3-dimensional differentiable manifold $M$ with components $[1, 0, 1]$ with respect to Chart $(M, (x_1, x_2, x_3))$

\[
\begin{align*}
\frac{dx_1}{dt} &= D_x^1 \\
\frac{dx_2}{dt} &= D_x^2 \\
\frac{dx_3}{dt} &= D_x^3 \\
\frac{dD_x^1}{dt} &= B_0 D_x^2 q t e^{-\frac{(x_1^2 + x_2^2)}{L^2}}/(T m) \\
\frac{dD_x^2}{dt} &= -B_0 D_x^1 q t e^{-\frac{(x_1^2 + x_2^2)}{L^2}}/(T m) \\
\frac{dD_x^3}{dt} &= 0
\end{align*}
\]

Generate a solution of the system and an interpolation of this solution:

```
sage: sol = c.solve(step=0.2, # needs scipy
....:     parameters_values={B_0:1, m: 1, q:1, L:10, T:1},
....:     solution_key='carac time 1', verbose=True)
```

Performing numerical integration with method 'odeint'...
Numerical integration completed.
Checking all points are in the chart domain...
All points are in the chart domain.
The resulting list of points was associated with the key 'carac time 1' (if this key already referred to a former numerical solution, such a solution was erased).
```
sage: interp = c.interpolate(solution_key='carac time 1', # needs scipy
....:     interpolation_key='interp 1', verbose=True)
```
Performing cubic spline interpolation by default...
Interpolation completed and associated with the key 'interp 1' (if this key already referred to a former interpolation, such an interpolation was erased).

Such an interpolation is required to evaluate the curve and the vector tangent to the curve for any value of the curve parameter:

```
sage: # needs scipy
sage: p = c(1.9, verbose=True)
Sage: p
```
Evaluating point coordinates from the interpolation associated with the key 'interp 1' by default...
```
sage: p
Point on the 3-dimensional differentiable manifold $M$
```

(continues from previous page)
Plotting a numerical solution (with or without its tangent vector field) also requires the solution to be interpolated at least once:

```sage
interpolate(solution_key=None, method=None, interpolation_key=None, verbose=False)
```

Interpolate the chosen numerical solution using the given interpolation method.

INPUT:

- solution_key – (default: None) key which the numerical solution to interpolate is associated to; a
2.5. Differentiable Maps and Curves
default value is chosen if none is provided

- method – (default: None) interpolation scheme to use; algorithms available are
  - 'cubic spline', which makes use of GSL via Spline

- interpolation_key – (default: None) key which the resulting interpolation will be associated to; a default value is given if none is provided

- verbose – (default: False) prints information about the interpolation in progress

OUTPUT:

- built interpolation object

EXAMPLES:

Interpolating a numerical solution previously computed:

```sage
sage: M = Manifold(3, 'M')
sage: X.<x1,x2,x3> = M.chart()
sage: [t, B_0, m, q, L, T] = var('t B_0 m q L T')
sage: B = B_0*t/T*exp(-(x1^2 + x2^2)/L^2)
sage: D = X.symbolic_velocities()
sage: eqns = [q*B/m*D[1], -q*B/m*D[0], 0]
sage: p = M.point((0,0,0), name='p')
sage: Tp = M.tangent_space(p)
sage: v = Tp((1,0,1))
sage: c = M.integrated_curve(eqns, D, (t, 0,5), v, name =c)
sage: # needs scipy
sage: sol = c.solve(method='odeint',
.....: solution_key='sol_T1',
.....: parameters_values=(B_0:1, m:1, q:1, L:10, T:1))
sage: interp = c.interpolate(method='cubic spline',
.....: solution_key='sol_T1',
.....: interpolation_key='interp_T1',
.....: verbose=True)
```

Interpolation completed and associated with the key 'interp_T1' (if this key already referred to a former interpolation, such an interpolation was erased).

```sage
sage: interp = c.interpolate(verbos=True)
```

Interpolating the numerical solution associated with the key 'sol_T1' by default...

Performing cubic spline interpolation by default...

Resulting interpolation will be associated with the key 'cubic spline-interp-sol_T1' by default.

Interpolation completed and associated with the key 'cubic spline-interp-sol_T1' (if this key already referred to a former interpolation, such an interpolation was erased).

**interpolation** (interpolation_key=None, verbose=False)

Return the interpolation object associated with the given key.

INPUT:

- interpolation_key – (default: None) key which the requested interpolation is associated to; a default value is chosen if none is provided

- verbose – (default: False) prints information about the interpolation object returned

OUTPUT:
• requested interpolation object

EXAMPLES:

Requesting an interpolation object previously computed:

```
sage: M = Manifold(3, 'M')
sage: X.<x1,x2,x3> = M.chart()
sage: [t, B_0, m, q, L, T] = var('t B_0 m q L T')
sage: B = B_0*t/T*exp(-(x1^2 + x2^2)/L^2)
sage: D = X.symbolic_velocities()
sage: eqns = [q*B/m*D[1], -q*B/m*D[0], 0]
sage: p = M.point((0,0,0), name='p')
sage: Tp = M.tangent_space(p)
sage: v = Tp((1,0,1))
sage: c = M.integrated_curve(eqns, D, (t, 0,5), v, name='c')

sage: # needs scipy
sage: sol = c.solve(method='odeint',
....:     solution_key='sol_T1',
....:     parameters_values={(B_0:1, m:1, q:1, L:10, T:1)}

sage: interp = c.interpolate(method='cubic spline',
....:     solution_key='sol_T1',
....:     interpolation_key='interp_T1')

sage: default_interp = c.interpolation(verbose=True)
Returning the interpolation associated with the key 'interp_T1' by default...
sage: default_interp == interp
True
sage: interp_mute = c.interpolation()
```

**plot_integrated**

```
plot_integrated(chart=None, ambient_coords=None, mapping=None, prange=None,
interpolation_key=None, include_end_point=(True, True), end_point_offset=(0.001, 0.001), verbose=False, color='red', style='-', label_axes=True, display_tangent=False, color_tangent='blue', across_charts=False, thickness=1, plot_points=75, aspect_ratio='automatic', plot_points_tangent=10, width_tangent=1, scale=1, **kwds)
```

Plot the 2D or 3D projection of self onto the space of the chosen two or three ambient coordinates, based on the interpolation of a numerical solution previously computed.

See also:

**plot** for complete information about the input.

ADDITIONAL INPUT:

• interpolation_key – (default: None) key associated to the interpolation object used for the plot; a default value is chosen if none is provided
• verbose – (default: False) prints information about the interpolation object used and the plotting in progress
• display_tangent – (default: False) determines whether some tangent vectors should also be plotted
• color_tangent – (default: blue) color of the tangent vectors when these are plotted
• plot_points_tangent – (default: 10) number of tangent vectors to display when these are plotted
• width_tangent – (default: 1) sets the width of the arrows representing the tangent vectors
• scale – (default: 1) scale applied to the tangent vectors before displaying them

EXAMPLES:

Trajectory of a particle of unit mass and unit charge in an unit, axial, uniform, stationary magnetic field:

```sage
M = Manifold(3, 'M')
X.<x1,x2,x3> = M.chart()
var('t')
D = X.symbolic_velocities()
eqns = [D[1], -D[0], 0]
p = M.point((0,0,0), name='p')
Tp = M.tangent_space(p)
v = Tp((1,0,1))
c = M.integrated_curve(eqns, D, (t, 0,6), v, name='c')
sage: # needs scipy
sage: sol = c.solve()
```

Plotting from the interpolation associated with the key 'cubic spline-interp-odeint' by default...

A tiny final offset equal to 0.000301507537688442 was introduced for the last point in order to safely compute it from the interpolation.

```sage
c_plot_2d = c.plot_integrated(ambient_coords=[x1, x2],
.....: thickness=2.5,
.....: display_tangent=True, plot_points=200,
.....: plot_points_tangent=10, scale=0.5,
.....: color='blue', color_tangent='red',
.....: verbose=True)
```

```
solution (solution_key=None, verbose=False)
Return the solution (list of points) associated with the given key.

INPUT:
- solution_key – (default: None) key which the requested numerical solution is associated to; a default value is chosen if none is provided
- verbose – (default: False) prints information about the solution returned

OUTPUT:
- list of the numerical points of the solution requested

EXAMPLES:

Requesting a numerical solution previously computed:

```sage
M = Manifold(3, 'M')
X.<x1,x2,x3> = M.chart()
[t, B_0, m, q, L, T] = var('t B_0 m q L T')
B = B_0*t/T*exp(-(x1^2 + x2^2)/L^2)
D = X.symbolic_velocities()
eqns = [q*B/m*D[1], -q*B/m*D[0], 0]
p = M.point((0,0,0), name='p')
Tp = M.tangent_space(p)
v = Tp((1,0,1))
```
```
sage: c = M.integrated_curve(eqns, D, (t,0,5), v, name='c')
sage: # needs scipy
sage: sol = c.solve(solution_key='sol_T1',
....: parameters_values={B_0:1, m:1, q:1, L:10, T:1})
sage: sol_bis = c.solution(«verbose=True»)
Returning the numerical solution associated with the key 'sol_T1' by default...
sage: sol_bis == sol
True
sage: sol_ter = c.solution(solution_key='sol_T1')
sage: sol_ter == sol
True
sage: sol_mute = c.solution()
sage: sol_mute == sol
True

solve(«step=None, method='odeint', solution_key=None, parameters_values=None, verbose=False, **control_param»)
Integrate the curve numerically over the domain of definition.

INPUT:

- **step** – (default: None) step of integration; default value is a hundredth of the domain of integration if none is provided
- **method** – (default: 'odeint') numerical scheme to use for the integration of the curve; available algorithms are:
  - 'odeint' - makes use of scipy.integrate.odeint via Sage solver desolve_odeint(); odeint invokes the LSODA algorithm of the ODEPACK suite, which automatically selects between implicit Adams method (for non-stiff problems) and a method based on backward differentiation formulas (BDF) (for stiff problems).
  - 'rk4_maxima' - 4th order classical Runge-Kutta, which makes use of Maxima’s dynamics package via Sage solver desolve_system_rk4() (quite slow)
  - 'dopri5' - Dormand-Prince Runge-Kutta of order (4,5) provided by scipy.integrate.ode
  - 'dop853' - Dormand-Prince Runge-Kutta of order 8(5,3) provided by scipy.integrate.ode
and those provided by GSL via Sage class ode_solver:
  - 'rk2' - embedded Runge-Kutta (2,3)
  - 'rk4' - 4th order classical Runge-Kutta
  - 'rkf45' - Runge-Kutta-Felhberg (4,5)
  - 'rkck' - embedded Runge-Kutta-Cash-Karp (4,5)
  - 'rk8pd' - Runge-Kutta Prince-Dormand (8,9)
  - 'rk2imp' - implicit 2nd order Runge-Kutta at Gaussian points
  - 'rk4imp' - implicit 4th order Runge-Kutta at Gaussian points
  - 'gear1' - $M = 1$ implicit Gear
  - 'gear2' - $M = 2$ implicit Gear
  - 'bsimp' - implicit Bulirsch-Stoer (requires Jacobian)
• solution_key – (default: None) key which the resulting numerical solution will be associated to; a default value is given if none is provided

• parameters_values – (default: None) list of numerical values of the parameters present in the system defining the curve, to be substituted in the equations before integration

• verbose – (default: False) prints information about the computation in progress

• **control_param – extra control parameters to be passed to the chosen solver; see the example with rtol and atol below

OUTPUT:

• list of the numerical points of the computed solution

EXAMPLES:

Computing a numerical solution:

```python
sage: M = Manifold(3, 'M')
sage: X.<x1,x2,x3> = M.chart()
sage: [t, B_0, m, q, L, T] = var('t B_0 m q L T')
sage: B = B_0*t/T*exp(-(x1^2 + x2^2)/L^2)
sage: D = X.symbolic_velocities()
sage: eqns = [q*B/m*D[1], -q*B/m*D[0], 0]
sage: p = M.point((0,0,0), name='p')
sage: Tp = M.tangent_space(p)
sage: v = Tp((1,0,1))
sage: c = M.integrated_curve(eqns, D, (t, 0,5), v, name='c')
sage: sol = c.solve(parameters_values={B_0:1, m: 1, q: 1, L: 10, T: 1},
                 verbose=True)
Performing numerical integration with method 'odeint'...
Resulting list of points will be associated with the key 'odeint' by default.
Numerical integration completed.
Checking all points are in the chart domain...
All points are in the chart domain.
The resulting list of points was associated with the key 'odeint' (if this key already referred to a former numerical solution, such a solution was erased).

The first 3 points of the solution, in the form [t, x1, x2, x3]:

```
Specifying the relative and absolute error tolerance parameters to be used in `desolve_odeint()`:

```python
sage: sol = c.solve(parameters_values={B_0:1, m:1, q:1, L:10, T:1},
                   rtol=1e-12, atol=1e-12)
```

Using a numerical method different from the default one:

```python
sage: sol = c.solve(parameters_values={B_0:1, m:1, q:1, L:10, T:1},
                   method='rk8pd')
```

`solve_across_charts` *(charts=None, step=None, solution_key=None, parameters_values=None, verbose=False, **control_param)*

Integrate the curve numerically over the domain of integration, with the ability to switch chart mid-integration.

The only supported solver is `scipy.integrate.ode`, because it supports basic event handling, needed to detect when the curve is reaching the frontier of the chart. This is an adaptive step solver. So the `step` is not the step of integration but instead the step used to peak at the current chart, and switch if needed.

**INPUT:**

- `step` – (default: None) step of chart checking; default value is a hundredth of the domain of integration if none is provided. If your curve can’t find a new frame on exiting the current frame, consider reducing this parameter.

- `charts` – (default: None) list of chart allowed. The integration stops once it leaves those charts. By default the whole atlas is taken (only the top-charts).

- `solution_key` – (default: None) key which the resulting numerical solution will be associated to; a default value is given if none is provided

- `parameters_values` – (default: None) list of numerical values of the parameters present in the system defining the curve, to be substituted in the equations before integration

- `verbose` – (default: False) prints information about the computation in progress

- `**control_param` – extra control parameters to be passed to the solver

**OUTPUT:**

- list of the numerical points of the computed solution

**EXAMPLES:**

Let us use `solve_across_charts()` to integrate a geodesic of the Euclidean plane (a straight line) in polar coordinates.

In pure polar coordinates \((r, \theta)\), artefacts can appear near the origin because of the fast variation of \(\theta\), resulting in the direction of the geodesic being different before and after getting close to the origin.

The solution to this problem is to switch to Cartesian coordinates near \((0, 0)\) to avoid any singularity.

First let’s declare the plane as a 2-dimensional manifold, with two charts \(P\) en \(C\) (for “Polar” and “Cartesian”) and their transition maps:

```python
sage: M = Manifold(2, 'M', structure='Riemannian')
sage: C.<x,y> = M.chart(coord_restrictions=lambda x,y: x**2+y**2 < 3**2)
sage: P.<r,th> = M.chart(coord_restrictions=lambda r, th: r > 2)
sage: P_to_C = P.transition_map(C,(r*cos(th), r*sin(th)))
sage: C_to_P = C.transition_map(P,(sqrt(x**2+y**2), atan2(y,x)))
```
Here we added restrictions on those charts, to avoid any singularity. The intersection is the donut region $2 < r < 3$.

We still have to define the metric. This is done in the Cartesian frame. The metric in the polar frame is computed automatically:

```
sage: g = M.metric()
sage: g[0,0,C]=1
sage: g[1,1,C]=1
sage: g[P.frame(), :, P]
[ 1 0]
[ 0 r^2]
```

To visualize our manifold, let’s declare a mapping between every chart and the Cartesian chart, and then plot each chart in term of this mapping:

```
sage: phi = M.diff_map(M, {(C,C): [x, y], (P,C): [r *cos(th), r *sin(th)]})
sage: fig = P.plot(number_values=9, chart=C, mapping=phi, #
˓→needs sage.plot
....:     color='grey', ranges={r:(2, 6), th:(0,2*pi)})
sage: fig += C.plot(number_values=13, chart=C, mapping=phi, #
˓→needs sage.plot
....:     color='grey', ranges={x:(-3, 3), y:(-3, 3)})
```

There is a clear non-empty intersection between the two charts. This is the key point to successfully switch chart during the integration. Indeed, at least 2 points must fall in the intersection.

**Geodesic integration**

Let’s define the time as $t$, the initial point as $p$, and the initial velocity vector as $v$ (define as a member of the tangent space $T_p$). The chosen geodesic should enter the central region from the left and leave it to the right:

```
sage: t = var('t')
sage: p = M((5,pi+0.3), P)
sage: Tp = M.tangent_space(p)
sage: v = Tp((-1,-0.03), P .frame().at(p))
```

While creating the integrated geodesic, we need to specify the optional argument `across_chart=True`, to prepare the compiled version of the changes of charts:

```
sage: c = M.integrated_geodesic(g, (t, 0, 10), v, across_charts=True)
```

The integration is done as usual, but using the method `solve_across_charts()` instead of `solve()`. This forces the use of scipy.integrate.ode as the solver, because of event handling support.

The argument `verbose=True` will cause the solver to write a small message each time it is switching chart:

```
sage: sol = c.solve_across_charts(step=0.1, verbose=True)
```

Performing numerical integration with method 'ode'.
Integration will take place on the whole manifold domain.
Resulting list of points will be associated with the key 'ode_multichart' by default.

Exiting chart, trying to switch to another chart.
New chart found. Resuming integration.
Exiting chart, trying to switch to another chart.

(continues on next page)
As expected, two changes of chart occur.

The returned solution is a list of pairs \((chart, \text{solution})\), where each solution is given on a unique chart, and the last point of a solution is the first of the next.

The following code prints the corresponding charts:

```python
sage: for chart, solution in sol:
    print(chart)
Chart (M, (r, th))
Chart (M, (x, y))
Chart (M, (r, th))
```

The interpolation is done as usual:

```python
sage: interp = c.interpolate()
```

To plot the result, you must first be sure that the mapping encompasses all the chart, which is the case here. You must also specify \(\text{across\_charts=True}\) in order to call \(\text{plot\_integrated()}\) again on each part. Finally, \(\text{color}\) can be a list, which will be cycled through:

```python
sage: fig += c.plot_integrated(mapping=phi, color = ["green","red"], #-- needs sage.plot
                           thickness=3, plot_points = 100, across_charts = True)
```

\[\text{solve\_analytical}\] \((\text{verbose=False})\)

Solve the differential system defining \texttt{self} analytically.

Solve analytically the differential system defining a curve using Maxima via Sage solver \texttt{desolve\_system}. In case of success, the analytical expressions are added to the dictionary of expressions representing the curve. Pay attention to the fact that \texttt{desolve\_system} only considers initial conditions given at an initial parameter value equal to zero, although the parameter range may not contain zero. Yet, assuming that it does, values of the coordinates functions at such zero initial parameter value are denoted by the name of the coordinate function followed by the string "\_0".

\textbf{OUTPUT:}

- list of the analytical expressions of the coordinate functions (when the differential system could be solved analytically), or boolean \texttt{False} (in case the differential system could not be solved analytically)

\textbf{EXAMPLES:}

Analytical expression of the trajectory of a charged particle in a uniform, stationary magnetic field:

```python
sage: M = Manifold(3, 'M')
sage: X.<x1,x2,x3> = M.chart()
sage: [t, B_0, m, q] = var('t B_0 m q')
sage: D = X.symbolic_velocities()
sage: eqns = [q*B_0/m*D[1], -q*B_0/m*D[0], 0]
sage: p = M.point((0,0,0), name='p')
sage: Tp = M.tangent_space(p)
sage: v = Tp((1,0,1))
```
2.5. Differentiable Maps and Curves
Manifolds, Release 10.3

(continued from previous page)

\begin{verbatim}
\texttt{sage: c = M.integrated_curve(eqns, D, (t,0,5), v, name='c')
\texttt{sage: sys = c.system(verbose=True)}
\end{verbatim}
Curve c in the 3-dimensional differentiable manifold M integrated over the Real interval (0, 5) as a solution to the following system, written with respect to Chart (M, (x1, x2, x3)):

Initial point: Point p on the 3-dimensional differentiable manifold M with coordinates [0, 0, 0] with respect to Chart (M, (x1, x2, x3))
Initial tangent vector: Tangent vector at Point p on the 3-dimensional differentiable manifold M with components [1, 0, 1] with respect to Chart (M, (x1, x2, x3))

d(x1)/dt = Dx1
d(x2)/dt = Dx2
d(x3)/dt = Dx3
d(Dx1)/dt = B_0*Dx2*q/m
d(Dx2)/dt = -B_0*Dx1*q/m
d(Dx3)/dt = 0

\texttt{sage: sol = c.solve_analytical()}
\texttt{sage: c.expr()}
((B_0*q*x1_0 - Dx2_0*m*cos(B_0*q*t/m) + Dx1_0*m*sin(B_0*q*t/m) + Dx2_0*m)/(B_0*q),
(B_0*q*x2_0 + Dx1_0*m*cos(B_0*q*t/m) + Dx2_0*m*sin(B_0*q*t/m) - Dx1_0*m)/(B_0*q),
Dx3_0*t + x3_0)

\texttt{system (verbose=False)}

Provide a detailed description of the system defining the curve and return the system defining it: chart, equations and initial conditions.

INPUT:

• \texttt{verbose} – (default: False) prints a detailed description of the curve

OUTPUT:

• list containing
  – the equations
  – the initial conditions
  – the chart

EXAMPLES:

System defining an integrated curve:

\begin{verbatim}
\texttt{sage: M = Manifold(3, 'M')
\texttt{sage: X.<x1,x2,x3> = M.chart()}
\texttt{sage: [t, B_0, m, q, L, T] = var('t B_0 m q L T')
\texttt{sage: B = B_0*t/T*exp(-(x1^2 + x2^2)/L^2)
\texttt{sage: D = X.symbolic_velocities()}
\texttt{sage: eqns = [q*B/m*D[1], -q*B/m*D[0], 0]
\texttt{sage: p = M.point((0,0,0), name='p')
\texttt{sage:Tp = M.tangent_space(p)
\end{verbatim}

(continues on next page)
sage: v = Tp((1,0,1))
sage: c = M.integrated_curve(eqns, D, (t,0,5), v, name='c')
sage: sys = c.system(allowverbose=True)
Curve c in the 3-dimensional differentiable manifold M integrated over the Real interval (0, 5) as a solution to the following system, written with respect to Chart (M, (x1, x2, x3)):

Initial point: Point p on the 3-dimensional differentiable manifold M with coordinates [0, 0, 0] with respect to Chart (M, (x1, x2, x3))
Initial tangent vector: Tangent vector at Point p on the 3-dimensional differentiable manifold M with components [1, 0, 1] with respect to Chart (M, (x1, x2, x3))

d(x1)/dt = Dx1
d(x2)/dt = Dx2
d(x3)/dt = Dx3
d(Dx1)/dt = B_0*Dx2*q*t*e^(-(x1^2 + x2^2)/L^2)/(T*m)
d(Dx2)/dt = -B_0*Dx1*q*t*e^(-(x1^2 + x2^2)/L^2)/(T*m)
d(Dx3)/dt = 0

sage: sys_mute = c.system()
sage: sys_mute == sys
True

tangent_vector_eval_at (t, interpolation_key=None, verbose=False)
Return the vector tangent to self at the given curve parameter with components evaluated from the given interpolation.

INPUT:

- t – curve parameter value at which the tangent vector is evaluated
- interpolation_key – (default: None) key which the interpolation requested to compute the tangent vector is associated to; a default value is chosen if none is provided
- verbose – (default: False) prints information about the interpolation used

OUTPUT:

- TangentVector tangent vector with numerical components

EXAMPLES:

Evaluating a vector tangent to the curve:

sage: M = Manifold(3, 'M')
sage: X.<x1,x2,x3> = M.chart()
sage: [t, B_0, m, q, L, T] = var('t B_0 m q L T')
sage: B = B_0*t/T*exp(-(x1^2 + x2^2)/L^2)
sage: D = X.symbolic_velocities()
sage: eqns = [q*B/m*D[1], -q*B/m*D[0], 0]
sage: p = M.point((0,0,0), name='p')
sage: Tp = M.tangent_space(p)
sage: v = Tp((1,0,1))
sage: c = M.integrated_curve(eqns, D, (t,0,5), v, name='c')
sage: # needs scipy
```python
sage: sol = c.solve(method='odeint',
....:     solution_key='sol_T1',
....:     parameters_values=(B_0:1, m:1, q:1, L:10, T:1))
....:
sage: interp = c.interpolate(method='cubic_spline',
....:     solution_key='sol_T1',
....:     interpolation_key='interp_T1')

sage: tg_vec = c.tangent_vector_eval_at(1.22, verbose=True)
Evaluating tangent vector components from the interpolation associated with the key 'interp_T1' by default...

sage: tg_vec
Tangent vector at Point on the 3-dimensional differentiable manifold M
sage: tg_vec[:]
# abs tol 1e-12
[0.7392640422917979, -0.6734182509826023, 1.0]

sage: tg_vec_mute = c.tangent_vector_eval_at(1.22, interpolation_key='interp_T1')

sage: tg_vec_mute == tg_vec
True
```

class `sage.manifolds.differentiable.integrated_curve.IntegratedGeodesic`

Geodesic on the manifold with respect to a given metric.

**INPUT:**

- `parent` — `IntegratedGeodesicSet` the set of curves $\text{Hom}_{\text{geodesic}}(I, M)$ to which the curve belongs
- `metric` — `PseudoRiemannianMetric` metric with respect to which the curve is a geodesic
- `curve_parameter` — symbolic expression to be used as the parameter of the curve (the equations defining an instance of `IntegratedGeodesic` are such that $t$ will actually be an affine parameter of the curve);
- `initial_tangent_vector` — `TangentVector` initial tangent vector of the curve
- `chart` — (default: `None`) chart on the manifold in terms of which the equations are expressed; 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

**EXAMPLES:**
Geodesics of the unit 2-sphere \( S^2 \). Start with declaring the standard polar coordinates \( (\theta, \phi) \) on \( S^2 \) and the corresponding coordinate frame \( (e_\theta, e_\phi) \):

```python
sage: S2 = Manifold(2, 'S^2', structure='Riemannian', start_index=1)
sage: polar.<th,ph>=S2.chart('th ph')
sage: epolar = polar.frame()
```

Set the standard round metric:

```python
sage: g = S2.metric()
sage: g[1,1], g[2,2] = 1, (sin(th))^2
```

Set generic initial conditions for the geodesics to compute:

```python
sage: [th0, ph0, v_th0, v_ph0] = var(th0 ph0 v_th0 v_ph0)
sage: p = S2.point((th0, ph0), name='p')
sage: Tp = S2.tangent_space(p)
sage: v = Tp((v_th0, v_ph0), basis=epolar.at(p))
```

Declare the corresponding integrated geodesic and display the differential system it satisfies:

```python
sage: [t, tmin, tmax] = var('t tmin tmax')
sage: c = S2.integrated_geodesic(g, (t, tmin, tmax), v,
....: chart=polar, name='c')
sage: sys = c.system(verbose=True)
```

Set a dictionary providing the parameter range and the initial conditions for various geodesics:

```python
sage: dict_params={'equat':{tmin:0,tmax:3,th0:pi/2,ph0:0.1,v_th0:0,v_ph0:1},
....: 'longi':{tmin:0,tmax:3,th0:0.1,ph0:0.1,v_th0:0,ph0:0},
....: 'angle':{tmin:0,tmax:3,th0:pi/4,ph0:0.1,v_th0:0,ph0:0}}
```

Use \( \mathbb{R}^3 \) as the codomain of the standard map embedding \( (S^2, (\theta, \phi)) \) in the 3-dimensional Euclidean space:

```python
sage: R3 = Manifold(3, 'R3', start_index=1)
sage: cart.<X,Y,Z> = R3.chart()
sage: euclid_embedding = S2.diff_map(R3,
....: (polar, cart):[sin(th)*cos(ph),sin(th)*sin(ph),cos(th)])
```

Solve, interpolate and prepare the plot for the solutions corresponding to the three initial conditions previously set:
Plot the resulting geodesics on the grid of polar coordinates lines on $S^2$ and check that these are great circles:

```sage
# needs scipy sage.plot
graph3D_embedded_polar_coords = polar.plot(chart=cart,
    mapping=euclid_embedding,
    number_values=15, color='yellow')
```

<table>
<thead>
<tr>
<th>Graphics3d Object</th>
<th>Graphics3d Object</th>
</tr>
</thead>
</table>

Return the system defining the geodesic: chart, equations and initial conditions.

```sage
system(verbos=False)
```
INPUT:

- verbose – (default: False) prints a detailed description of the curve

OUTPUT:

- list containing
  - the equations
  - the initial equations
  - the chart

EXAMPLES:

System defining a geodesic:

```python
sage: S2 = Manifold(2, 'S^2', structure='Riemannian')
sage: X.<theta,phi> = S2.chart()
sage: t, A = var('t A')
sage: g = S2.metric()
sage: g[0,0] = A
sage: g[1,1] = A*sin(theta)^2
sage: p = S2.point((pi/2,0), name='p')
sage: Tp = S2.tangent_space(p)
sage: v = Tp((1/sqrt(2),1/sqrt(2)))
sage: c = S2.integrated_geodesic(g, (t, 0, pi), v, name='c')
sage: sys = c.system(verbose=True)
```

Geodesic c in the 2-dimensional Riemannian manifold S^2 equipped with Riemannian metric g on the 2-dimensional Riemannian manifold S^2, and integrated over the Real interval (0, pi) as a solution to the following geodesic equations, written with respect to Chart (S^2, (theta, phi)):

Initial point: Point p on the 2-dimensional Riemannian manifold S^2 with coordinates [1/2*pi, 0] with respect to Chart (S^2, (theta, phi))

Initial tangent vector: Tangent vector at Point p on the 2-dimensional Riemannian manifold S^2 with components [1/2*sqrt(2), 1/2*sqrt(2)] with respect to Chart (S^2, (theta, phi))

```python
d(theta)/dt = Dtheta
d(phi)/dt = Dphi
d(Dtheta)/dt = Dphi^2*cos(theta)*sin(theta)
d(Dphi)/dt = -2*Dphi*Dtheta*cos(theta)/sin(theta)
```

```python
sage: sys_bis = c.system()
sage: sys_bis == sys
True
```
2.6 Tangent Spaces

2.6.1 Tangent Spaces

The class `TangentSpace` implements tangent vector spaces to a differentiable manifold.

AUTHORS:

- Travis Scrimshaw (2016): review tweaks

REFERENCES:

- Chap. 3 of [Lee2013]

```python
class sage.manifolds.differentiable.tangent_space.TangentSpace(point: ManifoldPoint, base_ring=None)
```

Bases: `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:

```python
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 $K = \mathbb{R}$ here):

```python
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:

```python
sage: Tp.bases()
[Basis (∂/∂x,∂/∂y) on the Tangent space at Point p on the 2-dimensional differentiable manifold M]
sage: M.frames()
[Coordinate frame (M, (∂/∂x,∂/∂y))]
```

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 (∂/∂u,∂/∂v) on the Tangent space at Point p on the 2-dimensional differentiable manifold M
sage: Tp.bases()
[(Basis (∂/∂x,∂/∂y) on the Tangent space at Point p on the 2-dimensional differentiable manifold M,
  Basis (∂/∂u,∂/∂v) on the Tangent space at Point p on the 2-dimensional differentiable manifold M)]

All the bases defined on $T_p$ 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()
∂/∂x + 2 ∂/∂y
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:
Actually \( p_1 \) and \( p \) share the same tangent space because they compare equal:

```python
sage: p1 == p
True
```

The tangent-space uniqueness holds even if the points are created in different coordinate systems:

```python
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
```

An isomorphism of the tangent space with an inner product space with distinguished basis:

```python
sage: g = M.metric('g')
sage: g[:] = ((1, 0), (0, 1))
sage: Q_Tp_xy = g[c_xy.frame(), :](p.coordinates(c_xy)); Q_Tp_xy
[1 0]
[0 1]
sage: W_Tp_xy = VectorSpace(SR, 2, inner_product_matrix=Q_Tp_xy)
sage: phi_Tp_xy = Tp.isomorphism_with_fixed_basis(Tp.bases()[0], codomain=W_Tp_xy)
Generic morphism:
From: Tangent space at Point p on the 2-dimensional differentiable manifold M
To: Ambient quadratic space of dimension 2 over Symbolic Ring
Inner product matrix:
[1 0]
[0 1]
sage: Q_Tp_uv = g[c_uv.frame(), :](p.coordinates(c_uv)); Q_Tp_uv
[1/2 0]
[0 1/2]
sage: W_Tp_uv = VectorSpace(SR, 2, inner_product_matrix=Q_Tp_uv)
sage: phi_Tp_uv = Tp.isomorphism_with_fixed_basis(Tp.bases()[1], codomain=W_Tp_uv)
Generic morphism:
From: Tangent space at Point p on the 2-dimensional differentiable manifold M
To: Ambient quadratic space of dimension 2 over Symbolic Ring
Inner product matrix:
[1/2 0]
[0 1/2]
sage: t1, t2 = Tp.tensor((1,0)), Tp.tensor((1,0))
(continues on next page)
sage: t1[:1] = (8, 15)
sage: t2[:1] = (47, 11)
sage: t1[Tp.bases()[0],:]
[8, 15]
sage: phi_Tp_xy(t1), phi_Tp_xy(t2)
((8, 15), (47, 11))
sage: phi_Tp_xy(t1).inner_product(phi_Tp_xy(t2))
541

sage: Tp_xy_to_uv = M.change_of_frame(c_xy.frame(), c_uv.frame()).at(p); Tp_xy_to_uv
Automorphism of the Tangent space at Point p on the 2-dimensional differentiable
→manifold M
sage: Tp.set_change_of_basis(Tp.bases()[0], Tp.bases()[1], Tp_xy_to_uv)
sage: t1[Tp.bases()[1],:]
[23, -7]
sage: phi_Tp_uv(t1), phi_Tp_uv(t2)
((23, -7), (58, 36))
sage: phi_Tp_uv(t1).inner_product(phi_Tp_uv(t2))
541

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

collection()

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():

2.6. Tangent Spaces
One can also use the global function `dim`:

```sage
sage: dim(Tp)
2
```

**dimension()**

Return the vector space dimension of `self`.

**EXAMPLES:**

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

One can also use the global function `dim`:

```sage
sage: dim(Tp)
2
```

---

### 2.6.2 Tangent Vectors

The class `TangentVector` implements tangent vectors to a differentiable manifold.

**AUTHORS:**

- Travis Scrimshaw (2016): review tweaks

**REFERENCES:**

- Chap. 3 of [Lee2013]

```python
class sage.manifolds.differentiable.tangent_vector.TangentVector(parent, name=None, latex_name=None)
```

**Bases:** `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`); symbol given to the vector
- `latex_name` – (default: `None`); LaTeX symbol to denote the vector; if `None`, `name` will be used
EXAMPLES:

A tangent vector \( v \) on a 2-dimensional manifold:

```python
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)
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 ∂/∂x + ∂/∂y
sage: v.parent()
Tangent space at Point p on the 2-dimensional differentiable manifold M
sage: v in Tp
True
```

Tangent vectors can also be constructed via the manifold method `tangent_vector()`:

```python
sage: v = M.tangent_vector(p, (-2, 1), name='v'); v
Tangent vector v at Point p on the 2-dimensional differentiable
  manifold M
sage: v.display()
v = -2 ∂/∂x + ∂/∂y
```

or via the method `at()` of vector fields:

```python
sage: vf = M.vector_field(x - 4*y/3, (x-y)^2, name='v')
sage: v = vf.at(p); v
Tangent vector v at Point p on the 2-dimensional differentiable
  manifold M
sage: v.display()
v = -2 ∂/∂x + ∂/∂y
```

By definition, a tangent vector at \( p \in M \) is a *derivation at* \( p \) on the space \( C^∞(M) \) of smooth scalar fields on \( M \). Indeed let us consider a generic scalar field \( f \):

```python
sage: f = M.scalar_field(function('F')(x,y), name='f')
sage: f.display()
f: M → ℝ
  (x, y) ↦ F(x, y)
```

The tangent vector \( v \) maps \( f \) to the real number \( v^i ∂F^i \big|_p \):

```python
sage: v(f)
-2*D[0](F)(2, 3) + D[1](F)(2, 3)
sage: vdf(x, y) = v[0]*diff(f.expr(), x) + v[1]*diff(f.expr(), y)
sage: X(p)
(2, 3)
sage: bool( v(f) == vdf(*X(p)) )
True
```

and if \( g \) is a second scalar field on \( M \):

```python
sage: g = M.scalar_field(function('G')(x,y), name='g')
```

then the product \( fg \) is also a scalar field on \( M \):
and we have the derivation law $v(fg) = v(f)g(p) + f(p)v(g)$:

```sage```
bool( v(f* g) == v(f)* g(p) + f(p)* v(g) )
```
```
True
```

See also:

- FiniteRankFreeModuleElement for more documentation.

**plot**

```python```
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 $\Phi : S^2 \to \mathbb{R}^3$).

**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:

```python
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):

```python
sage: v.plot() # needs sage.plot
Graphics object consisting of 2 graphics primitives
```

Plot atop of the chart grid:

```python
sage: X.plot() + v.plot() # needs sage.plot
Graphics object consisting of 20 graphics primitives
```

2.6. Tangent Spaces
sage: X.plot() + v.plot(color='green', scale=2, label='V')  # needs sage.plot
Graphics object consisting of 20 graphics primitives

sage: X.plot() + v.plot(print_label=False)  # needs sage.plot
Graphics object consisting of 19 graphics primitives

sage: X.plot() + v.plot(color='green', label_color='black',
                      fontsize=20, label_offset=0.2)  # needs sage.plot
....:
Graphics object consisting of 20 graphics primitives

sage: X.plot() + v.plot(linestyle=':', width=4, arrowsize=8, fontsize=20)  # needs sage.plot
....:
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()
2.6. Tangent Spaces
2.6. Tangent Spaces
v = (a + 1) \frac{\partial}{\partial x} - b^2 \frac{\partial}{\partial y}

\texttt{sage: X.plot() + v.plot(parameters={a: -2, b: 3})}
\texttt{# needs sage.plot}
\texttt{Graphics object consisting of 20 graphics primitives}

Special case of the zero vector:

\texttt{sage: v = Tp.zero() ; v}
\texttt{Tangent vector zero at Point p on the 2-dimensional differentiable manifold M}
\texttt{sage: X.plot() + v.plot()}
\texttt{# needs sage.plot}
\texttt{Graphics object consisting of 19 graphics primitives}

Vector tangent to a 4-dimensional manifold:

\texttt{sage: M = Manifold(4, 'M')}
\texttt{sage: X.<t,x,y,z> = M.chart()}
\texttt{sage: p = M((0,1,2,3), name='p')}
\texttt{sage: Tp = M.tangent_space(p)}
\texttt{sage: v = Tp((5,4,3,2), name='v') ; v}
\texttt{Tangent vector v at Point p on the 4-dimensional differentiable manifold M}

We cannot make a 4D plot directly:

\texttt{sage: v.plot()}
\texttt{# needs sage.plot}
\texttt{Traceback (most recent call last):}
\texttt{...}
\texttt{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 \texttt{ambient_coords}. For instance, for a 2-dimensional plot in terms of the coordinates \((x, y)\):

\texttt{sage: v.plot(ambient_coords=(x,y))}
\texttt{# needs sage.plot}
\texttt{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 \((t, x, y)\):

\texttt{sage: g = v.plot(ambient_coords=(t,x,z))}
\texttt{# needs sage.plot}
\texttt{sage: print(g)}
\texttt{# needs sage.plot}
\texttt{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:

\texttt{sage: (X.plot(ambient_coords=(t,x,z)) \# long time}
\texttt{# needs sage.plot}
\texttt{....: + v.plot(ambient_coords=(t,x,z),}
\texttt{....: label_offset=0.5, width=6))}
\texttt{Graphics3d Object}
2.6. Tangent Spaces
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: $S^2 \rightarrow R^3$
on U: $(th, ph) \mapsto (x, y, z) = (cos(ph)*sin(th), sin(ph)*sin(th), cos(th))$
```

```
sage: p = U.point((pi/4, 7*pi/4), name = 'p')
sage: v = XS.frame()[1].at(p) ; v  # the coordinate vector $\partial/\partial phi$ at $p$
Tangent vector $\partial/\partial phi$ at Point $p$ on the 2-dimensional differentiable manifold $S^2$
```

```
sage: graph_v = v.plot(mapping=F)  # needs sage.plot
sage: graph_S2 = XS.plot(chart=X3, mapping=F, number_values=9)  # long time,
                                    # needs sage.plot
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 $[\cdot, \cdot]$ (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:

- $\texttt{VectorFieldModule}$ for vector fields with values on a generic (in practice, not parallelizable) differentiable manifold $M$.
- $\texttt{VectorFieldFreeModule}$ for vector fields with values on a parallelizable manifold $M$.

AUTHORS:
- Travis Scrimshaw (2016): structure of Lie algebroid (github issue #20771)

REFERENCES:
- [KN1963]
- [Lee2013]
- [ONe1983]

```python
class sage.manifolds.differentiable.vectorfield_module.VectorFieldFreeModule(domain, dest_map=None)
```

Bases: $\texttt{FiniteRankFreeModule}$

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

$\Phi : U \rightarrow M$

the vector field module $\mathcal{X}(U, \Phi)$ is the set of all vector fields of the type

$v : U \rightarrow T M$

(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 $\texttt{DiffScalarFieldAlgebra}$). 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 $\mathbb{R}$).
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 \( \Phi : U \to M \) (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 \( \mathbb{R}^2 \):

```
sage: M = Manifold(2, 'R^2')
sage: cart.<x,y> = M.chart()  # Cartesian coordinates on \( \mathbb{R}^2 \)
sage: XM = M.vector_field_module() ; XM
Free module \( X(\mathbb{R}^2) \) of vector fields on the 2-dimensional differentiable manifold \( \mathbb{R}^2 \)
```

Since \( \mathbb{R}^2 \) is obviously parallelizable, \( XM \) is a free module:

```
sage: isinstance(XM, FiniteRankFreeModule)
True
```

Some elements:

```
sage: XM.an_element().display()
\frac{\partial}{\partial x} + 2 \frac{\partial}{\partial y}
sage: XM.zero().display()
zero = 0
sage: v = XM([-y,x]) ; v
Vector field on the 2-dimensional differentiable manifold \( \mathbb{R}^2 \)
```

An example of module of vector fields with a destination map \( \Phi \) different from the identity map, namely a mapping \( \Phi : I \to \mathbb{R}^2 \), where \( I \) is an open interval of \( \mathbb{R} \):

```
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
\( I \to \mathbb{R}^2 \)
```

(continues on next page)
The rank of the free module $\mathcal{X}(I, \Phi)$ is the dimension of the manifold $\mathbb{R}^2$, namely two:

```sage```
XIM.rank()
```
```
2
```

A basis of it is induced by the coordinate vector frame of $\mathbb{R}^2$:

```sage```
XIM.bases()
```
```
[Vector frame (I, (∂/∂x,∂/∂y)) with values on the 2-dimensional differentiable manifold R^2]
```

Some elements of this module:

```sage```
XIM.an_element().display()
```
```
2 ∂/∂x + 2 ∂/∂y
```

```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 ∂/∂x + t^2 ∂/∂y
```

The test suite is passed:

```sage```
TestSuite(XIM).run()
```

Let us introduce an open subset of $J \subset I$ and the vector field module corresponding to the restriction of $\Phi$ to it:

```sage```
J = I.open_subset(J, coord_def = {canon: t < pi})
XJM = J.vector_field_module(dest_map=Phi.restrict(J)); XJM
```
```
Free module X(J,Phi) of vector fields along the Open subset J of the 1-dimensional differentiable manifold I mapped into the 2-dimensional differentiable manifold R^2
```

We have then:

```sage```
XJM.default_basis()
```
```
Vector frame (J, (∂/∂x,∂/∂y)) with values on the 2-dimensional differentiable manifold R^2
```

```sage```
XJM.default_basis() is XIM.default_basis().restrict(J)
```
```
True
```

```sage```
v.restrict(J)
```
```
Vector field along the Open subset J of the 1-dimensional differentiable manifold I with values on the 2-dimensional differentiable manifold R^2
```

```sage```
v.restrict(J).display()
```
```
t ∂/∂x + t^2 ∂/∂y
```

Let us now consider the module of vector fields on the circle $S^1$; we start by constructing the $S^1$ manifold:
\[ S^1 \] 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 \( \frac{\partial}{\partial t} \) and \( \frac{\partial}{\partial u} \) coincide, as we can check explicitly:

\[
\text{sage: } c_t.frame()[0].display(c_u.frame().restrict(W))
\]
\[
\frac{\partial}{\partial t} = \frac{\partial}{\partial u}
\]

Therefore, we can extend \( \frac{\partial}{\partial t} \) to all \( V \) and hence to all \( S^1 \), to form a vector field on \( S^1 \) whose components w.r.t. both \( \frac{\partial}{\partial t} \) and \( \frac{\partial}{\partial u} \) are 1:

\[
\text{sage: } e = M.vector_frame('e')
\]
\[
\text{sage: } U.set_change_of_frame(e.restrict(U), c_t.frame(),
\text{.......: } U.tangent_identity_field())
\]
\[
\text{sage: } V.set_change_of_frame(e.restrict(V), c_u.frame(),
\text{.......: } V.tangent_identity_field())
\]

\[
\text{sage: } e[0].display(c_t.frame())
\]
\[
e_0 = \frac{\partial}{\partial t}
\]
\[
\text{sage: } e[0].display(c_u.frame())
\]
\[
e_0 = \frac{\partial}{\partial u}
\]

Equipped with the frame \( e \), the manifold \( S^1 \) is manifestly parallelizable:

\[
\text{sage: } M.is_manifestly_parallelizable()
\]
True

Consequently, the module of vector fields on \( S^1 \) is a free module:

\[
\text{sage: } XM = M.vector_field_module() ; XM
\]

Free module \( X(S^1) \) of vector fields on the 1-dimensional differentiable manifold \( S^1 \)

\[
\text{sage: } isinstance(XM, FiniteRankFreeModule)
\]
True

\[
\text{sage: } XM.category()
\]

Category of finite dimensional modules

\[
\text{over Algebra of differentiable scalar fields}
\]
on the 1-dimensional differentiable manifold \( S^1 \)

\[
\text{sage: } XM.base_ring() \text{ is } M.scalar_field_algebra()
\]
True

The zero element:

\[
\text{sage: } z = XM.zero() ; z
\]

Vector field zero on the 1-dimensional differentiable manifold \( S^1 \)

\[
\text{sage: } z.display()
\]
zero = 0

(continues on next page)
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, indices=None, latex_indices=None, symbol_dual=None, latex_symbol_dual=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` – (default: None) either a string, to be used as a common base for the symbols of the elements of the basis, or a tuple of strings, representing the individual symbols of the elements of the basis
- `latex_symbol` – (default: None) either a string, to be used as a common base for the LaTeX symbols of the elements of the basis, or a tuple of strings, representing the individual LaTeX symbols of the elements of the basis; if None, `symbol` is used in place of `latex_symbol`
- `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 $e(p) = \tilde{e}(\Phi(p))$
- `indices` – (default: None; used only if `symbol` is a single string) tuple of strings representing the indices labelling the elements of the basis; if None, the indices will be generated as integers within the range declared on `self`
- `latex_indices` – (default: None) tuple of strings representing the indices for the LaTeX symbols of the elements of the basis; if None, `indices` is used instead
- `symbol_dual` – (default: None) same as `symbol` but for the dual basis; if None, `symbol` must be a string and is used for the common base of the symbols of the elements of the dual basis
- `latex_symbol_dual` – (default: None) same as `latex_symbol` but for the dual basis

OUTPUT:

- a `VectorFrame` representing a basis on `self`

EXAMPLES:

```sage
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

$$\Phi : U \rightarrow M$$

such that this module is the set $\mathcal{X}(U, \Phi)$ of all vector fields of the type

$$v : U \rightarrow TM$$

(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 \mathfrak{X}(U, \Phi), returns the domain U of \Phi.

OUTPUT:
• a DifferentiableManifold representing the domain of the vector fields that belong to this mod-
ule

EXAMPLES:

sage: M = Manifold(2, 'M')
sage: X.<x,y> = 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 self is \mathfrak{X}(U, \Phi), the p-th exterior power of its dual is the set \Omega^p(U, \Phi) of p-forms
along U with values on \Phi(U). It is a free module over \mathcal{C}^k(U), the ring (algebra) of differentiable scalar fields
on U.

INPUT:
• p – non-negative integer

OUTPUT:
• for p = 0, the base ring, i.e. \mathcal{C}^k(U)
• for p \geq 1, a DiffFormFreeModule representing the module \Omega^p(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.dual_exterior_power(2)
Free module $\Omega^2(M)$ of 2-forms on the 2-dimensional differentiable manifold $M$
\begin{verbatim}
 sage: XM.dual_exterior_power(1)
 Free module $\Omega^1(M)$ 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
\end{verbatim}

See also:

$\texttt{DiffFormFreeModule}$ for more examples and documentation.

$\texttt{exterior_power}(p)$

Return the $p$-th exterior power of $\texttt{self}$.

If the vector field module $\texttt{self}$ is $X(U, \Phi)$, its $p$-th exterior power is the set $A^p(U, \Phi)$ of $p$-vector fields along $U$ with values on $\Phi(U)$. It is a free module over $C^k(U)$, the ring (algebra) of differentiable scalar fields on $U$.

INPUT:

- $p$ – non-negative integer

OUTPUT:

- for $p = 0$, the base ring, i.e. $C^k(U)$
- for $p = 1$, the vector field free module $\texttt{self}$, since $A^1(U, \Phi) = X(U, \Phi)$
- for $p \geq 2$, instance of $\texttt{MultivectorFreeModule}$ representing the module $A^p(U, \Phi)$

EXAMPLES:

\begin{verbatim}
 sage: M = Manifold(2, 'M')
 sage: X.<x,y> = M.chart()  # makes M parallelizable
 sage: XM = M.vector_field_module()
 sage: XM.exterior_power(2)
 Free module $\Lambda^2(M)$ of 2-vector fields on the 2-dimensional differentiable manifold $M$
 sage: XM.exterior_power(1)
 Free module $X(M)$ of vector fields on the 2-dimensional differentiable manifold $M$
 sage: XM.exterior_power(1) is XM
 True
 sage: XM.exterior_power(0)
 Algebra of differentiable scalar fields on the 2-dimensional differentiable manifold $M$
 sage: XM.exterior_power(0) is M.scalar_field_algebra()
 True
\end{verbatim}

See also:

$\texttt{MultivectorFreeModule}$ for more examples and documentation.
**general_linear_group()**

Return the general linear group of self.

If the vector field module is $\mathfrak{X}(U, \Phi)$, the *general linear group* is the group $\text{GL}(\mathfrak{X}(U, \Phi))$ of automorphisms of $\mathfrak{X}(U, \Phi)$. Note that an automorphism of $\mathfrak{X}(U, \Phi)$ can also be viewed as a *field* along $U$ of automorphisms of the tangent spaces of $V = \Phi(U)$.

**OUTPUT:**

- a `AutomorphismFieldParalGroup` representing $\text{GL}(\mathfrak{X}(U, \Phi))$

**EXAMPLES:**

```python
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:**

```python
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.
**poisson_tensor**(name=None, latex_name=None)

Construct a Poisson tensor on the current vector field module.

OUTPUT:

- instance of *PoissonTensorFieldParal*

**EXAMPLES:**

Standard Poisson tensor on $\mathbb{R}^2$:

```python
sage: M.<q, p> = EuclideanSpace(2)
sage: poisson = M.vector_field_module().poisson_tensor('varpi')
sage: poisson.set_comp()[1,2] = -1
sage: poisson.display()
varpi = -e_q∧e_p
```

**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 symmetric bilinear form
- latex_name – string (default: None); LaTeX symbol to denote the symmetric bilinear form; if None, the LaTeX symbol is set to name

OUTPUT:

- a *TensorFieldParal* of tensor type $(0, 2)$ and symmetric

**EXAMPLES:**

```python
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.

**symplectic_form**(name=None, latex_name=None)

Construct a symplectic form on the current vector field module.

OUTPUT:

- instance of *SymplecticFormParal*

**EXAMPLES:**

Standard symplectic form on $\mathbb{R}^2$:

```python
sage: M.<q, p> = EuclideanSpace(2)
sage: omega = M.vector_field_module().symplectic_form('omega', r'\omega')
sage: omega.set_comp()[1,2] = -1
sage: omega.display()
omega = -dq∧dp
```
**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
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) ∂/∂x⊗dx - y ∂/∂x⊗dy + x*y ∂/∂y⊗dx + (-y^2 + 2) ∂/∂y⊗dy
```

The same set of components transformed into a type-(0, 2) tensor field:

```sage
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, sym, antisym)*

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 (do-
main: Differentiable-Mani-
fold, dest_map: DiffMap
1 None
= None)

Bases: UniqueRepresentation, ReflexiveModule_base

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

$$\Phi : U \rightarrow M,$$

the vector field module $\mathcal{X}(U, \Phi)$ is the set of all vector fields of the type

$$v : U \rightarrow TM$$

(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)

$$[X, Y] = X \circ Y - Y \circ X$$

2.7. Vector Fields
over the scalar fields 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 $\mathfrak{X}(M, \text{Id}_M)$ by merely $\mathfrak{X}(M)$. Other common cases are $\Phi$ being an immersion and $\Phi$ being a curve in $M$ ($U$ is then an open interval of $\mathbb{R}$).

**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)),
sage: ....: intersection_name='W', restrictions1 = x^2+y^2!=0,
sage: ....: 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
```

$\mathfrak{X}(M)$ 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
```

$\mathfrak{X}(M)$ 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: issubclass(XU, FiniteRankFreeModule)
True
```

(continues on next page)
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_contravariant_tensor**(degree, name=None, latex_name=None)

Construct an alternating contravariant tensor on the vector field module self.

An alternating contravariant tensor on self is actually a multivector field along the differentiable manifold $U$ over which self is defined.

**INPUT:**

- degree – degree of the alternating contravariant tensor (i.e. its tensor rank)
- name – (default: None) string; name given to the alternating contravariant tensor
- latex_name – (default: None) string; LaTeX symbol to denote the alternating contravariant tensor; if none is provided, the LaTeX symbol is set to name

**OUTPUT:**

- instance of `MultivectorField`

**EXAMPLES:**

```
sage: M = Manifold(2, 'M')
sage: XM = M.vector_field_module()
sage: XM.alternating_contravariant_tensor(2, name='a')
2-vector field a on the 2-dimensional differentiable manifold M
```

An alternating contravariant tensor of degree 1 is simply a vector field:

```
sage: XM.alternating_contravariant_tensor(1, name='a')
Vector field a on the 2-dimensional differentiable manifold M
```
See also:

MultivectorField for more examples and documentation.

alternating_form\(\text{(degree, name=None, latex_name=None)}\)
Construct an alternating form on the vector field module \(\text{self}\).
An alternating form on \(\text{self}\) is actually a differential form along the differentiable manifold \(U\) over which \(\text{self}\) 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:

\begin{verbatim}
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
\end{verbatim}

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:

\begin{verbatim}
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
\end{verbatim}

automorphism\(\text{(name=None, latex_name=None)}\)
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')
XM = M.vector_field_module()
XM.automorphism()
Field of tangent-space automorphisms on the 2-dimensional differentiable manifold M
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

\[ v : U \longrightarrow TM \]

(\( TM \) 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')
XM = M.vector_field_module()
XM.destination_map()
Identity map Id_M of the 5-dimensional differentiable manifold M
U = Manifold(2, 'U')
Phi = U.diff_map(M, name='Phi')
XU = U.vector_field_module(dest_map=Phi)
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 $\mathfrak{X}(U, \Phi)$, 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 Omega^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 $\mathfrak{X}(U, \Phi)$, the $p$-th exterior power of its dual is the set $\Omega^p(U, \Phi)$ 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 $p = 0$, the base ring, i.e. $C^k(U)$
- for $p \geq 1$, instance of `DiffFormModule` representing the module $\Omega^p(U, \Phi)$

**EXAMPLES:**

```
sage: M = Manifold(2, 'M')
sage: XM = M.vector_field_module()
sage: XM.dual_exterior_power(2)
Module Omega^2(M) of 2-forms on the 2-dimensional differentiable manifold M
sage: XM.dual_exterior_power(1)
Module Omega^1(M) of 1-forms on the 2-dimensional differentiable manifold M
sage: XM.dual_exterior_power(1) is XM.dual()
True
```

(continues on next page)
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.

eexterior_power(<p>)

Return the $p$-th exterior power of `self`.

If the vector field module `self` is $\mathfrak{X}(U, \Phi)$, its $p$-th exterior power is the set $A^p(U, \Phi)$ of $p$-vector fields 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 $p = 0$, the base ring, i.e. $C^k(U)$
• for $p = 1$, the vector field module `self`, since $A^1(U, \Phi) = \mathfrak{X}(U, \Phi)$
• for $p \geq 2$, instance of `MultivectorModule` representing the module $A^p(U, \Phi)$

EXAMPLES:

```sage
sage: M = Manifold(2, 'M')
sage: XM = M.vector_field_module()
sage: XM.exterior_power(2)
Module $A^2(M)$ of 2-vector fields on the 2-dimensional differentiable manifold M
sage: XM.exterior_power(1)
Module $X(M)$ of vector fields on the 2-dimensional differentiable manifold M
sage: XM.exterior_power(1) is XM
True
sage: XM.exterior_power(0)
Algebra of differentiable scalar fields on the 2-dimensional differentiable manifold M
sage: XM.exterior_power(0) is M.scalar_field_algebra()
True
```

See also:

`MultivectorModule` for more examples and documentation.

general_linear_group()

Return the general linear group of `self`.

If the vector field module is $\mathfrak{X}(U, \Phi)$, the `general linear group` is the group $GL(\mathfrak{X}(U, \Phi))$ of automorphisms of $\mathfrak{X}(U, \Phi)$. Note that an automorphism of $\mathfrak{X}(U, \Phi)$ 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 $GL(\mathfrak{X}(U, \Phi))$
EXAMPLES:

```sage
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()**

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.

**OUTPUT:**

- instance of `AutomorphismField`

**EXAMPLES:**

Get the identity map on a vector field module:

```sage
sage: M = Manifold(2, 'M')
sage: XM = M.vector_field_module()
sage: Id = XM.identity_map(); Id
Field of tangent-space identity maps on the 2-dimensional
differentiable manifold M
```

If the identity should be renamed, one has to create a copy:

```sage
sage: Id.set_name('1')
Traceback (most recent call last):
...
ValueError: the name of an immutable element cannot be changed
sage: one = Id.copy('1'); one
Field of tangent-space automorphisms 1 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:**
manifolds = Manifold(2, 'M')
sage: XM = M.vector_field_module()
sage: XM.linear_form()
l-form on the 2-dimensional differentiable manifold M
sage: XM.linear_form(name='a')
l-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 metric (symmetric bilinear form) on the current vector field module.

A 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.

poisson_tensor (name=None, latex_name=None)

Construct a Poisson tensor on the current vector field module.

OUTPUT:

- instance of PoissonTensorField

EXAMPLES:

Poisson tensor on the 2-sphere:

sage: M = manifolds.Sphere(2, coordinates='stereographic')
sage: XM = M.vector_field_module()
sage: varpi = XM.poisson_tensor(name='varpi', latex_name=r'\varpi')
sage: varpi
2-vector field \( \varpi \) on the 2-sphere \( S^2 \) of radius 1 smoothly embedded in the Euclidean space \( E^3 \)

**symplectic_form** *(name=None, latex_name=None)*

Construct a symplectic form on the current vector field module.

**OUTPUT:**

- instance of \( \text{SymplecticForm} \)

**EXAMPLES:**

Symplectic form on the 2-sphere:

```plaintext
sage: M = manifolds.Sphere(2, coordinates='stereographic')
sage: XM = M.vector_field_module()
sage: omega = XM.symplectic_form(name='omega', latex_name=r'\omega')
sage: omega
Symplectic form \( \omega \) on the 2-sphere \( S^2 \) of radius 1 smoothly embedded in the Euclidean space \( E^3 \)
```

**tensor** *(*args, **kwds)*

Construct a tensor field on the domain of \( \text{self} \) or a tensor product of \( \text{self} \) with other modules.

If \( \text{args} \) consist of other parents, just delegate to \( \text{tensor_product}() \).

Otherwise, construct a tensor (i.e., a tensor field on the domain of the vector field module) from the following input.

**INPUT:**

- \( \text{tensor_type} - \) pair \((k,l)\) with \( k \) being the contravariant rank and \( l \) the covariant rank
- \( \text{name} - \) (string; default: None) name given to the tensor
- \( \text{latex_name} - \) (string; default: None) LaTeX symbol to denote the tensor; if none is provided, the LaTeX symbol is set to \( \text{name} \)
- \( \text{sym} - \) (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:
  - \( \text{sym}=(0,1) \) for a symmetry between the 1st and 2nd arguments
  - \( \text{sym}=[(0,2),(1,3,4)] \) for a symmetry between the 1st and 3rd arguments and a symmetry between the 2nd, 4th and 5th arguments
- \( \text{antisym} - \) (default: None) antisymmetry or list of antisymmetries among the arguments, with the same convention as for \( \text{sym} \)
- \( \text{specific_type} - \) (default: None) specific subclass of \( \text{TensorField} \) for the output

**OUTPUT:**

- instance of \( \text{TensorField} \) representing the tensor defined on the vector field module with the provided characteristics

**EXAMPLES:**

```plaintext
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 $M$

Delegation to `tensor_product()`:

```sage```
M = Manifold(2, 'M')
XM = M.vector_field_module()
XM.tensor(XM)
```
Module $T^{(2,0)}(M)$ of type-$(2,0)$ tensors fields on the 2-dimensional differentiable manifold $M$

```sage```
XM.tensor(XM, XM.dual(), XM)
```
Module $T^{(3,1)}(M)$ of type-$(3,1)$ tensors fields on the 2-dimensional differentiable manifold $M$

```sage```
XM.tensor(XM).tensor(XM.dual().tensor(XM.dual()))
```
Traceback (most recent call last):
...
AttributeError: 'TensorFieldModule_with_category' object has no attribute '_basis_sym'...

See also:

`TensorField` for more examples and documentation.

tensor_module\((k, l, \text{sym}, \text{antisym})\)

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')
XM = M.vector_field_module()
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:
See `TensorFieldModule` for more examples and documentation.

### zero()

Return the zero of `self`.

**EXAMPLES:**

```python
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 $\mathbb{R}$).

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:**

- Marco Mancini (2015): parallelization of vector field plots
- Travis Scrimshaw (2016): review tweaks
- Eric Gourgoulhon (2017): vector fields inherit from multivector fields
- Eric Gourgoulhon (2018): dot and cross products, operators norm and curl

**REFERENCES:**

- [KN1963]
- [Lee2013]
- [ONe1983]
- [BG1988]
class sage.manifolds.differentiable.vectorfield.VectorField(vector_field_module, name=None, latex_name=None)

Bases: MultivectorField

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

$$\Phi : U \rightarrow M,$$

a vector field along $U$ with values on $M$ is a differentiable map

$$v : U \rightarrow TM$$

($TM$ 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 $\mathbb{R}$).

Note: If $M$ is parallelizable, then VectorFieldParal must be used instead.

INPUT:

- vector_field_module – module $\mathfrak{X}(U, \Phi)$ 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:

```python
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()
   ....: 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: eU = c_xy.frame() ; eV = c_tu.frame()
sage: c_tuW = c_tu.restrict(W) ; eVW = c_tuW.frame()
sage: v = M.vector_field(name='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 $eU$:

```python
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 $V$:

```python
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:

```python
sage: v.display(eU)
\[v = -y \frac{\partial}{\partial x} + (x + 1) \frac{\partial}{\partial y}\]
sage: v.display(eV)
\[v = (u + 1) \frac{\partial}{\partial t} + (-t - 1) \frac{\partial}{\partial u}\]
```

The vector field acting on scalar fields:

```python
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 \rightarrow \mathbb{R}
on \(U\): \((x, y) \mapsto 2x^2 - 2y^2 + 2x + 2y\)
on \(V\): \((t, u) \mapsto 2t^2u + 2t\)
```

Some checks:

```python
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:

```python
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
```

```
```
**bracket** (*other*)

Return the Lie bracket \([self, other]\).  

**INPUT:**  
- *other* – a *VectorField*  

**OUTPUT:**  
- the *VectorField* \([self, other]\)

**EXAMPLES:**

```
sage: M = Manifold(3, 'M')
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) ∂/∂y + (-y - z) ∂/∂z
```

Some checks:

```
sage: vw == - w.bracket(v)
True
sage: f = M.scalar_field({X: x+y*z})
sage: vw(f) == v(w(f)) - w(v(f))
True
sage: vw == w.lie_derivative(v)
True
```

**cross** (*other, metric=None*)

Return the cross product of *self* with another vector field (with respect to a given metric), assuming that the domain of *self* is 3-dimensional.

If *self* is a vector field \(u\) on a 3-dimensional differentiable orientable manifold \(M\) and *other* is a vector field \(v\) on \(M\), the **cross product** (also called **vector product**) of \(u\) by \(v\) with respect to a pseudo-Riemannian metric \(g\) on \(M\) is the vector field \(w = u \times v\) defined by

\[
w^i = \epsilon^i_{jk} u^j v^k = g^{il} \epsilon_{ljk} u^j v^k
\]

where \(\epsilon\) is the volume 3-form (Levi-Civita tensor) of \(g\) (cf. `volume_form()`)  

**Note:** The method **cross_product** is meaningful only if for vector fields on a 3-dimensional manifold.

**INPUT:**  
- *other* – a vector field, defined on the same domain as *self*  
- *metric* – (default: None) the pseudo-Riemannian metric \(g\) involved in the definition of the cross product; if none is provided, the domain of *self* is supposed to be endowed with a default metric (i.e. is supposed to be pseudo-Riemannian manifold, see `PseudoRiemannianManifold`) and the latter is used to define the cross product  

**OUTPUT:**  
- instance of *VectorField* representing the cross product of *self* by *other*.  

---

2.7. Vector Fields  

555
EXAMPLES:

Cross product in the Euclidean 3-space:

```sage
M.<x,y,z> = EuclideanSpace()
sage: u = M.vector_field(-y, x, 0, name='u')
sage: v = M.vector_field(x, y, 0, name='v')
sage: w = u.cross_product(v);
Vector field u x v on the Euclidean space E^3
sage: w.display()
u x v = (-x^2 - y^2) e_z
```

A shortcut alias of `cross_product` is `cross`:

```sage
sage: u.cross(v) == w
True
```

The cross product of a vector field with itself is zero:

```sage
sage: u.cross_product(u).display()
u x u = 0
```

Cross product with respect to a metric that is not the default one:

```sage
h = M.riemannian_metric('h')
sage: h[1,1], h[2,2], h[3,3] = 1/(1+y^2), 1/(1+z^2), 1/(1+x^2)
sage: w = u.cross_product(v, metric=h);
Vector field on the Euclidean space E^3
sage: w.display()
-(x^2 + y^2)*sqrt(x^2 + 1)/(sqrt(y^2 + 1)*sqrt(z^2 + 1)) e_z
```

Cross product of two vector fields along a curve (arc of a helix):

```sage
R.<t> = manifolds.RealLine()
sage: C = M.curve((cos(t), sin(t), t), (t, 0, 2*pi), name='C')
sage: u = C.tangent_vector_field()
sage: u.display()
C' = -sin(t) e_x + cos(t) e_y + e_z
sage: I = C.domain(); I
Real interval (0, 2*pi)
sage: v = I.vector_field(-cos(t), sin(t), 0, dest_map=C)
sage: v.display()
-cos(t) e_x + sin(t) e_y
sage: w = u.cross_product(v);
Vector field along the Real interval (0, 2*pi) with values on the
Euclidean space E^3
sage: w.display()
-sin(t) e_x - cos(t) e_y + (2*cos(t)^2 - 1) e_z
```

Cross product between a vector field along the curve and a vector field on the ambient Euclidean space:

```sage
e_x = M.cartesian_frame()[1]
sage: w = u.cross_product(e_x);
Vector field C' x e_x along the Real interval (0, 2*pi) with values
on the Euclidean space E^3
sage: w.display()
C' x e_x = e_y - cos(t) e_z
```
cross_product (other, metric=None)

Return the cross product of self with another vector field (with respect to a given metric), assuming that the domain of self is 3-dimensional.

If self is a vector field $u$ on a 3-dimensional differentiable orientable manifold $M$ and other is a vector field $v$ on $M$, the cross product (also called vector product) of $u$ by $v$ with respect to a pseudo-Riemannian metric $g$ on $M$ is the vector field $w = u \times v$ defined by

$$w^i = \epsilon^i_{jk} u^j v^k = g^{il} \epsilon_{ljk} u^j v^k$$

where $\epsilon$ is the volume 3-form (Levi-Civita tensor) of $g$ (cf. volume_form()).

Note: The method cross_product is meaningful only if for vector fields on a 3-dimensional manifold.

INPUT:

- other – a vector field, defined on the same domain as self
- metric – (default: None) the pseudo-Riemannian metric $g$ involved in the definition of the cross product; if none is provided, the domain of self is supposed to be endowed with a default metric (i.e. is supposed to be pseudo-Riemannian manifold, see PseudoRiemannianManifold) and the latter is used to define the cross product

OUTPUT:

- instance of VectorField representing the cross product of self by other.

EXAMPLES:

Cross product in the Euclidean 3-space:

```python
sage: M.<x,y,z> = EuclideanSpace()
sage: u = M.vector_field(-y, x, 0, name='u')
sage: v = M.vector_field(x, y, 0, name='v')
sage: w = u.cross_product(v); w
Vector field u x v on the Euclidean space E^3
sage: w.display()
u x v = (-x^2 - y^2) e_z
```

A shortcut alias of cross_product is cross:

```python
sage: u.cross(v) == w
True
```

The cross product of a vector field with itself is zero:

```python
sage: u.cross_product(u).display()
u x u = 0
```

Cross product with respect to a metric that is not the default one:

```python
sage: h = M.riemannian_metric('h')
sage: h[1,1], h[2,2], h[3,3] = 1/(1+y^2), 1/(1+z^2), 1/(1+x^2)
sage: w = u.cross_product(v, metric=h); w
Vector field on the Euclidean space E^3
sage: w.display()
-(x^2 + y^2)*sqrt(x^2 + 1)/(sqrt(y^2 + 1)*sqrt(z^2 + 1)) e_z
```

Cross product of two vector fields along a curve (arc of a helix):
Cross product between a vector field along the curve and a vector field on the ambient Euclidean space:

```
sage: e_x = M.cartesian_frame()[1]
sage: w = u.cross_product(e_x); w
Vector field C' x e_x along the Real interval (0, 2*pi) with values on the Euclidean space E^3
sage: w.display()
(-sin(t) e_x - cos(t) e_y + (2*cos(t)^2 - 1) e_z)
```

**curl** *(metric=None)*

Return the curl of self with respect to a given metric, assuming that the domain of self is 3-dimensional.

If self is a vector field v on a 3-dimensional differentiable orientable manifold M, the curl of v with respect to a metric g on M is the vector field defined by

\[
\text{curl } v = (\ast (d v^\flat))^g
\]

where v^\flat is the 1-form associated to v by the metric g (see down()), \ast (d v^\flat) is the Hodge dual with respect to g of the 2-form d v^\flat (exterior derivative of v^\flat) (see hodge_dual()) and (\ast (d v^\flat))^g is corresponding vector field by g-duality (see up()).

An alternative expression of the curl is

\[
(curl \ v)^i = \epsilon^{ijk} \nabla_j v_k
\]

where \( \nabla \) is the Levi-Civita connection of g (cf. LeviCivitaConnection) and \( \epsilon \) the volume 3-form (Levi-Civita tensor) of g (cf. volume_form())

**Note:** The method curl is meaningful only if self is a vector field on a 3-dimensional manifold.

**INPUT:**

- metric – (default: None) the pseudo-Riemannian metric g involved in the definition of the curl; if none is provided, the domain of self is supposed to be endowed with a default metric (i.e. is supposed to be pseudo-Riemannian manifold, see PseudoRiemannianManifold) and the latter is used to define the curl

**OUTPUT:**

- instance of VectorField representing the curl of self
EXAMPLES:

Curl of a vector field in the Euclidean 3-space:

```
sage: M.<x,y,z> = EuclideanSpace()
sage: v = M.vector_field(-y, x, 0, name='v')
sage: v.display()
v = -y e_x + x e_y
sage: s = v.curl(); s
Vector field curl(v) on the Euclidean space E^3
sage: s.display()
curl(v) = 2 e_z
```

The function `curl()` from the `operators` module can be used instead of the method `curl()`:

```
sage: from sage.manifolds.operators import curl
sage: curl(v) == s
True
```

If one prefers the notation `rot` over `curl`, it suffices to do:

```
sage: from sage.manifolds.operators import curl as rot
sage: rot(v) == s
True
```

The curl of a gradient vanishes identically:

```
sage: f = M.scalar_field(function('F')(x,y,z))
sage: gradf = f.gradient()
sage: gradf.display()
d(F)/dx e_x + d(F)/dy e_y + d(F)/dz e_z
sage: s = curl(gradf); s
Vector field on the Euclidean space E^3
sage: s.display()
0
```

```
∂(other, metric=None)
```

Return the scalar product of `self` with another vector field (with respect to a given metric).

If `self` is the vector field \( u \) and other is the vector field \( v \), the scalar product of \( u \) by \( v \) with respect to a given pseudo-Riemannian metric \( g \) is the scalar field \( s \) defined by

\[
s = u \cdot v = g(u, v) = g_{ij}u^i v^j
\]

**INPUT:**

- `other` – a vector field, defined on the same domain as `self`
- `metric` – (default: `None`) the pseudo-Riemannian metric \( g \) involved in the definition of the scalar product; if none is provided, the domain of `self` is supposed to be endowed with a default metric (i.e. is supposed to be pseudo-Riemannian manifold, see `PseudoRiemannianManifold`) and the latter is used to define the scalar product

**OUTPUT:**

- instance of `DiffScalarField` representing the scalar product of `self` by `other`.

**EXAMPLES:**

Scalar product in the Euclidean plane:
An shortcut alias of \texttt{dot\_product} is \texttt{dot}:

\begin{verbatim}
sage: u.dot(v) == s
True
\end{verbatim}

A test of orthogonality:

\begin{verbatim}
sage: v[:] = -y, x
sage: u.dot_product(v) == 0
True
\end{verbatim}

Scalar product with respect to a metric that is not the default one:

\begin{verbatim}
sage: h = M.riemannian_metric('h')
sage: h[:1,1], h[:2,2] = 1/(1+y^2), 1/(1+x^2)
sage: s = u.dot_product(v, metric=h); s
Scalar field h(u,v) on the Euclidean plane \(\mathbb{E}^2\)
sage: s.display()
h(u,v): \(\mathbb{E}^2 \to \mathbb{R}\)
\( (x, y) \mapsto -(x^3*y - x*y^3)/((x^2 + 1)*y^2 + x^2 + 1) \)
\end{verbatim}

Scalar product of two vector fields along a curve (a lemniscate of Gerono):

\begin{verbatim}
sage: R.<t> = manifolds.RealLine()
sage: C = M.curve([sin(t), sin(2*t)/2], (t, 0, 2*pi), name='C')
sage: u = C.tangent_vector_field(name='u')
sage: u.display()
u = \cos(t) \ e_x + (2*\cos(t)^2 - 1) \ e_y
sage: I = C.domain(); I
Real interval \((0, 2*pi)\)
sage: v = I.vector_field(cos(t), -1, dest_map=C, name='v')
sage: v.display()
v = \cos(t) \ e_x - \ e_y
sage: s = u.dot_product(v); s
Scalar field u.v on the Real interval \((0, 2*pi)\)
sage: s.display()
u.v: \( (0, 2*pi) \to \mathbb{R} \)
\( t \mapsto \sin(t)^2 \)
\end{verbatim}

Scalar product between a vector field along the curve and a vector field on the ambient Euclidean plane:

\begin{verbatim}
sage: e_x = M.cartesian_frame()[1]
sage: s = u.dot_product(e_x); s
Scalar field u.e_x on the Real interval \((0, 2*pi)\)
sage: s.display()
\begin{align*}
u.e_x: (0, 2*pi) & \to \mathbb{R} \\
\quad & t \mapsto \cos(t)
\end{align*}
\end{verbatim}
**dot_product** (*other, metric=None*)

Return the scalar product of *self* with another vector field (with respect to a given metric).

If *self* is the vector field *u* and other is the vector field *v*, the scalar product of *u* by *v* with respect to a given pseudo-Riemannian metric *g* is the scalar field *s* defined by

\[ s = u \cdot v = g(u, v) = g_{ij}u^iv^j \]

**INPUT:**

- *other* – a vector field, defined on the same domain as *self*
- *metric* – (default: None) the pseudo-Riemannian metric *g* involved in the definition of the scalar product; if none is provided, the domain of *self* is supposed to be endowed with a default metric (i.e. is supposed to be pseudo-Riemannian manifold, see *PseudoRiemannianManifold*) and the latter is used to define the scalar product

**OUTPUT:**

- instance of *DiffScalarField* representing the scalar product of *self* by *other*.

**EXAMPLES:**

Scalar product in the Euclidean plane:

```python
sage: M.<x,y> = EuclideanSpace()
sage: u = M.vector_field(x, y, name='u')
sage: v = M.vector_field(y, x, name='v')
sage: s = u.dot_product(v); s
Scalar field u.v on the Euclidean plane E^2
sage: s.display()
 u.v: E^2 → ℝ
       (x, y) ↦ 2*x*y
```

A shortcut alias of *dot_product* is *dot*:

```python
sage: u.dot(v) == s
True
```

A test of orthogonality:

```python
sage: v[:]= -y, x
sage: u.dot_product(v) == 0
True
```

Scalar product with respect to a metric that is not the default one:

```python
sage: h = M.riemannian_metric('h')
sage: h[1,1], h[2,2] = 1/(1+y^2), 1/(1+x^2)
sage: s = u.dot_product(v, metric=h); s
Scalar field h(u,v) on the Euclidean plane E^2
sage: s.display()
h(u,v): E^2 → ℝ
      (x, y) ↦ -(x^3*y - x*y^3)/((x^2 + 1)*y^2 + x^2 + 1)
```

Scalar product of two vector fields along a curve (a lemniscate of Gerono):

```python
sage: R.<t> = manifolds.RealLine()
sage: C = M.curve([sin(t), sin(2*t)/2], (t, 0, 2*pi), name='C')
sage: u = C.tangent_vector_field(name='u')
```

(continues on next page)
Scalar product between a vector field along the curve and a vector field on the ambient Euclidean plane:

```python
sage: e_x = M.cartesian_frame()[1]
sage: s = u.dot_product(e_x); s
Scalar field u.e_x on the Real interval (0, 2*pi)
sage: s.display()
u.e_x: (0, 2*pi) → ℝ
t ↦ cos(t)
```

**norm** (*metric=None*)

Return the norm of *self* (with respect to a given metric).

The *norm* of a vector field *v* with respect to a given pseudo-Riemannian metric *g* is the scalar field ||*v*|| defined by

\[
||v|| = \sqrt{g(v, v)}
\]

**Note:** If the metric *g* is not positive definite, it may be that ||*v*|| takes imaginary values.

**INPUT:**

- *metric* – (default: None) the pseudo-Riemannian metric *g* involved in the definition of the norm; if none is provided, the domain of *self* is supposed to be endowed with a default metric (i.e. is supposed to be pseudo-Riemannian manifold, see `PseudoRiemannianManifold`) and the latter is used to define the norm

**OUTPUT:**

- instance of `DiffScalarField` representing the norm of *self*.

**EXAMPLES:**

Norm in the Euclidean plane:

```python
sage: M.<x,y> = EuclideanSpace()
sage: v = M.vector_field(-y, x, name='v')
sage: s = v.norm(); s
Scalar field |v| on the Euclidean plane E^2
sage: s.display()
|v|: E^2 → ℝ
(x, y) ↦ sqrt(x^2 + y^2)
```

The global function `norm()` can be used instead of the method `norm()`:
Norm with respect to a metric that is not the default one:

\[
\text{sage: } h = M.\text{riemannian}\_\text{metric}(\text{'h'})
\]
\[
\text{sage: } h[1,1], h[2,2] = 1/(1+y^2), 1/(1+x^2)
\]
\[
\text{sage: } s = v.\text{norm}(\text{metric}=h); \ s
\]
\[
\text{Scalar field } |v|_h \text{ on the Euclidean plane } \mathbb{E}^2
\]
\[
|v|_h: \mathbb{E}^2 \rightarrow \mathbb{R}
\]
\[
(x, \ y) \mapsto \sqrt{(2*x^2 + 1)*y^2 + x^2)/(\sqrt{x^2 + 1})*\sqrt{y^2 + 1})
\]

Norm of the tangent vector field to a curve (a lemniscate of Gerono):

\[
\text{sage: } R.\lt t \gt = \text{manifolds.}\text{RealLine}()
\]
\[
\text{sage: } C = M.\text{curve}([\sin(t), \sin(2*t)/2], (t, 0, 2*\pi), \text{name}='C')
\]
\[
\text{sage: } v = C.\text{tangent}\_\text{vector}\_\text{field}()
\]
\[
|C'|: (0, 2*\pi) \rightarrow \mathbb{R}
\]
\[
t \mapsto \sqrt{4*cos(t)^4 - 3*cos(t)^2 + 1}
\]

\textbf{plot} (\text{chart=None, ambient_coords=None, mapping=None, chart_domain=None, fixed_coords=None,}
\text{ ranges=None, number_values=None, steps=None, parameters=None, label_axes=True, max_range=8,}
\text{ scale=1, color='blue', **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:

- \textbf{chart} – (default: None) the ambient chart (see above); if None, the default chart of the vector field’s domain is used
- \textbf{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
- \textbf{mapping} – \textit{DiffMap} (default: None); differentiable map \(\Phi\) providing the link between the vector field’s domain and the ambient chart \textbf{chart}; if None, the identity map is assumed
- \textbf{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
- \textbf{fixed_coords} – (default: None) dictionary with keys the coordinates of \textbf{chart_domain} that are kept fixed and with values the value of these coordinates; if None, all the coordinates of \textbf{chart_domain} are used
- \textbf{ranges} – (default: None) dictionary with keys the coordinates of \textbf{chart_domain} to be used and values tuples \((x_{\text{min}}, x_{\text{max}})\) specifying the coordinate range for the plot; if None, the entire coordinate range declared during the construction of \textbf{chart_domain} is considered (with \(-\infty\) replaced by \(-\text{max_range}\) and \(+\infty\) by \text{max_range})
- \textbf{number_values} – (default: None) either an integer or a dictionary with keys the coordinates of \textbf{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```
```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: v = M.vector_field(-y, x, name='v')
sage: v.display()
v = -y ∂/∂x + x ∂/∂y
sage: v.plot() #←needs sage.plot
Graphics object consisting of 80 graphics primitives```

Plot with various options:

```sage```
```
sage: v.plot(scale=0.5, color='green', linestyle='--', width=1, #←needs sage.plot
....:     arrowsize=6)
Graphics object consisting of 80 graphics primitives```

```sage```
```
sage: v.plot(max_range=4, number_values=5, scale=0.5) #←needs sage.plot
Graphics object consisting of 24 graphics primitives```

Plot using parallel computation:
2.7. Vector Fields
2.7. Vector Fields
Let us now consider a vector field on a 4-dimensional manifold:

\begin{verbatim}
sage: M = Manifold(4, 'M')
sage: X.<t,x,y,z> = M.chart()
sage: v = M.vector_field((t/8)^2, -t*y/4, t*x/4, t*z/4, name='v')
\end{verbatim}
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},
          ranges={x: (-2,2), y: (-2,2), t: (-1, 4)},
          number_values=4)
```

![3D plot of vector field v]
or, for a 2D plot:

```
sage: v.plot(ambient_coords=(x, y), fixed_coords={t: 1, z: 0})       # long
    ← time, needs sage.plot
Graphics object consisting of 80 graphics primitives
```

```
sage: v.plot(ambient_coords=(x, t), fixed_coords={y: 1, z: 0})       # long
    ← time, needs sage.plot
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 $\mathbb{R}^3$:

```
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):	heta 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: $S^2 \rightarrow R^3$
on U: $(\theta, \phi) \mapsto (x, y, z) = (\cos(\phi)\sin(\theta), \sin(\phi)\sin(\theta), \cos(\theta))$
sage: v = XS.frame()[1] ; v # the coordinate vector $\partial/\partial \phi$
Vector field $\partial/\partial \phi$ on the Open subset $U$ of the 2-dimensional differentiable manifold $S^2$
```

(continues on next page)
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,
    latex_name=None)

Bases: FiniteRankFreeModuleElement, MultivectorFieldParal, 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

\[
\Phi : U \rightarrow M,
\]
a vector field along \( U \) with values on \( M \) is a differentiable map

\[
v : U \rightarrow TM
\]
\((TM\) 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 \( \mathbb{R} \)).

**Note:** If \( M \) is not parallelizable, then VectorField must be used instead.

**INPUT:**
- vector_field_module -- free module \( \mathfrak{X}(U, \Phi) \) 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(name='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 \( \mathfrak{X}(M) \) 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().__class__.__name__
VectorFieldParal
sage: V = M.vector_field_module(); V
```
A vector field is a tensor field of rank 1 and of type \((1,0)\):
Components of a vector field with respect to a given frame:

```python
sage: e = M.vector_frame('e') ; M.set_default_frame(e)
sage: v[0], v[1], v[2] = (1+y, 4*x*z, 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 [:]:

```python
sage: v[:] = (1+y, 4*x*z, 9)
sage: v[:]
[1 + y, 4*x*z, 9]
```

The components are also read on the expansion on the frame e, as provided by the method `display()`:

```python
sage: v.display()  # expansion in the default frame
V = (1 + y) e_0 + 4*x*z e_1 + 9 e_2
```

A subset of the components can be accessed by using slice notation:

```python
sage: v[1:] = (-2, -x*y)
sage: v[:]
[1 + y, -2, -x*y]
sage: v[:2]
[1 + y, -2]
```

Components in another frame:

```python
sage: f = M.vector_frame('f')
sage: for i in range(3):
....:    v.set_comp(f)[i] = (i+1)**3 * c_xyz[i]
sage: v.comp(f)[2]
27*z
sage: v[f, 2]  # equivalent to above
27*z
sage: v.display(f)
V = x f_0 + 8*y f_1 + 27*z f_2
```

One can set the components at the vector definition:

```python
sage: v = M.vector_field(1+y, 4*x*z, 9, name='V')
sage: v.display()
V = (1 + y) e_0 + 4*x*z e_1 + 9 e_2
```

If the components regard a vector frame different from the default one, the vector frame has to be specified via the argument `frame`:

```python
sage: v = M.vector_field(x, 8*y, 27*z, frame=f, name='V')
sage: v.display(f)
V = x f_0 + 8*y f_1 + 27*z f_2
```

For providing the components in various frames, one may use a dictionary:
It is also possible to construct a vector field from a vector of symbolic expressions (or any other iterable):

```
sage: v = M.vector_field(vector([1+y, 4*x*z, 9]), name = V)
sage: v.display()
V = (y + 1) e_0 + 4*x*z e_1 + 9 e_2
```

The range of the indices depends on the convention set for the manifold:

```
sage: M = Manifold(3, M, start_index=1)
sage: e = M.vector_frame('e'); M.setdefault_frame(e)
sage: v = M.vector_field(1+y, 4*x*z, 9, name = V)
sage: v[0]
Traceback (most recent call last):
  ... IndexError: index out of range: 0 not in [1, 3]
sage: v[1]  # OK
y + 1
```

A vector field acts on scalar fields (derivation along the vector field):

```
sage: f = M.scalar_field(x*y^2, name = f)
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: c.t = R.chart()  # canonical chart on R
sage: Phi = R.diff_map(M, [cos(t), sin(t)], name = 'Phi')
  Differentiable map $\Phi$ from the 1-dimensional differentiable manifold $R$
to the 2-dimensional differentiable manifold $M$
sage: Phi.display()
$\Phi: R \to M$
t \mapsto (x, y) = (\cos(t), \sin(t))
sage: w = R.vector_field(-sin(t), cos(t), dest_map = Phi, name = '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
```
Value at a given point:

```python
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 = ∂/∂y
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 $\Phi : U \to M$ 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 $R$).

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:**

- Travis Scrimshaw (2016): review tweaks
- Eric Gourgoulhon (2018): some refactoring and more functionalities in the choice of symbols for vector frame elements (github issue #24792)

**REFERENCES:**

- [Lee2013]

**EXAMPLES:**

Introducing a chart on a manifold automatically endows it with a vector frame: the coordinate frame associated to the chart:

```python
sage: M = Manifold(3, 'M')
sage: X.<x,y,z> = M.chart()
sage: M.frames()
[Coordinate frame (M, (∂/∂x,∂/∂y,∂/∂z))]
sage: M.frames()[0] is X.frame()
True
```

A new vector frame can be defined from a family of 3 linearly independent vector fields:
The first frame defined on a manifold is its *default frame*; in the present case it is the coordinate frame associated to the chart \( X \):

```python
sage: M.default_frame()
Coordinate frame \((\partial/\partial x, \partial/\partial y, \partial/\partial z)\)
```

The default frame can be changed via the method `set_default_frame()`:

```python
sage: M.set_default_frame(e)
sage: M.default_frame()
Vector frame \((e_0, e_1, e_2)\)
```

The elements of a vector frame are vector fields on the manifold:

```python
sage: for vec in e:
    print(vec)
   ...:
   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 of a vector frame can be accessed by its index:

```python
sage: e[0]
Vector field \(e_0\) on the 3-dimensional differentiable manifold \(M\)
```

The slice operator `:` can be used to access to more than one element:

```python
sage: e[0:2]
(Vector field \(e_0\) on the 3-dimensional differentiable manifold \(M\),
   Vector field \(e_1\) on the 3-dimensional differentiable manifold \(M\))
```

Vector frames can be constructed from scratch, without any connection to previously defined frames or vector fields (the connection can be performed later via the method `set_change_of_frame()`):

```python
sage: f = M.vector_frame('f'); f
Vector frame \((f_0, f_1, f_2)\)
```
The index range depends on the starting index defined on the manifold:

```python
sage: M = Manifold(3, 'M', start_index=1)
sage: X.<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]
```

Let us check that the vector fields $e_i$ are the frame vectors from their components with respect to the frame $e$:

```python
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, by default, bares the same name (here $e$):

```python
sage: M.coframes()
[Coordinate coframe (M, (dx,dy,dz)), Coframe (M, (e^1,e^2,e^3))]
```

```python
sage: f = M.coframes()[1] ; f
Coframe (M, (e^1,e^2,e^3))
```

```python
sage: f is e.coframe()
True
```

Each element of the coframe is a 1-form:

```python
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)
```

Let us check that the coframe $(e^i)$ is indeed the dual of the vector frame $(e_i)$:

```python
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
```

```python
sage: f[2](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)
```

(continues on next page)
The coordinate frame associated to spherical coordinates of the sphere $S^2$:

```python
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'\theta:[0,\pi]:\theta ph:[0,2*pi]:\phi')
sage: b = M.default_frame(); b
Coordinate frame (S^2, (\partial/\partial th,\partial/\partial ph))
sage: b[1]
Vector field \partial/\partial th on the 2-dimensional differentiable manifold S^2
sage: b[2]
Vector field \partial/\partial ph on the 2-dimensional differentiable manifold S^2
```

The orthonormal frame constructed from the coordinate frame:

```python
sage: e = M.vector_frame('e', (b[1], b[2]/sin(th))); e
Vector frame (S^2, (e_1,e_2))
sage: e[1].display()
e_1 = \partial/\partial th
sage: e[2].display()
e_2 = 1/sin(th) \partial/\partial ph
```

The change-of-frame automorphisms and their matrices:

```python
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)]
```

```python
class sage.manifolds.differentiable.vectorframe.CoFrame(frame, symbol,
latex_symbol=None, indices=None, latex_indices=None)
```

**Bases:** FreeModuleCoBasis

Coframe on a differentiable manifold.

By *coframe*, it is meant a field $f$ on some differentiable manifold $U$ endowed with a differentiable map $\Phi : U \to M$ 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 $\mathbb{R}$).

**INPUT:**

- frame – the vector frame dual to the coframe
- symbol – either a string, to be used as a common base for the symbols of the 1-forms constituting the coframe, or a tuple of strings, representing the individual symbols of the 1-forms
• `latex_symbol` – (default: `None`) either a string, to be used as a common base for the LaTeX symbols of the 1-forms constituting the coframe, or a tuple of strings, representing the individual LaTeX symbols of the 1-forms; if `None`, `symbol` is used in place of `latex_symbol`.

• `indices` – (default: `None`; used only if `symbol` is a single string) tuple of strings representing the indices labelling the 1-forms of the coframe; if `None`, the indices will be generated as integers within the range declared on the coframe’s domain.

• `latex_indices` – (default: `None`) tuple of strings representing the indices for the LaTeX symbols of the 1-forms of the coframe; if `None`, `indices` is used instead.

EXAMPLES:

Coframe on a 3-dimensional manifold:

```python
sage: M = Manifold(3, 'M', start_index=1)
sage: X.<x,y,z> = M.chart()
sage: v = M.vector_frame('v')
sage: from sage.manifolds.differentiable.vectorframe import CoFrame
coframe = CoFrame(v, ('e'))
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:

```python
sage: a = v.dual_basis() ; a
coframe (M, (v^1,v^2,v^3))
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 `[]`:

```python
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`:

```python
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 \( \Phi : U \to M \) is the differentiable map associated with \( f \) (possibly \( \Phi = \text{Id}_U \))

**EXAMPLES:**

Cobasis of a tangent space on a 2-dimensional manifold:

```python
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_with_category'>
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
```

**set_name** (symbol, latex_symbol=None, indices=None, latex_indices=None, index_position='up', include_domain=True)

Set (or change) the text name and LaTeX name of *self*.

**INPUT:**

- **symbol** – either a string, to be used as a common base for the symbols of the 1-forms constituting the coframe, or a list/tuple of strings, representing the individual symbols of the 1-forms
- **latex_symbol** – (default: None) either a string, to be used as a common base for the LaTeX symbols of the 1-forms constituting the coframe, or a list/tuple of strings, representing the individual LaTeX symbols of the 1-forms; if None, symbol is used in place of latex_symbol
- **indices** – (default: None; used only if symbol is a single string) tuple of strings representing the indices labelling the 1-forms of the coframe; if None, the indices will be generated as integers within the range declared on *self*
- **latex_indices** – (default: None) tuple of strings representing the indices for the LaTeX symbols of the 1-forms; if None, indices is used instead
- **index_position** – (default: 'up') determines the position of the indices labelling the 1-forms of the coframe; can be either 'down' or 'up'
- **include_domain** – (default: True) boolean determining whether the name of the domain is included in the beginning of the coframe name

**EXAMPLES:**

```python
sage: M = Manifold(2, 'M')
sage: e = M.vector_frame('e').coframe(); e
Coframe (M, (e^0,e^1))
sage: e.set_name('f'); e
Coframe (M, (f^0,f^1))
```
```python
sage: e.set_name('epsilon', latex_symbol=r'\epsilon')
sage: latex(e)
\left(M, \left(\epsilon^{0},\epsilon^{1}\right)\right)
sage: e.set_name(include_domain=False); e
Coframe (e^0,e^1)
sage: e.set_name(['\alpha', '\beta'], latex_symbol=[r'\alpha', r'\beta']); e
Coframe (M, (a,b))
sage: latex(e)
\left(M, \left(\alpha,\beta\right)\right)
sage: e.set_name(indices=['x','y'], latex_indices=[r'\xi', r'\zeta']); e
Coframe (M, (e^x,e^y))
sage: latex(e)
\left(M, \left(e^{\xi},e^{\zeta}\right)\right)
```

```python
class sage.manifolds.differentiable.vectorframe.CoordCoFrame(coord_frame, symbol, latex_symbol=None, indices=None, latex_indices=None)

Bases: 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
- `symbol` – either a string, to be used as a common base for the symbols of the 1-forms constituting the coframe, or a tuple of strings, representing the individual symbols of the 1-forms
- `latex_symbol` – (default: None) either a string, to be used as a common base for the LaTeX symbols of the 1-forms constituting the coframe, or a tuple of strings, representing the individual LaTeX symbols of the 1-forms; if None, symbol is used in place of `latex_symbol`
- `indices` – (default: None; used only if `symbol` is a single string) tuple of strings representing the indices labelling the 1-forms of the coframe; if None, the indices will be generated as integers within the range declared on the vector frame's domain
- `latex_indices` – (default: None) tuple of strings representing the indices for the LaTeX symbols of the 1-forms of the coframe; if None, indices is used instead

**EXAMPLES:**

Coordinate coframe on a 3-dimensional manifold:

```python
sage: M = Manifold(3, 'M', start_index=1)
sage: X.<x,y,z> = M.chart()
sage: M.frames()
[Coordinate frame (M, (∂/∂x,∂/∂y,∂/∂z))]
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 `[]`:
The coframe is the dual of the coordinate frame:

```
sage: e = X.frame() ; e
Coordinate frame (M, (∂/∂x,∂/∂y,∂/∂z))
sage: dX[1](e[1]).expr(), dX[2](e[2]).expr(), dX[3](e[3]).expr()
(1, 0, 0)
sage: dX[1](e[1]).expr(), dX[2](e[2]).expr(), dX[3](e[3]).expr()
(0, 1, 0)
sage: dX[1](e[1]).expr(), dX[2](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
dsage: dX[1].exterior_derivative() == 0
True
```

**class** `sage.manifolds.differentiable.vectorframe.Coframe(chart)`

* Bases: `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...
sage: M.chart(r'\theta:[0,pi]\phi:[0,2*pi])
Chart (S^2, (\theta, \phi))
sage: b = M.default_frame()
sage: b
Coordinate frame (S^2, (\partial/\partial \theta, \partial/\partial \phi))
sage: b[1]
Vector field \partial/\partial \theta on the 2-dimensional differentiable manifold S^2
sage: b[2]
Vector field \partial/\partial \phi on the 2-dimensional differentiable manifold S^2
sage: latex(b)
```

(continues on next page)
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 \(C^k_{ij}\) defined by

\[
[e_i, e_j] = C^k_{ij} e_k.
\]

In the present case, since \((e_i)\) is a coordinate frame, \(C^k_{ij} = 0\).

OUTPUT:

- the structure coefficients \(C^k_{ij}\), as a vanishing instance of \(\text{CompWithSym}\) with 3 indices ordered as \((k, i, j)\)

EXAMPLES:

Structure coefficients of the coordinate frame associated to spherical coordinates in the Euclidean space \(\mathbb{R}^3\):

```
sage: M = Manifold(3, 'R^3', r'\mathbb{R}^3', start_index=1)  # Part of \mathbb{R}^3 covered by spherical coord.
sage: c_spher = M.chart(r'r:0\rightarrow +oo theta:0\rightarrow \pi ph:0\rightarrow 2\pi')
sage: b = M.default_frame() ; b
Coordinate frame (\mathbb{R}^3, (\partial/\partial r, \partial/\partial \theta, \partial/\partial \phi))
sage: c = b.structure_coeff() ; c
3-indices components w.r.t. Coordinate frame (R^3, (\partial/\partial r, \partial/\partial \theta, \partial/\partial \phi)), with antisymmetry on the index positions (1, 2)
sage: c == 0
True
```

```
class sage.manifolds.differentiable.vectorframe.VectorFrame
(\text{vector_field_module},
 symbol,
 latex_symbol=\text{None},
 from_frame=\text{None},
 indices=\text{None},
 latex_indices=\text{None},
 symbol_dual=\text{None},
 latex_symbol_dual=\text{None})
```

Bases: 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 \( \Phi : U \to M \) 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 \( \mathbb{R} \)).

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 \( \text{coframe()} \).

INPUT:

- vector_field_module – free module \( X(U, \Phi) \) of vector fields along \( U \) with values on \( M \supset \Phi(U) \)
- symbol – either a string, to be used as a common base for the symbols of the vector fields constituting the vector frame, or a tuple of strings, representing the individual symbols of the vector fields
- latex_symbol – (default: None) either a string, to be used as a common base for the LaTeX symbols of the vector fields constituting the vector frame, or a tuple of strings, representing the individual LaTeX symbols of the vector fields; if None, symbol is used in place of latex_symbol
- from_frame – (default: None) vector frame \( \hat{e} \) on the codomain \( M \) of the destination map \( \Phi \); the constructed frame \( e \) is then such that \( \forall p \in U, e(p) = \hat{e}(\Phi(p)) \)
- indices – (default: None; used only if symbol is a single string) tuple of strings representing the indices labelling the vector fields of the frame; if None, the indices will be generated as integers within the range declared on the vector frame’s domain
- latex_indices – (default: None) tuple of strings representing the indices for the LaTeX symbols of the vector fields; if None, indices is used instead
- symbol_dual – (default: None) same as symbol but for the dual coframe; if None, symbol must be a string and is used for the common base of the symbols of the elements of the dual coframe
- latex_symbol_dual – (default: None) same as latex_symbol but for the dual coframe

EXAMPLES:

Defining a vector frame on a 3-dimensional manifold:

```sage
sage: M = Manifold(3, 'M', start_index =1)
sage: X.<x,y,z> = M.chart()
sage: e = M.vector_frame('e') ; e
Vector frame (M, (e_1,e_2,e_3))
sage: latex(e)
\left(M, \left(e_{1},e_{2},e_{3}\right)\right)
```

The individual elements of the vector frame are accessed via square brackets, with the possibility to invoke the slice operator ‘:’ to get more than a single element:

```sage
sage: e[2]
Vector field e_2 on the 3-dimensional differentiable manifold M
sage: e[1:3]
(Vector field e_1 on the 3-dimensional differentiable manifold M,
 Vector field e_2 on the 3-dimensional differentiable manifold M)
sage: e[:]
(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)
```

The LaTeX symbol can be specified:
By default, the elements of the vector frame are labelled by integers within the range specified at the manifold declaration. It is however possible to fully customize the labels, via the argument `indices`:

```python
sage: u = M.vector_frame('u', indices=('x', 'y', 'z')); u
Vector frame (M, (u_x,u_y,u_z))
```

The LaTeX format of the indices can be adjusted:

```python
sage: v = M.vector_frame('v', indices=('a', 'b', 'c'),
                      latex_indices=(r'\alpha', r'\beta', r'\gamma'))
```

The symbol of each element of the vector frame can also be freely chosen, by providing a tuple of symbols as the first argument of `vector_frame`; it is then mandatory to specify as well some symbols for the dual coframe:

```python
sage: h = M.vector_frame(('a', 'b', 'c'), symbol_dual=('A', 'B', 'C'))
```

Example with a non-trivial map $\Phi$ (see above); a vector frame along a curve:

```python
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')
```

The value of the vector frame at a given point is a basis of the corresponding tangent space:
Vector frames are bases of free modules formed by vector fields:

```sage
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
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 \( \Phi : U \to V \) 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 \( \Phi : U \to V \)

**OUTPUT:**

- • vector frame \( \tilde{e} \) along \( U \) defined above.

**EXAMPLES:**

Vector frame along a curve:

```sage
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
```

(continues on next page)
Differentiable map $\Phi$ from the 1-dimensional differentiable manifold $R$ to the 2-dimensional differentiable manifold $M$

```sage```
```
es = X.frame() ; e
Coordinate frame $(M, (\partial/\partial x, \partial/\partial y))$
```
```
sage: te = e.along(Phi) ; te
Vector frame $(R, (\partial/\partial x, \partial/\partial y))$ 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```
```
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```
```
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```
```
sage: e.ambient_domain() == e.domain()
True
```
```
An example with a non trivial map $\Phi$:

```sage```
```
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 $M$
```
```
(continues on next page)
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, (∂/∂x,∂/∂y))
sage: ep = e.at(p) ; ep
Basis (∂/∂x,∂/∂y) 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_with_category'>
sage: ep[0]
Tangent vector ∂/∂x at Point p on the 2-dimensional differentiable manifold M
sage: ep[1]
Tangent vector ∂/∂y 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 (∂/∂x,∂/∂y) on the Tangent space at Point p on the 2-dimensional differentiable manifold M]
```

Let us consider a vector frame that is a not a coordinate one:

```
sage: aut = M.automorphism_field()
sage: aut[:1] = [[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 (∂/∂x,∂/∂y) on the Tangent space at Point p on the 2-dimensional differentiable manifold M]
```
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 $\text{aut}$ defined above):

```python
sage: Tp.change_of_basis(ep, fp)
Automorphism of the Tangent space at Point $p$ on the 2-dimensional differentiable manifold $M$
```

```python
sage: Tp.change_of_basis(ep, fp).display()
5 \partial/\partial x \otimes dx + 2 \partial/\partial y \otimes dy
```

```python
sage: Tp.change_of_basis(fp, ep)
Automorphism of the Tangent space at Point $p$ on the 2-dimensional differentiable manifold $M$
```

```python
sage: Tp.change_of_basis(fp, ep).display()
1/5 \partial/\partial x \otimes dx + 1/2 \partial/\partial y \otimes dy
```

The dual bases:

```python
sage: e.coframe()
Coordinate coframe $(M, (dx, dy))$
```

```python
sage: ep.dual_basis()
Dual basis $(dx, dy)$ on the Tangent space at Point $p$ on the 2-dimensional differentiable manifold $M$
```

```python
sage: ep.dual_basis() is e.coframe().at(p)
True
```

```python
sage: f.coframe()
Coframe $(M, (f^0, f^1))$
```

```python
sage: fp.dual_basis()
Dual basis $(f^0, f^1)$ on the Tangent space at Point $p$ on the 2-dimensional differentiable manifold $M$
```

```python
sage: fp.dual_basis() is f.coframe().at(p)
True
```

coframe()

Return the coframe of self.

EXAMPLES:

```python
sage: M = Manifold(2, 'M')
sage: e = M.vector_frame('e')
sage: e.coframe()
Coframe $(M, (e^0, e^1))$
```

```python
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 $\Phi : U \to M$ 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:

```python
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 $\Phi$:

```python
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')
```

```python
sage: Phi
Differentiable map Phi from the 1-dimensional differentiable manifold U to the 2-dimensional differentiable manifold M
```

```python
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
```

```python
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:

```python
sage: M = Manifold(2, 'M')
sage: e = M.vector_frame('e')
sage: e.domain()
2-dimensional differentiable manifold M
```

```python
sage: U = M.open_subset('U')
sage: f = e.restrict(U)
sage: f.domain()
Open subset U of the 2-dimensional differentiable manifold M
```

ew_frame(change_of_frame, symbol=None, latex_symbol=None, indices=None, latex_indices=None, symbol_dual=None, latex_symbol_dual=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` – either a string, to be used as a common base for the symbols of the vector fields constituting the vector frame, or a list/tuple of strings, representing the individual symbols of the vector fields
- `latex_symbol` – (default: `None`) either a string, to be used as a common base for the LaTeX symbols of the vector fields constituting the vector frame, or a list/tuple of strings, representing the individual LaTeX symbols of the vector fields; if `None`, `symbol` is used in place of `latex_symbol`

594 Chapter 2. Differentiable Manifolds
• indices – (default: None; used only if symbol is a single string) tuple of strings representing the indices labelling the vector fields of the frame; if None, the indices will be generated as integers within the range declared on self

• latex_indices – (default: None) tuple of strings representing the indices for the LaTeX symbols of the vector fields; if None, indices is used instead

• symbol_dual – (default: None) same as symbol but for the dual coframe; if None, symbol must be a string and is used for the common base of the symbols of the elements of the dual coframe

• latex_symbol_dual – (default: None) same as latex_symbol but for the dual coframe

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: X.<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, (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 \( \mathbb{R}^2 \) to the unit disk:

```python
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] = [[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:

```python
sage: e_U[1].display()
e_1 = (-y^2 + 1) \partial/\partial x + (x^2 + 1) \partial/\partial y
sage: e_U[2].display()
e_2 = 2 \partial/\partial y
```

They are actually the restrictions of the original frame vectors:

```python
sage: e_U[1] is e[1].restrict(U)
True
True
```

set_name (symbol, latex_symbol=None, indices=None, latex_indices=None, index_position='down', include_domain=True)

Set (or change) the text name and LaTeX name of self.

INPUT:

• symbol – either a string, to be used as a common base for the symbols of the vector fields constituting the vector frame, or a list/tuple of strings, representing the individual symbols of the vector fields

• latex_symbol – (default: None) either a string, to be used as a common base for the LaTeX symbols of the vector fields constituting the vector frame, or a list/tuple of strings, representing the individual LaTeX symbols of the vector fields; if None, symbol is used in place of latex_symbol

• indices – (default: None; used only if symbol is a single string) tuple of strings representing the indices labelling the vector fields of the frame; if None, the indices will be generated as integers within the range declared on self

• latex_indices – (default: None) tuple of strings representing the indices for the LaTeX symbols of the vector fields; if None, indices is used instead
• **index_position** – (default: 'down') determines the position of the indices labelling the vector fields of the frame; can be either 'down' or 'up'

• **include_domain** – (default: True) boolean determining whether the name of the domain is included in the beginning of the vector frame name

**EXAMPLES:**

```python
sage: M = Manifold(2, 'M')
sage: e = M.vector_frame('e'); e
Vector frame (M, (e_0,e_1))
sage: e.set_name('f'); e
Vector frame (M, (f_0,f_1))
sage: e.set_name('e', include_domain=False); e
Vector frame (e_0,e_1)
sage: e.set_name(['a', 'b']); e
Vector frame (M, (a,b))
sage: e.set_name('e', indices=['x', 'y']); e
Vector frame (M, (e_x,e_y))
sage: e.set_name('e', latex_symbol=r'\epsilon')
 Latex(e)
\left(M, \left(\epsilon_{0},\epsilon_{1}\right)\right)
sage: e.set_name(e, latex_symbol=[r'\alpha, r\beta'])
 Latex(e)
\left(M, \left(\alpha,\beta\right)\right)
sage: e.set_name(e, latex_symbol=E, 
......: latex_indices=[r'\alpha, r\beta'])
 Latex(e)
\left(M, \left(E_{\alpha},E_{\beta}\right)\right)
```

**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 \( C^k_{ij} \) defined by

\[
[e_i, e_j] = C^k_{ij} e_k
\]

**OUTPUT:**

• the structure coefficients \( C^k_{ij} \), 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 \(\mathbb{R}^3\):

```python
sage: M = Manifold(3, 'R^3', r'\mathbb{R}^3', start_index=1) # Part of R^3 covered by spherical coordinates
sage: c_spher.<r,th,ph> = M.chart(rr:(0,+oo) th:(0,pi)\theta ph:(0,2*pi):phi)
sage: ch_frame = M.automorphism_field()
sage: e = c_spher.frame().new_frame(ch_frame, e)
sage: e[1][:] # components of e_1 in the manifold's default frame (\partial/\partial r, \partial/\partial \theta, \partial/\partial \phi)
[1, 0, 0]
sage: e[2][:]
[0, 1/r, 0]
```

(continues on next page)
2.7.4 Group of Tangent-Space Automorphism Fields

Given a differentiable manifold $U$ and a differentiable map $\Phi : U \to M$ 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 $\text{GL}(\mathfrak{X}(U, \Phi))$ of the module $\mathfrak{X}(U, \Phi)$ of vector fields along $U$ with values on $M \supset \Phi(U)$ (see VectorFieldModule). Note that $\mathfrak{X}(U, \Phi)$ is a module over $C^k(U)$, the algebra of differentiable scalar fields on $U$. Elements of $\text{GL}(\mathfrak{X}(U, \Phi))$ are fields along $U$ of automorphisms of tangent spaces to $M$.

Two classes implement $\text{GL}(\mathfrak{X}(U, \Phi))$ depending whether $M$ is parallelizable or not: AutomorphismFieldParalGroup and AutomorphismFieldGroup.

AUTHORS:

- Eric Gourgoulhon (2015): initial version
- Travis Scrimshaw (2016): review tweaks
- Michael Jung (2019): improve treatment of the identity element

REFERENCES:

- Chap. 15 of [God1968]

```python
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))
```

**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
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
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
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
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 ∂/∂x⊗dx + 2 ∂/∂y⊗dy
sage: a.restrict(V).display()
2 ∂/∂u⊗du + 2 ∂/∂v⊗dv
```

The identity element of the group $G$:

```sage
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, (∂/∂x,∂/∂y))
sage: eV = V.default_frame() ; eV
Coordinate frame (V, (∂/∂u,∂/∂v))
```

(continues on next page)
sage: e.display(eU)
Id = ∂/∂x⊗dx + ∂/∂y⊗dy
sage: e.display(eV)
Id = ∂/∂u⊗du + ∂/∂v⊗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()

Module $X(M)$ of vector fields on the 2-dimensional differentiable manifold $M$

sage: G.base_module()  # $S^2$ is the union of $U$ and $V$

Module $X(M)$ of vector fields on the 2-dimensional differentiable manifold $M$
is $M$'s 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,
class sage.manifolds.differentiable.automorphismfield_group.AutomorphismFieldParallelGroup(vector_module)

Bases: 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 $\Phi : U \to M$ to a parallelizable manifold $M$ (possibly $U = M$ and $\Phi = \mathrm{Id}_M$), the group of tangent-space automorphism fields associated with $U$ and $\Phi$ is the general linear group $\mathrm{GL}(\mathfrak{X}(U, \Phi))$ of the module $\mathfrak{X}(U, \Phi)$ of vector fields along $U$ with values on $M \supset \Phi(U)$ (see VectorFieldFreeModule). Note that $\mathfrak{X}(U, \Phi)$ is a free module over $C^k(U)$, the algebra of differentiable scalar fields on $U$. Elements of $\mathrm{GL}(\mathfrak{X}(U, \Phi))$ 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 $\mathfrak{X}(U, \Phi)$ 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( \mathfrak{X}(M) \right)
```

$G$ is nothing but the general linear group of the module $\mathfrak{X}(M)$:

```
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 \frac{\partial}{\partial x} + 2 \frac{\partial}{\partial y}
sage: a(v)
Vector field on the 2-dimensional differentiable manifold M
sage: a(v).display()
2 \frac{\partial}{\partial x} - 2 \frac{\partial}{\partial y}

Indeed the matrix of $a$ with respect to the frame $(\partial_x, \partial_y)$ is:

sage: a[X.frame(),:]
[ 1  0]
[ 0 -1]

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
Construction of an element by providing its components with respect to the manifold’s default frame (frame associated to the coordinates \((x, y)\)):

\[
\text{sage: } b = G([1+x^2, 0], [0, 1+y^2]) \;; b
\]

Field of tangent-space automorphisms on the 2-dimensional differentiable manifold M

\[
\text{sage: } b.display()
\]

\((x^2 + 1) \partial/\partial x \otimes dx + (y^2 + 1) \partial/\partial y \otimes dy
\]

\[
\text{sage: } (-b).display() \quad \# \text{ the inverse automorphism}
\]

\[1/(x^2 + 1) \partial/\partial x \otimes dx + 1/(y^2 + 1) \partial/\partial y \otimes dy
\]

We check the group law on these elements:

\[
\text{sage: } (a*b)^{-1} == b^{-1} * a^{-1}
\]

True

Invertible tensor fields of type \((1, 1)\) can be converted to elements of \(G\):

\[
\text{sage: } t = M.tensor_field(1, 1, name='t')
\]

Field of tangent-space automorphisms \(t\) on the 2-dimensional differentiable manifold M

\[
\text{sage: } t1 = G(t) \;; t1
\]

Field of tangent-space automorphisms \(t\)\(^{-1}\) on the 2-dimensional differentiable manifold M

\[
\text{sage: } (t1^{-1}).display()
\]

\[t^{-1} = \frac{1}{(e^y + 1)} \partial/\partial x \otimes dx - \frac{x*y}{(x^2 + 1)*e^y + 1} \partial/\partial y \otimes dy + 1/(x^2 + 1) \partial/\partial y \otimes dy
\]

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:

\[
\text{sage: } T11 = M.tensor_field_module((1,1)) \;; T11
\]

Free module \(T^{(1,1)}(M)\) of type\(-(1,1)\) tensors fields on the 2-dimensional differentiable manifold M

\[
\text{sage: } T11.has_coerce_map_from(G)
\]

True

An explicit call of this coercion map is:

\[
\text{sage: } tt = T11(t1) \;; tt
\]

Tensor field \(t\) of type \((1,1)\) on the 2-dimensional differentiable manifold M

\[
\text{sage: } tt == t
\]

True

An implicit call of the coercion map is performed to subtract an element of \(G\) from an element of \(T^{(1,1)}(M)\):
as well as for the reverse operation:

\[
\textit{sage}: s = t1 - t ; s \\
\text{sage}: s = \text{t}$-$t1 \; s \\
\text{Tensor field t-t of type (1,1) on} \\
\text{the 2-dimensional differentiable manifold M} \\
\text{sage}: s.parent() \text{ is T11} \\
\text{True} \\
\text{sage}: s.display() \\
\text{t-t = 0}
\]

Element alias of \texttt{AutomorphismFieldParal}

### 2.7.5 Tangent-Space Automorphism Fields

The class \texttt{AutomorphismField} implements fields of automorphisms of tangent spaces to a generic (a priori not parallelizable) differentiable manifold, while the class \texttt{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

\texttt{class sage.manifolds.differentiable.automorphismfield.AutomorphismField(vector_field_module, name=\textit{None}, latex_name=\textit{None})}

Bases: \texttt{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 \(\Phi: U \to M\) to a differentiable manifold \(M\), a field of tangent-space automorphisms along \(U\) with values on \(M \supset \Phi(U)\) is a differentiable map

\[
a: U \to T^{(1,1)} M,
\]

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 \(\mathbb{R}\)).
Note: If $M$ is parallelizable, then $\text{AutomorphismFieldParal}$ 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')
U = M.open_subset('U') ; V = M.open_subset('V')
M.declare_union(U,V) # M is the union of U and V
c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart()
transf = c_xy.transition_map(c_uv, (x+y, x-y), intersection_name='W',
....: restrictions1= x>0, restrictions2= u+v>0)
inv = transf.inverse()
a = M.automorphism_field(name='a') ; a
Field of tangent-space automorphisms $a$ on the 2-dimensional
differentiable manifold $M$
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()
a[eU, :] = [[1,x], [0,2]]
```

It is equivalent to pass the components while defining $a$:

```sage
a = M.automorphism_field({eU: [[1,x], [0,2]]}, name='a')
```

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)
a.add_comp_by_continuation(eV, W, c_uv)
```

At this stage, the automorphism field $a$ is fully defined:

```sage
a.display(eU)
a = \frac{1}{4}u + \frac{1}{4}v + 3/2 \partial/\partial u \partial/\partial x + x \partial/\partial x \partial/\partial y + 2 \partial/\partial y \partial/\partial y
```

In particular, we may ask for its inverse on the whole manifold $M$:
Field of tangent-space automorphisms $a^{-1}$ on the 2-dimensional differentiable manifold $M$

```
sage: ia.display(eU)
a^{-1} = \partial/\partial x \otimes dx - 1/2 \cdot x \partial/\partial x \otimes dy + 1/2 \partial/\partial y \otimes dy
sage: ia.display(eV)
a^{-1} = (-1/8 \cdot u - 1/8 \cdot v + 3/4) \partial/\partial u \otimes du + (1/8 \cdot u + 1/8 \cdot v + 1/4) \partial/\partial v \otimes dv + (-1/8 \cdot u - 1/8 \cdot v + 1/4) \partial/\partial v \otimes du + (1/8 \cdot u + 1/8 \cdot v + 3/4) \partial/\partial u \otimes 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
```

```
add_comp (basis=None)

Return the components of self w.r.t. a given module basis for assignment, keeping the components w.r.t. other bases.

To delete the components w.r.t. other bases, use the method set_comp() instead.

INPUT:

- **basis** (default: None): basis in which the components are defined; if none is provided, the components are assumed to refer to the module’s default basis

```
Warning: If the automorphism field has already components in other bases, it is the user’s responsibility to make sure that the components to be added are consistent with them.

```

```
OUTPUT:

- components in the given basis, as an instance of the class 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
declare_union(U,V) # $S^2$ is the union of $U$ and $V$
sage: e_uv = c_uv.frame() # stereographic coordinates from the South pole
```
```
sage: a.add_comp(e_uv)
2-indices components w.r.t. Coordinate frame (V, (\partial/\partial u, \partial/\partial v))
sage: a.add_comp(e_uv)[0,0] = u+v
sage: a.add_comp(e_uv)[1,1] = u+v
sage: a.display(e_uv)
a = (u + v) \partial/\partial u \otimes du + (u + v) \partial/\partial v \otimes dv
```

Setting the components in a new frame:
E = V.vector_frame('e')
E.add_comp(E)
2-indices components w.r.t. Vector frame (V, (e_0,e_1))
E.add_comp(E)[0,1] = u*v
E.add_comp(E)[1,0] = u*v
E.display(E)
E = u*v e_0⊗e^1 + u*v e_1⊗e^0
The components with respect to e_uv are kept:
E.display(E_uv)
E = (u + v) ∂/∂u⊗du + (u + v) ∂/∂v⊗dv
Since the identity map is a special element, its components cannot be changed:
Id = M.tangent_identity_field()
Id.add_comp(E)[0,1] = u*v
Traceback (most recent call last):
...
ValueError: the components of an immutable element cannot be changed

**copy** *(name=None, latex_name=None)*

Return an exact copy of the automorphism field *self*.

**INPUT:**

* • name – (default: None) name given to the copy
* • latex_name – (default: None) LaTeX symbol to denote the copy; if none is provided, the LaTeX symbol is set to name

**Note:** The name and the derived quantities are not copied.

**EXAMPLES:**

M = Manifold(2, 'M')
U = M.open_subset('U') ; V = M.open_subset('V')
M.declare_union(U,V)  # M is the union of U and V
x,y = U.chart() ; u,v = V.chart()
xy_to_uv = x,y.transition_map(u,v, (x+y, x-y),
intersection_name='W', restrictions1= x>0,
restrictions2= u+v>0)
uv_to_xy = xy_to_uv.inverse()
Id = M.tangent_identity_field(); Id
Field of tangent-space identity maps on the 2-dimensional differentiable manifold M
one = Id.copy('1'); one
Field of tangent-space automorphisms 1 on the 2-dimensional differentiable manifold M

**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:  W = U.intersection(V)

Field of tangent-space automorphisms \( a^{-1} \) on the 2-dimensional differentiable manifold \( M \).

Let us check that \( ia \) is indeed the inverse of \( a \):

\[
\begin{align*}
\text{sage: } s &= a \cdot \text{contract}(ia) \\
\text{sage: } s[eU,:], s[eV,:] &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}
\end{align*}
\]

The result is cached:

\[
\begin{align*}
\text{sage: } \text{a.inverse()} \text{ is } ia &\quad \text{True} \\
\text{sage: } \text{ia is } a^{-1} &\quad \text{True} \\
\text{sage: } \text{ia is } \sim a &\quad \text{True}
\end{align*}
\]

Instead of \texttt{inverse}(), one can use the power minus one to get the inverse:

\[
\begin{align*}
\text{sage: } \text{ia is } a^{(-1)} &\quad \text{True} \\
\text{sage: } \text{ia is } -a &\quad \text{True}
\end{align*}
\]

\texttt{restrict} \( \text{(subdomain, dest_map=None)} \)
Return the restriction of \texttt{self} to some subdomain.

This is a redefinition of \texttt{sage.manifolds.differentiable.tensorfield.TensorField.restrict()} to take into account the identity map.

\textbf{INPUT:}

- \texttt{subdomain} -- \texttt{DifferentiableManifold} open subset \(V\) of \(\texttt{self._domain}\)
- \texttt{dest_map} -- (default: \texttt{None}) \texttt{DiffMap}; destination map \(\Phi : V \rightarrow N\), where \(N\) is a subdomain of \(\texttt{self._codomain}\); if \texttt{None}, the restriction of \(\texttt{self.base_module().destination_map()}\) to \(V\) is used

\textbf{OUTPUT:}

- a \texttt{AutomorphismField} representing the restriction

\textbf{EXAMPLES:}

Restrictions of an automorphism field on the 2-sphere:

\begin{verbatim}
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 stereo. coord. from North pole to W
sage: stereoS_W = W.atlas()[1] # restriction of stereo. coord. from South pole to W
sage: eN_W = stereoN_W.frame(); eS_W = stereoS_W.frame()
sage: a = M.automorphism_field({eN: [[ 1, atan(x^2+y^2)], [ 0, 3]],
... name='a'});
    a.add_comp_by_continuation(eS, W, chart=stereoS); a
Field of tangent-space automorphisms \(a\) on the 2-dimensional differentiable manifold \(S^2\)
\end{verbatim}

Field of tangent-space automorphisms \(a\) on the Open subset \(U\) of the 2-dimensional differentiable manifold \(S^2\)

\begin{verbatim}
sage: a.restrict(U)
Field of tangent-space automorphisms \(a\) on the Open subset \(U\) of the 2-dimensional differentiable manifold \(S^2\)
\end{verbatim}

Field of tangent-space automorphisms \(a\) on the Open subset \(V\) of the 2-dimensional differentiable manifold \(S^2\)

\begin{verbatim}
sage: a.restrict(V)
Field of tangent-space automorphisms \(a\) on the Open subset \(V\) of the 2-dimensional differentiable manifold \(S^2\)
\end{verbatim}

(continues on next page)
\[2u^2v^2 + v^4 \cdot (3u^4 - 2u^2v^2 + 3v^4 - 2(u^3v - u*v^3)*\arctan(1/(u^2 + v^2)))/(u^4 + 2u^2v^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]
```

**set_comp (basis=None)**

Return the components of self w.r.t. a given module basis for assignment.

The components with respect to other bases are deleted, in order to avoid any inconsistency. To keep them, use the method *add_comp*() instead.

**INPUT:**

- **basis** = (default: None) basis in which the components are defined; if none is provided, the components are assumed to refer to the module’s default basis

**OUTPUT:**

- components in the given basis, as an instance of the class *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()
```

(continues on next page)
sage: a = M.automorphism_field(name='a')
sage: a.set_comp(e_uv)
2-indices components w.r.t. Coordinate frame (V, (\partial/\partial u, \partial/\partial v))
sage: a.set_comp(e_uv)[0,0] = u+v
sage: a.set_comp(e_uv)[1,1] = u+v
sage: a.display(e_uv)
a = (u + v) \partial/\partial u \otimes du + (u + v) \partial/\partial v \otimes dv

Setting the components in a new frame:
sage: e = V.vector_frame('e')
sage: a.set_comp(e)
2-indices components w.r.t. Vector frame (V, (e_0,e_1))
sage: a.set_comp(e)[0,1] = u*v
sage: a.set_comp(e)[1,0] = u*v
sage: a.display(e)
a = u*v e_0 \otimes e^1 + u*v e_1 \otimes e^0

Since the frames \(e\) and \(e_{uv}\) are defined on the same domain, the components w.r.t. \(e_{uv}\) have been erased:

sage: a.display(c_uv.frame())
Traceback (most recent call last):
  ... ValueError: no basis could be found for computing the components in the Coordinate frame (V, (\partial/\partial u, \partial/\partial v))

Since the identity map is an immutable element, its components cannot be changed:

sage: id = M.tangent_identity_field()
sage: id.add_comp(e)[0,1] = u*v
Traceback (most recent call last):
  ... ValueError: the components of an immutable element cannot be changed

class\ sage.manifolds.differentiable.automorphismfield.AutomorphismFieldParal (vector_field_module, name=None, latex_name=None)

Bases: sage.manifolds.differentiable.tensorfield.TensorFieldParal

Field of tangent-space automorphisms with values on a parallelizable manifold.

Given a differentiable manifold \(U\) and a differentiable map \(\Phi : U \to M\) to a parallelizable manifold \(M\), a field of tangent-space automorphisms along \(U\) with values on \(M \supset \Phi(U)\) is a differentiable map

\[ a : U \to T^{(1,1)}M \]

\((T^{(1,1)}M\) 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 \(\mathbb{R}\)).
Note: If $M$ is not parallelizable, the class `AutomorphismField` must be used instead.

**INPUT:**

- `vector_field_module` – free module $\mathfrak{X}(U, \Phi)$ 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

**EXAMPLES:**

A $\pi/3$-rotation in the Euclidean 2-plane:

```python
sage: M = Manifold(2, 'R^2')
sage: c_xy.<x,y> = M.chart()
sage: rot = M.automorphism_field([[sqrt(3)/2, -1/2], [1/2, sqrt(3)/2]], name='R')
```

Field of tangent-space automorphisms $R$ on the 2-dimensional differentiable manifold $R^2$

```python
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()`:

```python
sage: inv = rot.inverse() ; inv
```

Field of tangent automorphisms $R^{-1}$ on the 2-dimensional differentiable manifold $R^2$

```python
sage: latex(inv)
```

$R^{-1}$

```python
sage: inv[:]
```

```
[1/2*sqrt(3) 1/2]
[ -1/2 1/2*sqrt(3)]
```

```python
sage: rot[:]
```

```
[1/2*sqrt(3) -1/2]
[ 1/2 1/2*sqrt(3)]
```

```python
sage: inv[:] * rot[:]
```

# check
```
[1 0]
[0 1]
```

Equivalently, one can use the power minus one to get the inverse:

```python
sage: inv is rot^(-1)
```

True

```python
sage: inv is ~rot
```

True

**at** *(point)*

Value of `self` at a given point.

If the current field of tangent-space automorphisms is

$$a : U \rightarrow T^{(1,1)}M$$
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:**

- \( \text{point} = \text{ManifoldPoint} \): 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:

```python
sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart()
sage: a = M.automorphism_field([[1+exp(y), x*y], [0, 1+x^2]],
...: name='a')
sage: a.display()
a = (e^y + 1) \frac{\partial}{\partial x} \otimes dx + x*y \frac{\partial}{\partial x} \otimes dy + (x^2 + 1) \frac{\partial}{\partial y} \otimes 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) \frac{\partial}{\partial x} \otimes dx - 6 \frac{\partial}{\partial x} \otimes dy + 5 \frac{\partial}{\partial y} \otimes 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 \( p \):

```python
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 = \frac{\partial}{\partial x} \otimes dx + \frac{\partial}{\partial y} \otimes 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 \( \text{self} \).

**EXAMPLES:**
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: a = M.automorphism_field([[0, 2], [-1, 0]], name='a')
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 $\Phi : V \to N$, 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([[1, x*y], [0, 3]], name='a'); a
Field of tangent-space automorphisms a on the 2-dimensional differentiable manifold R^2
sage: a.restrict(D)```
Field of tangent-space automorphisms $a$ on the Open subset $D$ of the
2-dimensional differentiable manifold $\mathbb{R}^2$
```
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 $\mathbb{R}^2$
```
```
sage: id.restrict(D)
Field of tangent-space identity maps on the Open subset $D$ of the
2-dimensional differentiable manifold $\mathbb{R}^2$
```
```
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:

- Travis Scrimshaw (2016): review tweaks

REFERENCES:

- [KN1963]
- [Lee2013]
- [ONe1983]

```python
class sage.manifolds.differentiable.tensorfield_module.TensorFieldFreeModule(vec-
tor_field_mod-
ule, ten-
sor_type)
```

Bases: `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
\[
\Phi : U \rightarrow M,
\]
the tensor field module \(T^{(k,l)}(U, \Phi)\) is the set of all tensor fields of the type
\[
t : U \rightarrow T^{(k,l)}M
\]
(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 \texttt{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 \(\mathbb{R}\)).

**Note:** If \(M\) is not parallelizable, the class \texttt{TensorFieldModule} should be used instead, for \(T^{(k,l)}(U, \Phi)\) is no longer a free module.

**INPUT:**
- \texttt{vector_field_module} – free module \(\mathcal{X}(U, \Phi)\) of vector fields along \(U\) associated with the map \(\Phi : U \rightarrow M\)
- \texttt{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 \(\mathbb{R}^3\):

```python
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 \(T^{(2,0)}(\mathbb{R}^3)\) of type\(-\)(2,0) tensors fields on the 3-dimensional differentiable manifold \(\mathbb{R}^3\)
```

\(T^{(2,0)}(\mathbb{R}^3)\) is a module over the algebra \(C^k(\mathbb{R}^3)\):

```python
sage: T20.category()
Category of tensor products of finite dimensional modules over
Algebra of differentiable scalar fields on the 3-dimensional differentiable manifold \(\mathbb{R}^3\)
sage: T20.base_ring() is M.scalar_field_algebra()
True
```

\(T^{(2,0)}(\mathbb{R}^3)\) is a free module:

```python
sage: from sage.tensor.modules.finite_rank_free_module import...
sage: isinstance(T20, FiniteRankFreeModule_abstract)
True
```

because \(M = \mathbb{R}^3\) is parallelizable:
sage: M.is_manifestly_parallelizable()
True

The zero element:

```plaintext
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:

```plaintext
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 $T^{(2,0)}(\mathbb{R}^3)$ coerces to any module of type-$(2,0)$ tensor fields defined on some subdomain of $\mathbb{R}^3$:

```plaintext
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: T20U.coerce_map_from(T20)  # the reverse is not true
Coercion map:
  From: Free module $T^{(2,0)}(\mathbb{R}^3)$ of type-$(2,0)$ tensors fields on the 3-
  dimensional differentiable manifold $\mathbb{R}^3$
  To:   Free module $T^{(2,0)}(\mathbb{U})$ of type-$(2,0)$ tensors fields on the Open subset $\mathbb{U}$
of the 3-dimensional differentiable manifold $\mathbb{R}^3$
```

The coercion map is actually the restriction of tensor fields defined on $\mathbb{R}^3$ to $U$.

There is also a coercion map from fields of tangent-space automorphisms to tensor fields of type $(1,1)$:

```plaintext
sage: T11 = M.tensor_field_module((1,1)) ; T11
Free module $T^{(1,1)}(\mathbb{R}^3)$ of type-$(1,1)$ tensors fields on the 3-
  dimensional differentiable manifold $\mathbb{R}^3$
```

```plaintext
sage: GL = M.automorphism_field_group() ; GL
General linear group of the Free module $X(\mathbb{R}^3)$ of vector fields on the 3-
  dimensional differentiable manifold $\mathbb{R}^3$
```

```plaintext
sage: T11.has_coerce_map_from(GL)
True
```

An explicit call to this coercion map is:

```plaintext
sage: id = GL.one() ; id
Field of tangent-space identity maps on the 3-dimensional differentiable manifold $\mathbb{R}^3$
```

```plaintext
sage: tid = T11(id) ; tid
Tensor field Id of type $(1,1)$ on the 3-dimensional differentiable manifold $\mathbb{R}^3$
```
Manifolds, Release 10.3

(continued from previous page)

```
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, tensor_type, category=None)

Bases: UniqueRepresentation, ReflexiveModule_tensor

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

\[
\Phi : U \rightarrow M,
\]

the tensor field module \(T^{(k,l)}(U, \Phi)\) is the set of all tensor fields of the type

\[
t : U \rightarrow T^{(k,l)}M
\]

(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 \(\Phi\) a curve in \(M\) (\(U\) is then an open interval of \(\mathbb{R}\)).

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
```

(continues on next page)
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 tensor products 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: from sage.tensor.modules.finite_rank_free_module import
    ...: FiniteRankFreeModule_abstract
sage: isinstance(T20, FiniteRankFreeModule_abstract)
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_abstract)
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 \( T^{(2,0)}(U) \):
Manifolds, Release 10.3

```
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]
```

(continues on next page)
sage: ta[eV,:]
[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 TensorField-Paral 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
class sage.manifolds.differentiable.tensorfield.TensorField(vector_field_module: VectorFieldModule, tensor_type: TensorType, name: str | None = None, latex_name: str | None = None, sym=None, antisym=None, parent=None)

Bases: ModuleElementWithMutability

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 \( \Phi : U \to M \). More precisely, given two non-negative integers \( k \) and \( l \) and a differentiable map

\[
\Phi : U \longrightarrow M,
\]

a \textit{tensor field of type} \( (k, l) \) \textit{along} \( U \) \textit{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

\[
t(p) : T^*_{[k \text{ times}]}M \times T_{[l \text{ times}]}M \longrightarrow K,
\]

where \( T^*_qM \) 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 \textit{tensor rank}.

The standard case of a tensor field \textit{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 \( \mathbb{R} \)).
If $M$ is parallelizable, the class \texttt{TensorFieldParal} should be used instead.

This is a Sage \textit{element} class, the corresponding \textit{parent} class being \texttt{TensorFieldModule}.

INPUT:

- \texttt{vector\_field\_module} – module $\mathfrak{X}(U, \Phi)$ of vector fields along $U$ associated with the map $\Phi : U \to M$ (cf. \texttt{VectorFieldModule})
- \texttt{tensor\_type} – pair $(k, l)$ with $k$ being the contravariant rank and $l$ the covariant rank
- \texttt{name} – (default: None) name given to the tensor field
- \texttt{latex\_name} – (default: None) \LaTeX{} symbol to denote the tensor field; if none is provided, the \LaTeX{} symbol is set to name
- \texttt{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:
  - $\texttt{sym} = (0, 1)$ for a symmetry between the 1st and 2nd arguments
  - $\texttt{sym} = [(0, 2), (1, 3, 4)]$ for a symmetry between the 1st and 3rd arguments and a symmetry between the 2nd, 4th and 5th arguments.
- \texttt{antisym} – (default: None) antisymmetry or list of antisymmetries among the arguments, with the same convention as for \texttt{sym}
- \texttt{parent} – (default: None) some specific parent (e.g. exterior power for differential forms); if None, \texttt{vector\_field\_module.tensor\_module(k, l)} is used

EXAMPLES:

Tensor field of type $(0,2)$ on the sphere $S^2$:

```python
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()
```

The parent of $t$ is not a free module, for the sphere $S^2$ is not parallelizable:

```python
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 the 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({eU: [-y, x]}, name='a')
sage: a.display(eV)
a = -v ∂/∂u + u ∂/∂v
sage: b = M.vector_field({eU: [y, -1]}, name='b')
sage: b.display(eV)
b = ((2*u + 1)*v^3 + (2*u^3 - u^2)*v)/(u^2 + v^2) ∂/∂u - (u^4 - v^4 + 2*u*v^2)/(u^2 + v^2) ∂/∂v
```

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
sage: # long time
sage: s = t(a.restrict(U), b) ; s
Scalar field t(a,b) on the Open subset U of the 2-dimensional differentiable manifold S^2
sage: s.display()
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)
```

The tensor itself can be defined only on some open subset of $S^2$, yielding a result whose domain is this subset:

```sage
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
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 f*t 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
```

2.8. Tensor Fields
Same examples with SymPy as the symbolic engine

From now on, we ask that all symbolic calculus on manifold $M$ are performed by SymPy:

```python
sage: M.set_calculus_method('sympy')
```

We define the tensor $t$ as above:

```python
sage: t = M.tensor_field(0, 2, {eU: [[1,0], [-2,3]]}, name='t')
sage: t.display(eU)
t = dx⊗dx - 2 dy⊗dx + 3 dy⊗dy
sage: t.add_comp_by_continuation(eV, W, chart=c_uv)  # long time
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*v*(u**2 - 2*u*v + v**2)/(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
```

The default coordinate representations of tensor components are now SymPy objects:

```python
sage: t[eV,1,1,c_uv].expr()  # long time
(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)
```

Let us consider two vector fields, $a$ and $b$, on $S^2$:

```python
sage: a = M.vector_field({eU: [-y, x]}, name='a')
sage: a.display(eV)
a = -v ∂/∂u + u ∂/∂v
sage: b = M.vector_field({eU: [y, -1]}, name='b')
sage: b.display(eV)
b = v*(2*u**3 - u**2 + 2*u*v**2 + v**2)/(u**2 + v**2) ∂/∂u + (-u**4 - 2*u*v**2 + v**4)/(u**2 + v**2) ∂/∂v
```

As a tensor field of type $(0, 2)$, $t$ acts on the pair $(a, b)$, resulting in a scalar field:

```python
sage: f = t(a, b)
sage: f.display()  # long time
t(a,b): S^2 → ℝ

| U: (x, y) | t(a,b) = -2*x*y - 3*x - y**2 |
| V: (u, v) | (-3*u**3 - 3*u*v**2 - 2*u*v - v**2)/(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:

```python
sage: s = t(a.restrict(U), b)
sage: s.display()  # long time
t(a,b): U → ℝ

| U: (x, y) | t(a,b) = -2*x*y - 3*x - y**2 |
| W: (u, v) | (-3*u**3 - 3*u*v**2 - 2*u*v - v**2)/(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
t(a,b): V → ℝ
(u, v) ↦ (-3*u**3 - 3*u*v**2 - 2*u*v - v**2)/(u**4 + 2*u**2*v**2 + v**4)
```

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')
s = f*t # long time
s[[0,0]] == f*t[[0,0]] # long time
True
s.restrict(U) == f.restrict(U) * t.restrict(U) # long time
True
s = f*t.restrict(U)
s.restrict(U) == f.restrict(U) * t.restrict(U)
True
```

Notice that the zero tensor field is immutable, and therefore its components cannot be changed:

```sage
zer = M.tensor_field_module((1, 1)).zero()
sage: zer.is_immutable()
True
sage: zer.set_comp()
Traceback (most recent call last):
  ...:          ValueError: the components of an immutable element cannot be changed
```

Other tensor fields can be declared immutable, too:

```sage
t.is_immutable()
False
t.set_immutable()
sage: t.is_immutable()
True
t.set_comp()
Traceback (most recent call last):
  ...:          ValueError: the components of an immutable element cannot be changed
```

`add_comp (basis=None)`

Return the components of `self` in a given vector frame for assignment.

### 2.8. Tensor Fields
The components with respect to other frames having the same domain as the provided vector frame are kept. To delete 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 `Components`; if such components did not exist previously, they are created

**EXAMPLES:**

```python
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() # 3-indices components w.r.t. Coordinate frame (V, (∂/∂u,∂/∂v))
sage: t = M.tensor_field(1, 2, name='t')

sage: t.add_comp(e_uv)
3-indices components w.r.t. Coordinate frame (V, (∂/∂u,∂/∂v))

sage: t.add_comp(e_uv)[1,0,1] = u+v

sage: t.display(e_uv)
t = (u + v) ∂/∂v⊗du⊗dv

The components with respect to e_uv are kept:

```python
sage: t.display(e_uv)
t = (u + v) ∂/∂v⊗du⊗dv
```

Since zero is a special element, its components cannot be changed:

```python
sage: z = M.tensor_field_module((1, 1)).zero()
sage: z.add_comp(e_uv)[1, 1] = u^2
Traceback (most recent call last):
...
ValueError: the components of an immutable element cannot be changed
```

**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 $S^2$:

```plaintext
sage: M = Manifold(2, 'S^2', start_index=1)
```

The two open subsets covered by stereographic coordinates (North and South):

```plaintext
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
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 = 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: W = U.intersection(V) # The complement of the two poles
sage: eU = c_xy.frame() ; eV = c_uv.frame()
sage: a = M.vector_field({eU: [x, 2+y]}, name='a')
```

At this stage, the vector field has been defined only on the open subset $U$ (through its components in the frame $eU$):

```plaintext
sage: a.display(eU)
a = x ∂/∂x + (y + 2) ∂/∂y
```

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 $W$:

```plaintext
sage: a.display(eV.restrict(W), c_uv.restrict(W))
a = (-4*u*v - u) ∂/∂u + (2*u^2 - 2*v^2 - v) ∂/∂v
```

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:

```plaintext
sage: a.add_comp_by_continuation(eV, W, chart=c_uv)
```

We have then:

```plaintext
sage: a.display(eV)
a = (-4*u*v - u) ∂/∂u + (2*u^2 - 2*v^2 - v) ∂/∂v
```

and $a$ is defined on the entire manifold $S^2$.

**add_expr_from_subdomain**(frame, subdomain)

Add an expression to an existing component from a 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 have additional expressions.
EXAMPLES:

We are going to consider a vector field in $\mathbb{R}^3$ along the 2-sphere:

```python
sage: M = Manifold(3, 'M', structure='Riemannian')
sage: S = Manifold(2, 'S', structure='Riemannian')
sage: E.<X,Y,Z> = M.chart()
```

Let us define $S$ in terms of stereographic charts:

```python
sage: U = S.open_subset('U')
sage: V = S.open_subset('V')
sage: S.declare_union(U,V)
sage: stereoN.<x,y> = U.chart()
sage: stereoS.<xp,yp> = V.chart("xp:x yp:y")
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= xp^2+yp^2!=0)
sage: stereoS_to_N = stereoN_to_S.inverse()
sage: W = U.intersection(V)
sage: stereoN_W = stereoN.restrict(W)
sage: stereoS_W = stereoS.restrict(W)
```

The embedding of $S^2$ in $\mathbb{R}^3$:

```python
sage: phi = S.diff_map(M, {(stereoN, E): [2*x/(1+x^2+y^2),
....: 2*y/(1+x^2+y^2),
....: (x^2+y^2-1)/(1+x^2+y^2)],[stereoS, E]: [2*xp/(1+xp^2+yp^2),
....: 2*yp/(1+xp^2+yp^2),
....: (1-xp^2-yp^2)/(1+xp^2+yp^2)],
....: name='Phi', latex_name=r'$\Phi$')
```

To define a vector field $v$ along $S$ taking its values in $M$, we first set the components on $U$:

```python
sage: v = M.vector_field(name='v').along(phi)
sage: vU = v.restrict(U)
sage: vU[:] = [x,y,x^2+y^2]
```

But because $M$ is parallelizable, these components can be extended to $S$ itself:

```python
sage: v.add_comp_by_continuation(E.frame().along(phi), U)
```

One can see that $v$ is not yet fully defined: the components (scalar fields) do not have values on the whole manifold:

```python
sage: sorted(v._components.values())[0]._comp[(0,)].display()
S → $\mathbb{R}$ on U: $(x, y) \mapsto x$
on W: $(xp, yp) \mapsto xp/(xp^2 + yp^2)$
```

To fix that, we first extend the components from $W$ to $V$ using `add_comp_by_continuation()`:

```python
sage: v.add_comp_by_continuation(E.frame().along(phi).restrict(V),
....: W, stereoS)
```

Then, the expression on the subdomain $V$ is added to the already known components on $S$ by:
The definition of \( v \) is now complete:

\[
\text{sage: sorted(v._components.values())[0]._comp[(2,)].display()}
\]

\[
S \to \mathbb{R}
\] on \( U \): (\( x, y \)) \mapsto x^2 + y^2
on \( V \): (\( xp, yp \)) \mapsto \frac{1}{xp^2 + yp^2}

\textbf{along (mapping)}

Return the tensor field deduced from \( \text{self} \) via a differentiable map, the codomain of which is included in the domain of \( \text{self} \).

More precisely, if \( \text{self} \) is a tensor field \( t \) on \( M \) and if \( \Phi : U \to M \) is a differentiable map from some differentiable manifold \( U \) to \( M \), the returned object is a tensor field \( \tilde{t} \) along \( U \) with values on \( M \) such that

\[
\forall p \in U, \tilde{t}(p) = t(\Phi(p)).
\]

\textbf{INPUT:}

- \( \text{mapping} \) – differentiable map \( \Phi : U \to M \)

\textbf{OUTPUT:}

- tensor field \( \tilde{t} \) along \( U \) defined above.

\textbf{EXAMPLES:}

Let us consider the 2-dimensional sphere \( S^2 \):

\[
\text{sage: M = Manifold(2, 'S^2') \# the 2-dimensional sphere \( S^2 \)}
\text{sage: U = M.open_subset('U') \# complement of the North pole}
\text{sage: c_xy.<x,y> = U.chart() \# stereographic coordinates from the North pole}
\text{sage: V = M.open_subset('V') \# complement of the South pole}
\text{sage: c_uv.<u,v> = V.chart() \# stereographic coordinates from the South pole}
\text{sage: M.declare_union(U,V) \# \( S^2 \) is the union of \( U \) and \( V \)}
\text{sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
\text{intersection_name='W', restrictions1= x^2+y^2!=0,
\text{restrictions2= u^2+v^2!=0)}}
\text{sage: uv_to_xy = xy_to_uv.inverse()}
\text{sage: W = U.intersection(V)}
\]

and the following map from the open interval \((0, 5\pi/2)\) to \( S^2 \), the image of it being the great circle \( x = 0, u = 0 \), which goes through the North and South poles:

\[
\text{sage: I.<t> = manifolds.OpenInterval(0, 5*pi/2)}
\text{sage: J = I.open_interval(0, 3*pi/2)}
\text{sage: K = I.open_interval(pi, 5*pi/2)}
\text{sage: c_J = J.canonical_chart(); c_K = K.canonical_chart()}
\text{sage: Phi = I.diff_map(M, {(c_J, c_xy):
\text{.....:} (0, sgn(pi-t)*sqrt((1+cos(t))/(1-cos(t)))),
\text{.....:} (c_K, c_uv):
\text{.....:} (0, sgn(t-2*pi)*sqrt((1-cos(t))/(1+cos(t))))},
\text{.....:} name='Phi')}
\]

Let us consider a vector field on \( S^2 \):
We have then:

\[
\text{sage: } \mathbf{w} = \mathbf{w}.\text{along}(\Phi); \mathbf{w}
\]
Vector field \(\mathbf{w}\) along the Real interval \((0, 5/2\pi)\) with values on
the 2-dimensional differentiable manifold \(S^2\)
\[
\text{sage: } \mathbf{w}.\text{display}(\mathbf{eU}.\text{along}(\Phi))
\]
\(\mathbf{w} = \partial/\partial x\)
\[
\text{sage: } \mathbf{w}.\text{display}(\mathbf{eV}.\text{along}(\Phi))
\]
\(\mathbf{w} = -(\cos(t) - 1)*\text{sgn}(-2\pi + t)^2/(\cos(t) + 1) \partial/\partial u\)

Some tests:

\[
\text{sage: } p = K.\text{an_element}()
\]
\[
\text{sage: } \mathbf{w}.\text{at}(p) == \mathbf{w}.\text{at}(\Phi(p))
\]
True
\[
\text{sage: } \mathbf{w}.\text{at}(J(4\pi/3)) == \mathbf{w}.\text{at}(K(4\pi/3))
\]
True
\[
\text{sage: } \mathbf{w}.\text{at}(I(4\pi/3)) == \mathbf{w}.\text{at}(K(4\pi/3))
\]
True
\[
\text{sage: } \mathbf{w}.\text{at}(K(7\pi/4)) == \mathbf{eU}[0].\text{at}(\Phi(I(7\pi/4))) # \text{since } \mathbf{eU}[0]=\partial/\partial x
\]
True

\textbf{antisymmetrize}(\textit{pos})

Antisymmetrization over some arguments.

\textbf{INPUT:}

\begin{itemize}
  \item \textit{pos} = (default: None) list of argument positions involved in the antisymmetrization (with the convention \textit{position}=0 for the first argument); if None, the antisymmetrization is performed over all the arguments
\end{itemize}

\textbf{OUTPUT:}

\begin{itemize}
  \item the antisymmetrized tensor field (instance of \textit{TensorField})
\end{itemize}

\textbf{EXAMPLES:}

Antisymmetrization of a type-(0, 2) tensor field on a 2-dimensional non-parallelizable manifold:
sage: s = a.antisymmetrize() ; s
2-form on the 2-dimensional differentiable manifold M
sage: s[eU, :]
[ 0  1/2*x - 1/2]
[-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 \texttt{sage.tensor.modules.free_module_tensor.}
FreeModuleTensor.antisymmetrize().

\texttt{apply\_map (fun, frame=None, chart=None, keep\_other\_components=False)}

Apply a function to the coordinate expressions of all components of \texttt{self} in a given vector frame.

This method allows operations like factorization, expansion, simplification or substitution to be performed on all components of \texttt{self} in a given vector frame (see examples below).

INPUT:

- \texttt{fun} – function to be applied to the coordinate expressions of the components
- \texttt{frame} – (default: None) vector frame defining the components on which the operation \texttt{fun} is to be performed; if None, the default frame of the domain of \texttt{self} is assumed
- \texttt{chart} – (default: None) coordinate chart; if specified, the operation \texttt{fun} is performed only on the coordinate expressions with respect to \texttt{chart} of the components w.r.t. \texttt{frame}; if None, the operation \texttt{fun} is performed on all available coordinate expressions
- \texttt{keep\_other\_components} – (default: False) determine whether the components with respect to vector frames distinct from \texttt{frame} and having the same domain as \texttt{frame} are kept. If \texttt{fun} is non-destructive, \texttt{keep\_other\_components} can be set to True; otherwise, it is advised to set it to False (the default) in order to avoid any inconsistency between the various sets of components

EXAMPLES:

Factorizing all components in the default frame of a vector field:

sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: a, b = var('a b')
sage: v = M.vector_field(x^2 - y^2, a*(b^2 - b)*x)
sage: v.display()
(x^2 - y^2) \frac{\partial}{\partial x} + a*(b^2 - b)*x \frac{\partial}{\partial y}
sage: v.apply_map(factor)
sage: v.display()
(x + y)*(x - y) \frac{\partial}{\partial x} + a*(b - 1)*b*x \frac{\partial}{\partial y}

Performing a substitution in all components in the default frame:
sage: v.apply_map(lambda f: f.subs({a: 2}))(x + y)*(x - y) ∂/∂x + 2*(b - 1)*b*x ∂/∂y

Specifying the vector frame via the argument frame:
sage: P.<p, q> = M.chart()
sage: X_to_P = X.transition_map(P, [x + 1, y - 1])
sage: P_to_X = X_to_P.inverse()
sage: v.display(P)(p^2 - q^2 - 2*p - 2*q) ∂/∂p + (-2*b^2 + 2*(b^2 - b)*p + 2*b) ∂/∂q
sage: v.apply_map(lambda f: f.subs({b: pi}), frame=P.frame())
sage: v.display(P)(p^2 - q^2 - 2*p - 2*q) ∂/∂p + (2*pi - 2*pi^2 - 2*(pi - pi^2)*p) ∂/∂q

Note that the required operation has been performed in all charts:
sage: v.display(P.frame(), P)(p^2 - q^2 - 2*p - 2*q) ∂/∂p + (2*pi - 2*pi^2 - 2*(pi - pi^2)*p) ∂/∂q
sage: v.display(P.frame(), X)(p + q)*(p - q - 2) ∂/∂x + 2*pi*(pi - 1)*x ∂/∂y
sage: v.apply_map(expand, chart=P)(p^2 - q^2 - 2*p - 2*q) ∂/∂x + (2*pi + 2*pi^2*p - 2*pi^2 - 2*pi*p) ∂/∂y
sage: v.display(X.frame(), X)(x + y)*(x - y) ∂/∂x + 2*pi*(pi - 1)*x ∂/∂y

By default, the components of \( v \) in frames distinct from the specified one have been deleted:
sage: X.frame() in v._components
False

When requested, they are recomputed by change-of-frame formulas, thereby enforcing the consistency between the representations in various vector frames. In particular, we can check that the substitution of \( b \) by \( \pi \), which was asked in \( P.frame() \), is effective in \( X.frame() \) as well:
sage: v.display(X.frame(), P)(p^2 - q^2 - 2*p - 2*q) ∂/∂p + (2*pi - 2*pi^2 - 2*(pi - pi^2)*p) ∂/∂q
sage: v.display(X.frame(), X)(p + q)*(p - q - 2) ∂/∂x + 2*pi*(pi - 1)*x ∂/∂y

When the requested operation does not change the value of the tensor field, one can use the keyword argument keep_other_components=True, in order to avoid the recomputation of the components in other frames:
sage: v.apply_map(factor, keep_other_components=True)
sage: v.display()(x + y)*(x - y) ∂/∂x + 2*pi*(pi - 1)*x ∂/∂y

The components with respect to \( P.frame() \) have been kept:
sage: P.frame() in v._components
True

One can restrict the operation to expressions in a given chart, via the argument chart:
sage: v.display(X.frame(), P)(p + q)*(p - q - 2) ∂/∂x + 2*pi*(pi - 1)*(p - 1) ∂/∂y
sage: v.apply_map(expand, chart=P)(p^2 - q^2 - 2*p - 2*q) ∂/∂x + (2*pi + 2*pi^2*p - 2*pi^2 - 2*pi*p) ∂/∂y
sage: v.display(X.frame(), X)(x + y)*(x - y) ∂/∂x + 2*pi*(pi - 1)*x ∂/∂y

634 Chapter 2. Differentiable Manifolds
at \(point\)
Value of \(self\) at a point of its domain.

If the current tensor field is

\[ t : U \rightarrow T^{(k,l)} M \]

associated with the differentiable map

\[ \Phi : U \rightarrow M, \]

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:

```python
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, {eU: [[ 1+y,x], [ 0,x+y]], name='a'})
sage: a.add_comp_by_continuation(eV, W, chart=c_uv)
sage: a.display(eU)
a = (y + 1) \partial/\partial x \otimes dx + x \partial/\partial x \otimes dy + (x + y) \partial/\partial y \otimes dy
sage: a.display(eV)
a = (u + 1/2) \partial/\partial u \otimes du + (-1/2*u - 1/2*v + 1/2) \partial/\partial u \otimes dv
+ 1/2 \partial/\partial v \otimes du + (1/2*u - 1/2*v + 1/2) \partial/\partial v \otimes dv
sage: p = M.point((2,3), chart=c_xy, name='p')
sage: a.display(eU.at(p))
Type-(1,1) tensor a on the Tangent space at Point p on the 2-dimensional differentiable manifold M
sage: a.display(eV.at(p))
```

**base_module()**

Return the vector field module on which \(self\) acts as a tensor.

**OUTPUT:**
• instance of \texttt{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
```

\texttt{comp} \texttt{(basis=\texttt{None}, from\_basis=\texttt{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:**

- \texttt{basis} – (default: \texttt{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
- \texttt{from\_basis} – (default: \texttt{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 \texttt{basis}

**OUTPUT:**

- components in the vector frame \texttt{basis}, as a \texttt{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, (∂/∂x,∂/∂y))
sage: V = M.open_subset('V')
sage: c_uv.<u, v> = V.chart()
sage: f = V.default_frame() ; f
Coordinate frame (V, (∂/∂u,∂/∂v))
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[e,1,1] = - u*v
sage: t.comp(e)
2-indices components w.r.t. Coordinate frame (U, (∂/∂x,∂/∂y))
sage: t.comp(e)[:]
[y^3 - x x + 2]
[ 0 0]
sage: t.comp(f)
2-indices components w.r.t. Coordinate frame (V, (∂/∂u,∂/∂v))
sage: t.comp(f)[:]
```

(continues on next page)
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: h
2-indices components w.r.t. Vector frame (V, (h_0,h_1))
sage: h[:]
[ 0 -u^3*v/(v + 1)]
[ 0 -u*v]
```

\texttt{contract(*args)}

Contraction of \texttt{self} with another tensor field on one or more indices.

**INPUT:**

- \texttt{pos1} – positions of the indices in the current tensor field involved in the contraction; \texttt{pos1} must be a sequence of integers, with 0 standing for the first index position, 1 for the second one, etc.; if \texttt{pos1} is not provided, a single contraction on the last index position of the tensor field is assumed

- \texttt{other} – the tensor field to contract with

- \texttt{pos2} – positions of the indices in \texttt{other} involved in the contraction, with the same conventions as for \texttt{pos1}; if \texttt{pos2} is not provided, a single contraction on the first index position of \texttt{other} is assumed

**OUTPUT:**

- tensor field resulting from the contraction at the positions \texttt{pos1} and \texttt{pos2} of the tensor field with \texttt{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, {eU: [[1, x], [0, 2]]}, name='a')
sage: a.add_comp_by_continuation(eV, W, chart=c_uv)
sage: b = M.tensor_field(2, 0, {eU: [[y, -1], [x+y, 2]]}, name='b')
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...
```

(continues on next page)
Check 1: components with respect to the manifold’s default frame (eU):

```
sage: all(bool(s[i,j] == sum(a[i,k]*b[k,j] for k in M.irange()))
       ....: for i in M.irange() for j in M.irange())
True
```

Check 2: components with respect to the frame eV:

```
sage: all(bool(s[eV,i,j] == sum(a[eV,i,k]*b[eV,k,j]
       ....: for k in M.irange())
       ....: for i in M.irange() for j in M.irange()))
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[^{ij}_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
```

The same double contraction using index notation:

```sage```
s == c[^.._kl]*b[^kl]  # 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({eU: [y, 1+x]}, name='a')
sage: a.display(eU)
a = y dx + (x + 1) dy
```
sage: b.display(eU)
b = x ∂/∂x + y^2 ∂/∂y
```
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({eU: [1+x, -y]}, name='b')
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

\textbf{copy} (name=None, latex_name=None)

Return an exact copy of \texttt{self}.

INPUT:

- \texttt{name} – (default: None) name given to the copy
- \texttt{latex_name} – (default: None) LaTeX symbol to denote the copy; if none is provided, the LaTeX symbol is set to \texttt{name}

\textbf{Note}: The name and the derived quantities are not copied.

\textbf{EXAMPLES}:

Copy of a type-\((1, 1)\) tensor field on a 2-dimensional manifold:

\begin{verbatim}
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) \partial/\partial x \otimes dx + 2 \partial/\partial y \otimes dx + (-y + 1) \partial/\partial y \otimes dy
sage: s == t
True
\end{verbatim}

If the original tensor field is modified, the copy is not:

\begin{verbatim}
sage: t[e_xy,0,0] = -1
sage: t.display(e_xy)
t = -\partial/\partial x \otimes dx + 2 \partial/\partial y \otimes dx + (-y + 1) \partial/\partial y \otimes dy
sage: s.display(e_xy)
(x + y) \partial/\partial x \otimes dx + 2 \partial/\partial y \otimes dx + (-y + 1) \partial/\partial y \otimes dy
sage: s == t
False
\end{verbatim}

\textbf{copy_from} (other)

Make \texttt{self} a copy of \texttt{other}.  

640 Chapter 2. Differentiable Manifolds
INPUT:

• other — other tensor field, in the same module as self

Note: While the derived quantities are not copied, the name is kept.

Warning: All previous defined components and restrictions will be deleted!

EXAMPLES:

```python
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,0,0] = x+y
sage: t.display(e_xy)
t = (x + y) \partial/\partial x \otimes dx + 2 \partial/\partial y \otimes dx + (-y + 1) \partial/\partial y \otimes dy
```

While the original tensor field is modified, the copy is not:

```python
sage: t[e_xy,0,0] = -1
sage: t.display(e_xy)
t = -\partial/\partial x \otimes dx + 2 \partial/\partial y \otimes dx + (-y + 1) \partial/\partial y \otimes dy
sage: s == t
False
```

dalembertian (metric=None)

Return the d’Alembertian of self with respect to a given Lorentzian metric.

The d’Alembertian of a tensor field t with respect to a Lorentzian metric g is nothing but the Laplace-Beltrami operator of g applied to t (see `laplacian()`); if self a tensor field t of type (k, l), the d’Alembertian of t with respect to g is then the tensor field of type (k, l) defined by

\[(\Box t)^{a_1...a_k}_{b_1...b_k} = \nabla_i \nabla^j t^{a_1...a_k}_{b_1...b_k},\]

where \(\nabla\) is the Levi-Civita connection of g (cf. `LeviCivitaConnection`) and \(\nabla^i := g^{ij} \nabla_j\).

Note: If the metric g is not Lorentzian, the name d’Alembertian is not appropriate and one should use `laplacian()` instead.

INPUT:
• **metric** – (default: None) the Lorentzian metric \( g \) involved in the definition of the d'Alembertian; if none is provided, the domain of \( \text{self} \) is supposed to be endowed with a default Lorentzian metric (i.e. is supposed to be Lorentzian manifold, see \texttt{PseudoRiemannianManifold}) and the latter is used to define the d'Alembertian

**OUTPUT:**

• instance of \texttt{TensorField} representing the d'Alembertian of \( \text{self} \)

**EXAMPLES:**

d'Alembertian of a vector field in Minkowski spacetime, representing the electric field of a simple plane electromagnetic wave:

\[
\begin{align*}
\text{sage: } & M = \text{Manifold}(4, \ 'M', \ \text{structure}='\text{Lorentzian}') \\
\text{sage: } & X,\langle t,x,y,z \rangle = M.\text{chart}() \\
\text{sage: } & g = M.\text{metric}() \\
\text{sage: } & g[0,0], g[1,1], g[2,2], g[3,3] = -1, \ 1, \ 1, \ 1 \\
\text{sage: } & e = M.\text{vector_field}(\text{name}='e') \\
\text{sage: } & e[1] = \cos(t-z) \\
\text{sage: } & e.\text{display() } \# \text{ plane wave propagating in the z direction} \\
\text{sage: } & \text{De} = e.\text{dalembertian(); De } \# \text{ long time} \\
\text{sage: } & \text{Vector field Box(e) on the 4-dimensional Lorentzian manifold M}
\end{align*}
\]

The function \texttt{dalembertian()} from the \texttt{operators} module can be used instead of the method \texttt{dalembertian()}:

\[
\begin{align*}
\text{sage: } & \text{from sage.manifolds.operators import dalembertian} \\
\text{sage: } & \text{dalembertian(e) == De } \# \text{ long time} \\
\text{sage: } & \text{True}
\end{align*}
\]

We check that the electric field obeys the wave equation:

\[
\begin{align*}
\text{sage: } & \text{De.display() } \# \text{ long time} \\
\text{sage: } & \text{Box(e) = 0}
\end{align*}
\]

\texttt{disp} (\texttt{frame=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:**

• **frame** – (default: None) vector frame with respect to which the tensor is expanded; if frame is None and chart is not None, the coordinate frame associated with chart is assumed; if both frame and chart are 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 on a 2-dimensional manifold:

\[
\begin{align*}
\text{sage: } & M = \text{Manifold}(2, \ 'M') \\
\text{sage: } & U = M.\text{open_subset('U')} \ ; \ V = M.\text{open_subset('V')} \\
\text{sage: } & M.\text{declare_union(U,V)} \ # \text{M is the union of U and V} \\
\text{sage: } & c_{xy}.\langle x,y \rangle = U.\text{chart()} \ ; \ c_{uv}.\langle u,v \rangle = V.\text{chart()} \\
\text{sage: } & xy_{\text{to uv}} = c_{xy}.\text{transition_map(c_{uv}, (x+y, x-y)},
\end{align*}
\]
intersection_name=W, restrictions1= x>0,
restrictions2= u+v>0)
sage: W = U.intersection(V)
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, W, c_uv)
sage: t.display(e_xy)
t = x \frac{\partial}{\partial x} \otimes dx + \frac{\partial}{\partial x} \otimes dy + y \frac{\partial}{\partial y} \otimes dx
sage: t.display(e_uv)
t = (1/2*u + 1/2) \frac{\partial}{\partial u} \otimes du + (1/2*u - 1/2) \frac{\partial}{\partial u} \otimes dv
+ (1/2*v + 1/2) \frac{\partial}{\partial v} \otimes du + (1/2*v - 1/2) \frac{\partial}{\partial v} \otimes dv

Since e_xy is M’s default frame, the argument e_xy can be omitted:

sage: e_xy is M.default_frame()
True
sage: t.display()
t = x \frac{\partial}{\partial x} \otimes dx + \frac{\partial}{\partial x} \otimes dy + y \frac{\partial}{\partial y} \otimes dx

Similarly, since e_uv is V’s default frame, the argument e_uv can be omitted when considering the restriction of t to V:

sage: t.restrict(V).display()
t = (1/2*u + 1/2) \frac{\partial}{\partial u} \otimes du + (1/2*u - 1/2) \frac{\partial}{\partial u} \otimes dv
+ (1/2*v + 1/2) \frac{\partial}{\partial v} \otimes du + (1/2*v - 1/2) \frac{\partial}{\partial v} \otimes dv

If the coordinate expression of the components are to be displayed in a chart distinct from the default one on the considered domain, then the chart has to be passed as the second argument of display. For instance, on W = U \cap V, two charts are available: c_xy.restrict(W) (the default one) and c_uv.restrict(W). Accordingly, one can have two views of the expansion of t in the same vector frame e_uv.restrict(W):

sage: t.display(e_uv.restrict(W))  # W's default chart assumed
t = (1/2*x + 1/2*y + 1/2) \frac{\partial}{\partial u} \otimes du + (1/2*x + 1/2*y - 1/2) \frac{\partial}{\partial u} \otimes dv
+ (1/2*x - 1/2*y + 1/2) \frac{\partial}{\partial v} \otimes du + (1/2*x - 1/2*y - 1/2) \frac{\partial}{\partial v} \otimes dv
sage: t.display(e_uv.restrict(W), c_uv.restrict(W))
t = (1/2*u + 1/2) \frac{\partial}{\partial u} \otimes du + (1/2*u - 1/2) \frac{\partial}{\partial u} \otimes dv
+ (1/2*v + 1/2) \frac{\partial}{\partial v} \otimes du + (1/2*v - 1/2) \frac{\partial}{\partial v} \otimes dv

As a shortcut, one can pass just a chart to display. It is then understood that the expansion is to be performed with respect to the coordinate frame associated with this chart. Therefore the above command can be abridged to:

sage: t.display(c_uv.restrict(W))
t = (1/2*u + 1/2) \frac{\partial}{\partial u} \otimes du + (1/2*u - 1/2) \frac{\partial}{\partial u} \otimes dv
+ (1/2*v + 1/2) \frac{\partial}{\partial v} \otimes du + (1/2*v - 1/2) \frac{\partial}{\partial v} \otimes dv

and one has:

sage: t.display(c_xy)
t = x \frac{\partial}{\partial x} \otimes dx + \frac{\partial}{\partial x} \otimes dy + y \frac{\partial}{\partial y} \otimes dx
sage: t.display(c_uv)
t = (1/2*u + 1/2) \frac{\partial}{\partial u} \otimes du + (1/2*u - 1/2) \frac{\partial}{\partial u} \otimes dv
+ (1/2*v + 1/2) \frac{\partial}{\partial v} \otimes du + (1/2*v - 1/2) \frac{\partial}{\partial v} \otimes dv
sage: t.display(c_xy.restrict(W))
One can ask for the display with respect to a frame in which \( t \) has not been initialized yet (this will automatically trigger the use of the change-of-frame formula for tensors):

\[
sage: a = V.automorphism_field()
sage: a[:][:] = [[1+v, -u^2], [0, 1-u]]
sage: f = e_uv.new_frame(a, 'f')
sage: [f[i].display() for i in M.irange()]
\]

\[
\begin{align*}
\text{f}_0 &= (v + 1) \partial/\partial u, \\
\text{f}_1 &= -u^2 \partial/\partial u + (-u + 1) \partial/\partial v
\end{align*}
\]

\[
sage: t.display(f)
\]

\[
\begin{align*}
t &= -1/2*((u^2*v + 1)/(u - 1)) f_0 \otimes f^0 \\
&+ 1/2*((u^2 + (u^2 + u - 1)*v - u + 1)/(u - 1)) f_1 \otimes f^1
\end{align*}
\]

A shortcut of \texttt{display() \texttt{is \texttt{disp()}}:}

\[
sage: t.disp(e_uv)
\]

\[
\begin{align*}
t &= (1/2*u + 1/2) \partial/\partial u \otimes du + (1/2*u - 1/2) \partial/\partial u \otimes dv \\
&+ (1/2*v + 1/2) \partial/\partial v \otimes du + (1/2*v - 1/2) \partial/\partial v \otimes dv
\end{align*}
\]

\[
\textbf{display (frame=None, chart=None)}
\]

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:**

- \texttt{frame} – (default: \texttt{None}) \texttt{vector frame} with respect to which the tensor is expanded; if \texttt{frame} is \texttt{None} and \texttt{chart} is not \texttt{None}, the coordinate frame associated with \texttt{chart} is assumed; if both \texttt{frame} and \texttt{chart} are \texttt{None}, the default frame of the domain of definition of the tensor field is assumed.

- \texttt{chart} – (default: \texttt{None}) \texttt{chart} with respect to which the components of the tensor field in the selected frame are expressed; if \texttt{None}, the default chart of the vector frame domain is assumed.

**EXAMPLES:**

Display of a type-(1, 1) tensor field on a 2-dimensional manifold:
\[ t = x \frac{\partial}{\partial x} \otimes dx + \frac{\partial}{\partial x} \otimes dy + y \frac{\partial}{\partial y} \otimes dx \]

\texttt{sage: t.display(e_uv)}
\[ t = \frac{1}{2}u + \frac{1}{2} \frac{\partial}{\partial u} \otimes du + \frac{1}{2}u - \frac{1}{2} \frac{\partial}{\partial u} \otimes dv \\
+ \frac{1}{2}v + \frac{1}{2} \frac{\partial}{\partial v} \otimes du + \frac{1}{2}v - \frac{1}{2} \frac{\partial}{\partial v} \otimes dv \]

Since \( e_{xy} \) is \( M \)'s default frame, the argument \( e_{xy} \) can be omitted:

\texttt{sage: e_xy is M.default_frame()}
True
\texttt{sage: t.display()}
\[ t = x \frac{\partial}{\partial x} \otimes dx + \frac{\partial}{\partial x} \otimes dy + y \frac{\partial}{\partial y} \otimes dx \]

Similarly, since \( e_{uv} \) is \( V \)'s default frame, the argument \( e_{uv} \) can be omitted when considering the restriction of \( t \) to \( V \):

\texttt{sage: t.restrict(V).display()}
\[ t = \frac{1}{2}u + \frac{1}{2} \frac{\partial}{\partial u} \otimes du + \frac{1}{2}u - \frac{1}{2} \frac{\partial}{\partial u} \otimes dv \\
+ \frac{1}{2}v + \frac{1}{2} \frac{\partial}{\partial v} \otimes du + \frac{1}{2}v - \frac{1}{2} \frac{\partial}{\partial v} \otimes dv \]

If the coordinate expression of the components are to be displayed in a chart distinct from the default one on the considered domain, then the chart has to be passed as the second argument of \texttt{display}. For instance, on \( W = U \cap V \), two charts are available: \( c_{xy}.restrict(W) \) (the default one) and \( c_{uv}.restrict(W) \). Accordingly, one can have two views of the expansion of \( t \) in the \texttt{same} vector frame \( e_{uv}.restrict(W) \):

\texttt{sage: t.display(e_uv.restrict(W))} \hspace{1cm} \# W's default chart assumed
\[ t = \frac{1}{2}x + \frac{1}{2}y + \frac{1}{2} \frac{\partial}{\partial u} \otimes du + \frac{1}{2}x + \frac{1}{2}y - \frac{1}{2} \frac{\partial}{\partial u} \otimes dv \\
+ \frac{1}{2}x - \frac{1}{2}y + \frac{1}{2} \frac{\partial}{\partial v} \otimes du + \frac{1}{2}x - \frac{1}{2}y - \frac{1}{2} \frac{\partial}{\partial v} \otimes dv \]

\texttt{sage: t.display(e_uv.restrict(W), c_uv.restrict(W))}
\[ t = \frac{1}{2}u + \frac{1}{2} \frac{\partial}{\partial u} \otimes du + \frac{1}{2}u - \frac{1}{2} \frac{\partial}{\partial u} \otimes dv \\
+ \frac{1}{2}v + \frac{1}{2} \frac{\partial}{\partial v} \otimes du + \frac{1}{2}v - \frac{1}{2} \frac{\partial}{\partial v} \otimes dv \]

As a shortcut, one can pass just a chart to \texttt{display}. It is then understood that the expansion is to be performed with respect to the coordinate frame associated with this chart. Therefore the above command can be abridged to:

\texttt{sage: t.display(c_uv.restrict(W))}
\[ t = \frac{1}{2}u + \frac{1}{2} \frac{\partial}{\partial u} \otimes du + \frac{1}{2}u - \frac{1}{2} \frac{\partial}{\partial u} \otimes dv \\
+ \frac{1}{2}v + \frac{1}{2} \frac{\partial}{\partial v} \otimes du + \frac{1}{2}v - \frac{1}{2} \frac{\partial}{\partial v} \otimes dv \]

and one has:

\texttt{sage: t.display(c_xy)}
\[ t = x \frac{\partial}{\partial x} \otimes dx + \frac{\partial}{\partial x} \otimes dy + y \frac{\partial}{\partial y} \otimes dx \]

\texttt{sage: t.display(c_uv)}
\[ t = \frac{1}{2}u + \frac{1}{2} \frac{\partial}{\partial u} \otimes du + \frac{1}{2}u - \frac{1}{2} \frac{\partial}{\partial u} \otimes dv \\
+ \frac{1}{2}v + \frac{1}{2} \frac{\partial}{\partial v} \otimes du + \frac{1}{2}v - \frac{1}{2} \frac{\partial}{\partial v} \otimes dv \]

\texttt{sage: t.display(c_xy.restrict(W))}
\[ t = x \frac{\partial}{\partial x} \otimes dx + \frac{\partial}{\partial x} \otimes dy + y \frac{\partial}{\partial y} \otimes dx \]

\texttt{sage: t.restrict(W).display(c_uv.restrict(W))}
\[ t = \frac{1}{2}u + \frac{1}{2} \frac{\partial}{\partial u} \otimes du + \frac{1}{2}u - \frac{1}{2} \frac{\partial}{\partial u} \otimes dv \\
+ \frac{1}{2}v + \frac{1}{2} \frac{\partial}{\partial v} \otimes du + \frac{1}{2}v - \frac{1}{2} \frac{\partial}{\partial v} \otimes dv \]

One can ask for the display with respect to a frame in which \( t \) has not been initialized yet (this will automatically trigger the use of the change-of-frame formula for tensors):

\section*{2.8. Tensor Fields}
sage: a = V.automorphism_field()
sage: a[:] = [[1+v, -u^2], [ 0, 1-u]]
sage: f = e_uv.new_frame(a, 'f')
sage: [f[i].display() for i in M.irange()]
[f_0 = (v + 1) \partial/\partial u, f_1 = -u^2 \partial/\partial u + (-u + 1) \partial/\partial v]

A shortcut of display() is disp():

sage: t.disp(e_uv)
t = (1/2*u + 1/2) \partial/\partial u \otimes du + (1/2*u - 1/2) \partial/\partial u \otimes dv
+ (1/2*v + 1/2) \partial/\partial v \otimes du + (1/2*v - 1/2) \partial/\partial v \otimes dv

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[e,0,0] = - x + y^3
sage: t[e,0,1] = 2+x
sage: t[f,1,1] = - u*v

(continues on next page)
Components in a chart frame:

\[
sage: t.display_comp(chart=c_xy)
\]
\[t^x_x = y^3 - x, \quad t^x_y = x + 2\]

\[
sage: t.display_comp(chart=c_uv)
\]
\[t^v_v = -u*v\]

See documentation of `sage.manifolds.differentiable.tensorfield_paral.TensorFieldParal.display_comp()` for more options.

**div** *(metric=None)*

Return the divergence of *self* (with respect to a given metric).

The divergence is taken on the last index: if *self* is a tensor field *t* of type \((k, 0)\) with \(k \geq 1\), the divergence of \(t\) with respect to the metric *g* is the tensor field of type \((k - 1, 0)\) defined by

\[
(div t)^{a_1 \ldots a_{k-1}} = \nabla_i t^{a_1 \ldots a_{k-1} i},
\]

where \(\nabla\) is the Levi-Civita connection of *g* (cf. `LeviCivitaConnection`).

This definition is extended to tensor fields of type \((k, l)\) with \(k \geq 0\) and \(l \geq 1\), by raising the last index with the metric *g*: \(div t\) is then the tensor field of type \((k, l - 1)\) defined by

\[
(div t)^{a_1 \ldots a_k b_1 \ldots b_{l-1}} = \nabla_i (g^{ij} t^{a_1 \ldots a_k b_1 \ldots b_{l-1} j}),
\]

where \(t^j\) is the tensor field deduced from \(t\) by raising the last index with the metric *g* (see `up()`).

**INPUT:**

- `metric` (default: None) the pseudo-Riemannian metric *g* involved in the definition of the divergence: if none is provided, the domain of *self* is supposed to be endowed with a default metric (i.e. is supposed to be pseudo-Riemannian manifold, see `PseudoRiemannianManifold`) and the latter is used to define the divergence.

**OUTPUT:**

- instance of either `DiffScalarField` if \((k, l) = (1, 0)\) (*self* is a vector field) or \((k, l) = (0, 1)\) (*self* is a 1-form) or of `TensorField` if \(k + l \geq 2\) representing the divergence of *self* with respect to `metric`

**EXAMPLES:**

Divergence of a vector field in the Euclidean plane:

\[
sage: M.<x,y> = EuclideanSpace()
sage: v = M.vector_field(x, y, name='v')
sage: s = v.divergence(); s
Scalar field div(v) on the Euclidean plane E^2
\]

\[
sage: s.display()
\]
\[
\text{div}(v): E^2 \to \mathbb{R}
\]
\[
(x, y) \mapsto 2
\]
A shortcut alias of `divergence` is `div`:

```python
sage: v.div() == s
True
```

The function `div()` from the `operators` module can be used instead of the method `divergence()`:

```python
sage: from sage.manifolds.operators import div
sage: div(v) == s
True
```

The divergence can be taken with respect to a metric tensor that is not the default one:

```python
sage: h = M.lorentzian_metric('h')
sage: h[1,1], h[2,2] = -1, 1/(1+x^2+y^2)
sage: s = v.div(h); s
Scalar field div_h(v) on the Euclidean plane E^2
sage: s.display()
div_h(v): E^2 → ℝ
(x, y) ↦ (x^2 + y^2 + 2)/(x^2 + y^2 + 1)
```

The standard formula

\[
\text{div}_h v = \frac{1}{\sqrt{|\det h|}} \frac{\partial}{\partial x^i} \left( \sqrt{|\det h|} v^i \right)
\]

is checked as follows:

```python
sage: sqrth = h.sqrt_abs_det().expr(); sqrth
1/sqrt(x^2 + y^2 + 1)
sage: s == 1/sqrth * sum( (sqrth*v[i]).diff(i) for i in M.irange())
True
```

A divergence-free vector:

```python
sage: w = M.vector_field(-y, x, name='w')
sage: w.div().display()
div(w): E^2 → ℝ
(x, y) ↦ 0
sage: w.div(h).display()
div_h(w): E^2 → ℝ
(x, y) ↦ 0
```

Divergence of a type-(2,0) tensor field:

```python
sage: t = v*w; t
Tensor field v⊗w of type (2,0) on the Euclidean plane E^2
sage: s = t.div(); s
Vector field div(v⊗w) on the Euclidean plane E^2
sage: s.display()
(div(v⊗w)) = -y e_x + x e_y
```

divergence([metric=None])

Return the divergence of `self` (with respect to a given metric).

The divergence is taken on the last index: if `self` is a tensor field `t` of type `(k, 0)` with `k ≥ 1`, the divergence of `t` with respect to the metric `g` is the tensor field of type `(k − 1, 0)` defined by

\[
(d \text{iv} \, t)^{a_1...a_{k-1}i} = \nabla_i t^{a_1...a_{k-1}i} = (\nabla t)^{a_1...a_{k-1}i}. 
\]
where $\nabla$ is the Levi-Civita connection of $g$ (cf. \texttt{LeviCivitaConnection}).

This definition is extended to tensor fields of type $(k, l)$ with $k \geq 0$ and $l \geq 1$, by raising the last index with the metric $g$: $\div t$ is then the tensor field of type $(k, l-1)$ defined by

$$(\div t)^{a_1 \ldots a_k}_{b_1 \ldots b_{l-1}} = \nabla_i (g^{i j} t^{a_1 \ldots a_k}_{b_1 \ldots b_{l-1} j}) = (\nabla t^i)^{a_1 \ldots a_k}_{b_1 \ldots b_{l-1} i},$$

where $t^i$ is the tensor field deduced from $t$ by raising the last index with the metric $g$ (see \texttt{up()}).

**INPUT:**

- \texttt{metric} (default: None) the pseudo-Riemannian metric $g$ involved in the definition of the divergence; if none is provided, the domain of \texttt{self} is supposed to be endowed with a default metric (i.e. is supposed to be pseudo-Riemannian manifold, see \texttt{PseudoRiemannianManifold}) and the latter is used to define the divergence.

**OUTPUT:**

- instance of either \texttt{DiffScalarField} if $(k, l) = (1, 0)$ (\texttt{self} is a vector field) or $(k, l) = (0, 1)$ (\texttt{self} is a 1-form) or of \texttt{TensorField} if $k + l \geq 2$ representing the divergence of \texttt{self} with respect to \texttt{metric}

**EXAMPLES:**

Divergence of a vector field in the Euclidean plane:

```
sage: M.<x,y> = EuclideanSpace()
sage: v = M.vector_field(x, y, name='v')
sage: s = v.divergence(); s
Scalar field div(v) on the Euclidean plane E^2
sage: s.display()
div(v): E^2 → ℝ
(x, y) ↦ 2
```

A shortcut alias of \texttt{divergence} is \texttt{div}:

```
sage: v.div() == s
True
```

The function \texttt{div()} from the \texttt{operators} module can be used instead of the method \texttt{divergence()}:

```
sage: from sage.manifolds.operators import div
sage: div(v) == s
True
```

The divergence can be taken with respect to a metric tensor that is not the default one:

```
sage: h = M.lorentzian_metric('h')
sage: h[1,1], h[2,2] = -1, 1/(1+x^2+y^2)
sage: s = v.div(h); s
Scalar field div_h(v) on the Euclidean plane E^2
sage: s.display()
div_h(v): E^2 → ℝ
(x, y) ↦ (x^2 + y^2 + 2)/(x^2 + y^2 + 1)
```

The standard formula

$$\div_h v = \frac{1}{\sqrt{\det h}} \frac{\partial}{\partial x^i} \left( \sqrt{\det h} \, v^i \right)$$

is checked as follows:
A divergence-free vector:

```python
sage: w = M.vector_field(-y, x, name='w')
sage: w.display()
div(w): E^2 → ℝ
   (x, y) ↦ 0
sage: w.div(h).display()
div_h(w): E^2 → ℝ
   (x, y) ↦ 0
```

Divergence of a type-\((2,0)\) tensor field:

```python
sage: t = v*w; t
Tensor field v⊗w of type \((2,0)\) on the Euclidean plane E^2
sage: s = t.div(); s
Vector field div(v⊗w) on the Euclidean plane E^2
sage: s.display()
div(v⊗w) = -y e_x + x e_y
```

\section*{domain()}

Return the manifold on which \self \ is defined.

\subsection*{OUTPUT:}

- instance of class \texttt{DifferentiableManifold}

\subsection*{EXAMPLES:}

```python
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 \(M\)
```

\section*{down (non\_degenerate\_form, pos=None)}

Compute a dual of the tensor field by lowering some index with a given non-degenerate form (pseudo-Riemannian metric or symplectic form).

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 \(T^p\) of type \((k-1, l+1)\) whose components are

\[
(T^p)^{a_1...a_{k-1}}_{b_1...b_{l+1}} = g_{b_1}...g_{b_{l+1}} T^{a_1...a_p i a_{p+1}...a_{k-1}}_{b_1...b_{l+1}},
\]

\(g_{ab}\) being the components of the metric tensor or the symplectic form, respectively.

The reverse operation is \texttt{TensorField.up()}.

\subsection*{INPUT:}

- \texttt{non\_degenerate\_form} – \texttt{non\_degenerate\_form} \(g\)
• 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(-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, [[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*(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
(continues on next page)
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: u1tdd = tdd.up(g, 1) ; u1tdd # raising operation, starting from the last index
Tensor field of type (1,1) on the 2-dimensional differentiable manifold M
```

```
sage: uutdd = tdd.up(g) ; uutdd # both indices are raised, starting from the first one
Tensor field of type (2,0) on the 2-dimensional differentiable manifold M
```

```
sage: uutdd == t # should be true
True
```

```
sage: uu0tddd == t # should be true
True
```

```
sage: uu1tddd == t # not true, because of the order of index raising to get
False
```

```
laplacian(metric=None)
```
Return the Laplacian of \texttt{self} with respect to a given metric (Laplace-Beltrami operator).

If \texttt{self} is a tensor field \( t \) of type \((k, l)\), the Laplacian of \( t \) with respect to the metric \( g \) is the tensor field of type \((k, l)\) defined by

\[
(\Delta t)^{a_1 \ldots a_k b_1 \ldots b_k} = \nabla_i \nabla^i t^{a_1 \ldots a_k b_1 \ldots b_k},
\]

where \( \nabla \) is the Levi-Civita connection of \( g \) (cf. \texttt{LeviCivitaConnection}) and \( \nabla^i t := g^{ij} \nabla_j \). The operator \( \Delta = \nabla^i \nabla_i \) is called the \textit{Laplace-Beltrami operator} of metric \( g \).

\textbf{INPUT:}

\begin{itemize}
  \item \texttt{metric} – (default: None) the pseudo-Riemannian metric \( g \) involved in the definition of the Laplacian; if none is provided, the domain of \texttt{self} is supposed to be endowed with a default metric (i.e. is supposed to be pseudo-Riemannian manifold, see \texttt{PseudoRiemannianManifold}) and the latter is used to define the Laplacian
\end{itemize}

\textbf{OUTPUT:}

\begin{itemize}
  \item instance of \texttt{TensorField} representing the Laplacian of \texttt{self}
\end{itemize}

\textbf{EXAMPLES:}

Laplacian of a vector field in the Euclidean plane:

\begin{Verbatim}
sage: M.<x,y> = EuclideanSpace()
sage: v = M.vector_field(x^3 + y^2, x*y, name='v')
sage: Dv = v.laplacian(); Dv
Vector field Delta(v) on the Euclidean plane E^2
sage: Dv.display()
Delta(v) = (6*x + 2) e_x
\end{Verbatim}

The function \texttt{laplacian()} from the \texttt{operators} module can be used instead of the method \texttt{laplacian()}:

\begin{Verbatim}
sage: from sage.manifolds.operators import laplacian
sage: laplacian(v) == Dv
True
\end{Verbatim}

In the present case (Euclidean metric and Cartesian coordinates), the components of the Laplacian are the Laplacians of the components:

\begin{Verbatim}
sage: all(Dv[[i]] == laplacian(v[[i]]) for i in M.irange())
True
\end{Verbatim}

The Laplacian can be taken with respect to a metric tensor that is not the default one:

\begin{Verbatim}
sage: h = M.lorentzian_metric('h')
sage: h[1,1], h[2,2] = -1, 1+x^2
sage: Dv = v.laplacian(h); Dv
Vector field Delta_h(v) on the Euclidean plane E^2
sage: Dv.display()
Delta_h(v) = -(8*x^5 - 2*x^4 - x^2*y^2 + 15*x^3 - 4*x^2 + 6*x - 2)/(x^4 + 2*x^2 + 1) e_x - 3*x^3*y/(x^4 + 2*x^2 + 1) e_y
\end{Verbatim}

\texttt{lie\_der}(\texttt{vector})

Lie derivative of \texttt{self} with respect to a vector field.

\textbf{INPUT:}

\begin{itemize}
  \item \texttt{vector} – vector field with respect to which the Lie derivative is to be taken
\end{itemize}
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 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: e_xy = c_xy.frame(); e_uv = c_uv.frame()
sage: t = M.tensor_field(1, 1, {e_xy: [[x, 1], [y, 0]]}, name='t')
sage: t.add_comp_by_continuation(e_uv, U.intersection(V), c_uv)
sage: w = M.vector_field({e_xy: [ -y, x]}, name='w')
w.add_comp_by_continuation(e_uv, U.intersection(V), c_uv)
sage: lt = t.lie_derivative(w)
Tensor field of type (1,1) on the 2-dimensional differentiable manifold M
sage: lt.display(e_xy)
∂/∂x⊗dx - x ∂/∂x⊗dy + (-y - 1) ∂/∂y⊗dy
sage: lt.display(e_uv)
-1/2*u ∂/∂u⊗du + (1/2*u + 1) ∂/∂v⊗du + 1/2*v ∂/∂u⊗dv + (1/2*v + 1) ∂/∂v⊗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({e_xy: [1-x, x-y]}, name='a')
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 ∂/∂x + (-y - 1) ∂/∂y
sage: a.lie_der(w).display(e_uv)
(v - 1) ∂/∂u + (u + 1) ∂/∂v
```

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:
Manifolds, Release 10.3

```
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 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: e_xy = c_xy.frame(); e_uv = c_uv.frame()
sage: t = M.tensor_field(1, 1, {e_xy: [[x, 1], [y, 0]]}, name='t')
sage: t.add_comp_by_continuation(e_uv, U.intersection(V), c_uv)
sage: w = M.vector_field({e_xy: [-y, x]}, name='w')
sage: w.add_comp_by_continuation(e_uv, U.intersection(V), c_uv)
sage: lt = t.lie_derivative(w)
sage: lt.display(e_xy)
∂/∂x⊗dx - x ∂/∂x⊗dy + (-y - 1) ∂/∂y⊗dy
sage: lt.display(e_uv)
-1/2*u ∂/∂u⊗du + (1/2*u + 1) ∂/∂u⊗dv + (-1/2*v + 1) ∂/∂v⊗du + 1/2*v ∂/∂v⊗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({e_xy: [1-x, x-y]}, name='a')
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 ∂/∂x + (-y - 1) ∂/∂y
sage: a.lie_der(w).display(e_uv)
(1-x) ∂/∂u + (u + 1) ∂/∂v
```

2.8. Tensor Fields 655
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 $\Psi : U \to V$, 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 $S \to M$ 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({eN: [ 1, 0]}, name='v')
sage: v.display()
v = ∂/∂x
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 = ∂/∂x
sage: vU == eN[1]
```

(continues on next page)
At this stage, defining the restriction of $v$ to the open subset $V$ fully specifies $v$:

```python
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) \partial/\partial u - 2*u*v \partial/\partial v
```

The restriction of the vector field to its own domain is of course itself:

```python
sage: v.restrict(M) is v
True
sage: vU.restrict(U) is vU
True
```

### set_calc_order(symbol, order, truncate=False)

Trigger a series expansion with respect to a small parameter in computations involving the tensor field.

This property is propagated by usual operations. The internal representation must be SR for this to take effect.

If the small parameter is $\epsilon$ and $T$ is `self`, the power series expansion to order $n$ is

$$T = T_0 + \epsilon T_1 + \epsilon^2 T_2 + \cdots + \epsilon^n T_n + O(\epsilon^{n+1}),$$

where $T_0, T_1, \ldots, T_n$ are $n+1$ tensor fields of the same tensor type as `self` and do not depend upon $\epsilon$.

**INPUT:**

- `symbol` – symbolic variable (the “small parameter” $\epsilon$) with respect to which the components of `self` are expanded in power series
- `order` – integer; the order $n$ of the expansion, defined as the degree of the polynomial representing the truncated power series in `symbol`
- `truncate` – (default: `False`) determines whether the components of `self` are replaced by their expansions to the given order

**EXAMPLES:**

Let us consider two vector fields depending on a small parameter $h$ on a non-parallelizable manifold:

```python
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
```
If we set the calculus order on one of the vector fields, any operation involving both of them is performed to that order:

```sage
sage: a.set_calc_order(h, 2)
sage: s = a + b
sage: s[eU, :]
[h^2 + 2, 1/2*h^2*y^2 + h*y - y + 1]
sage: s[eV, :]
[1/8*(u^2 - 2*u*v + v^2)*h^2 + h*u - 1/2*u + 1/2*v + 3,
-1/8*(u^2 - 2*u*v + v^2)*h^2 + h*v + 1/2*u - 1/2*v + 1]
```

Note that the components of \( a \) have not been affected by the above call to \( \text{set
calc\_order} \):

```sage
sage: a[eU, :]
[cos(h*x), -y]
sage: a[eV, :]
[cos(1/2*h*u)*cos(1/2*h*v) - sin(1/2*h*u)*sin(1/2*h*v) - 1/2*u + 1/2*v,
cos(1/2*h*u)*cos(1/2*h*v) - sin(1/2*h*u)*sin(1/2*h*v) + 1/2*u - 1/2*v]
```

To have \( \text{set
calc\_order} \) act on them, set the optional argument \( \text{truncate} \) to \( \text{True} \):

```sage
sage: a.set_calc_order(h, 2, truncate=True)
sage: a[eU, :]
[-1/2*h^2*x^2 + 1, -y]
sage: a[eV, :]
[-1/8*(u^2 + 2*u*v + v^2)*h^2 - 1/2*u + 1/2*v + 1,
-1/8*(u^2 + 2*u*v + v^2)*h^2 + 1/2*u - 1/2*v + 1]
```

\textbf{set\_comp(} \textit{basis}=\textit{None})

Return the components of \textit{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 \( \text{add\_comp()} \) instead.

\textbf{INPUT:}

- \textit{basis} – (default: \text{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

\textbf{OUTPUT:}

- components in the given frame, as a \textbf{Components}; if such components did not exist previously, they are created

\textbf{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, (∂/∂u,∂/∂v))
sage: t.set_comp(e_uv)[1,0,1] = u+v
sage: t.display(e_uv)
t = (u + v) ∂/∂v⊗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, (∂/∂u,∂/∂v))

Since zero is an immutable, its components cannot be changed:

sage: z = M.tensor_field_module((1, 1)).zero()
sage: z.set_comp(e)[0,1] = u*v
Traceback (most recent call last):
  ... ValueError: the components of an immutable element cannot be
changed

\textbf{set\_immutable}()

Set \texttt{self} and all restrictions of \texttt{self} immutable.

\textbf{EXAMPLES:}

sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: U = M.open_subset('U', coord_def={X: x^2+y^2<1})
sage: a = M.tensor_field(1, 1, [[1+y,x], [0,x+y]], name='a')
sage: aU = a.restrict(U)
sage: a.set_immutable()

\textbf{set\_name} \((name=None, latex_name=None)\)

Set (or change) the text name and \LaTeX\ name of \texttt{self}.

\textbf{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:

```python
sage: M = Manifold(2, 'M')
sage: t = M.tensor_field(1, 3); t
Tensor field t 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:

```python
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) ∂/∂x⊗dx⊗dy
sage: t.restrict(U) == s
True

If the restriction is defined on the very same domain, the tensor field becomes a copy of it (see **copy_from()**):

```python
sage: v = M.tensor_field(1, 2, name='v')
sage: v.set_restriction(t)
sage: v.restrict(U) == t.restrict(U)
True
```
**symmetries()**

Print the list of symmetries and antisymmetries.

**EXAMPLES:**

```sage
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
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()
....:
    restrictions1= x>0, restrictions2= u+v>0)
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, {eU: [[ 1,x], [ 2,y]], name='a'})
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 -1/4*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, using=None)
Trace (contraction) on two slots of the tensor field.
If a non-degenerate form is provided, the trace of a \((0, 2)\) tensor field is computed by first raising the last index.

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
• \using\ – (default: None) a non-degenerate form

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()
    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')
    a[e_xy,:] = [[1,x], [ 2,y]]
    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

The trace of a type-(0, 2) tensor field using a metric:

sage: g = M.metric('g')
    g[0,0], g[0,1], g[1,1] = 1, 0, 1
sage: g.trace(using=g).display()
M → ℝ
on U: (x, y) ↦ 2
on V: (u, v) ↦ 2

Instead of the explicit call to the method \texttt{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]]]
    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

2.8. Tensor Fields 663
Use of the index notation:

\begin{verbatim}
sage: b[^k_ki]
s 1-form on the 2-dimensional differentiable manifold M
sage: b[^k_ki] == s  # long time
True
\end{verbatim}

Indices not involved in the contraction may be replaced by dots:

\begin{verbatim}
sage: b[^k_k.] == s  # long time
True
\end{verbatim}

The symbol ^ may be omitted:

\begin{verbatim}
sage: b[k_k.] == s  # long time
True
\end{verbatim}

LaTeX notations are allowed:

\begin{verbatim}
sage: b[^{k}_{ki}] == s  # long time
True
\end{verbatim}

Contraction on first and third slots:

\begin{verbatim}
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
\end{verbatim}

Use of index notation:

\begin{verbatim}
sage: b[^k_.k] == s  # long time
True
\end{verbatim}

\texttt{up} (non_degenerate_form, pos=None)

Compute a dual of the tensor field by raising some index with the given tensor field (usually, a pseudo-Riemannian metric, a symplectic form or a Poisson tensor).

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 $\text{pos} = p$ yields the tensor field $T^\#$ of type $(k + 1, l - 1)$ whose components are

$$(T^\#)_{a_1...a_{k+1}}^{b_1...b_{l-1}} = g_{a_1}^{a_{k+1}} T_{a_1...a_k}^{b_1...b_{p-k}} g_{b_{p-k+1}}^{b_{l-1}},$$

$g_{ab}$ being the components of the inverse metric or the Poisson tensor, respectively.

The reverse operation is \texttt{TensorField.down()}.

\textbf{INPUT:}

- non_degenerate_form – non-degenerate form $g$, or a Poisson tensor
- 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

\textbf{OUTPUT:}

- the tensor field $T^\#$ resulting from the index raising operation
EXAMPLES:

Raising the index of a 1-form results in a vector field:

```sage
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(-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) ∂/∂x
-(x*y + 2*x + 2)/(x^2*y^2 + (x + 1)*y - x - 1) ∂/∂y
```

Using the index notation instead of `up()`:

```sage
sage: v == ig[^ab]*w[_b]
True
```

The reverse operation:

```sage
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
sage: g[^ab]*v[^b] == w
True
```

Raising the indices of a tensor field of type (0,2):

```sage
sage: t = M.tensor_field(0, 2, [[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
```

(continues on next page)
\[
sage: \text{tu1 = t.up(g, 1) ; tu1} \ # \text{raising the second index}
\]  
Tensor field of type (1,1) on the 2-dimensional differentiable manifold \(M\)

\[
sage: \text{tu1 == ig['\text{ac}']\*t['\text{bc}']} \ # \text{the same operation in index notation}
\]

\[
\text{True}
\]

\[
sage: \text{tu1[:]} \\
\frac{(2*x + 1)*y - 1)/(x^2*y^2 + (x + 1)*y - x - 1)}{(x + 1)*y - x - 1} \\
\frac{(4*x + 3)*y - 3)/(x^2*y^2 + (x + 1)*y - x - 1)}{(x + 1)*y - x - 1}
\]

\[
sage: \text{tuu1 = tu1.up(g) ; tuu1} \ # \text{the two indices have been raised, starting from the second one}
\]

Tensor field of type (2,0) on the 2-dimensional differentiable manifold \(M\)

\[
sage: \text{tuu1 == tu1['\text{a_c}']*ig['\text{cb}']} \ # \text{the same operation in index notation}
\]

\[
\text{True}
\]

\[
sage: \text{tuu0 == tuu1} \ # \text{the order of index raising is important}
\]

\[
\text{False}
\]

\[
sage: \text{tuu = t.up(g) ; tuu} \ # \text{both indices are raised, starting from the first one}
\]

Tensor field of type (2,0) on the 2-dimensional differentiable manifold \(M\)

\[
sage: \text{tuu0 == tuu} \ # \text{the same order for index raising has been applied}
\]

\[
\text{True}
\]

\[
sage: \text{tuu1 == tuu} \ # \text{to get tuu1, indices have been raised from the last one, contrary to tuu}
\]

\[
\text{False}
\]

\[
sage: \text{d0tuu = tuu.down(g, 0) ; d0tuu} \ # \text{the first index is lowered again}
\]

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

\[
sage: \text{dd0tuu = d0tuu.down(g)} \ ; \ dd0tuu \ # \text{the second index is then lowered}
\]

Tensor field of type (0,2) on the 2-dimensional differentiable manifold \(M\)

\[
sage: \text{dituu = tuu.down(g, 1) ; dituu} \ # \text{lowering operation, starting from the last index}
\]

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

\[
sage: \text{dd1tuu = dituu.down(g)} \ ; \ dd1tuu
\]

Tensor field of type (0,2) on the 2-dimensional differentiable manifold \(M\)

\[
sage: \text{dddtuu = dituu.down(g)} \ ; \ ddtuu \ # \text{both indices are lowered, starting from the last one}
\]

Tensor field of type (0,2) on the 2-dimensional differentiable manifold \(M\)

\[
sage: \text{dddtuu == t} \ # \text{should be true}
\]

\[
\text{True}
\]

\[
sage: \text{dd0ttuu == t} \ # \text{not true, because of the order of index lowering to get dd0ttuu}
\]

\[
\text{False}
\]

\[
sage: \text{dd1ttuu == t} \ # \text{should be true}
\]

\[
\text{True}
\]
2.8.3 Tensor Fields with Values on a Parallelizable Manifold

The class \texttt{TensorFieldParal} implements tensor fields along a differentiable manifolds with values on a parallelizable differentiable manifold. For non-parallelizable manifolds, see the class \texttt{TensorField}.

Various derived classes of \texttt{TensorFieldParal} are devoted to specific tensor fields:

- \texttt{VectorFieldParal} for vector fields (rank-1 contravariant tensor fields)
- \texttt{AutomorphismFieldParal} for fields of tangent-space automorphisms
- \texttt{DiffFormParal} for differential forms (fully antisymmetric covariant tensor fields)
- \texttt{MultivectorFieldParal} for multivector fields (fully antisymmetric contravariant tensor fields)

AUTHORS:

- Travis Scrimshaw (2016): review tweaks
- Eric Gourgoulhon (2018): method \texttt{TensorFieldParal.along()}
- Florentin Jaffredo (2018): series expansion with respect to a given parameter

REFERENCES:

- [KN1963]
- [Lee2013]
- [ONe1983]

EXAMPLES:

A tensor field of type (1, 1) on a 2-dimensional differentiable manifold:

\begin{verbatim}
sage: M = Manifold(2, 'M', start_index=1)
sage: c_xy.<x,y> = M.chart()
sage: t = M.tensor_field(1, 1, name='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
\end{verbatim}

Components with respect to the manifold's default frame are created by providing the relevant indices inside square brackets:

\begin{verbatim}
sage: t[1,1] = x^2
\end{verbatim}

Unset components are initialized to zero:

\begin{verbatim}
sage: t[:]
[x^2 0]
[ 0 0]
\end{verbatim}

It is also possible to initialize the components at the tensor field construction:

\begin{verbatim}
sage: t = M.tensor_field(1, 1, [[x^2, 0], [0, 0]], name='T')
sage: t[:]
[x^2 0]
[ 0 0]
\end{verbatim}
The full set of components with respect to a given vector frame is returned by the method \texttt{comp()}:

\begin{Verbatim}
\texttt{sage: t.comp(c_xy.frame())}
\end{Verbatim}

2-indices components w.r.t. Coordinate frame (M, (∂/∂x, ∂/∂y))

If no vector frame is mentioned in the argument of \texttt{comp()}, it is assumed to be the manifold’s default frame:

\begin{Verbatim}
\texttt{sage: M.default_frame()}
\texttt{Coordinate frame (M, (∂/∂x, ∂/∂y))}
\texttt{sage: t.comp() is t.comp(c_xy.frame())}
\end{Verbatim}

True

Individual components with respect to the manifold’s default frame are accessed by listing their indices inside double square brackets. They are \textit{scalar fields} on the manifold:

\begin{Verbatim}
\texttt{sage: t[[1,1]]}
\texttt{Scalar field on the 2-dimensional differentiable manifold M}
\texttt{sage: t[[1,1]].display()}
M → R
(x, y) ↦ x^2
\texttt{sage: t[[1,2]]}
\texttt{Scalar field zero on the 2-dimensional differentiable manifold M}
\texttt{sage: t[[1,2]].display()}
zero: M → R
(x, y) ↦ 0
\end{Verbatim}

A direct access to the coordinate expression of some component is obtained via the single square brackets:

\begin{Verbatim}
\texttt{sage: t[1,1]}
x^2
\texttt{sage: t[1,1] is t[[1,1]].coord_function() # the coordinate function}
True
\texttt{sage: t[1,1] is t[[1,1]].coord_function(c_xy)}
True
\texttt{sage: t[1,1].expr() is t[[1,1]].expr() # the symbolic expression}
True
\end{Verbatim}

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:

\begin{Verbatim}
\texttt{sage: c_uv.<u,v> = M.chart()}
\texttt{sage: xy_to_uv = c_xy.transition_map(c_uv, [x+y, x-y])}
\texttt{sage: uv_to_xy = xy_to_uv.inverse()}
\texttt{sage: t[1,1, c_uv]}
1/4*u^2 + 1/2*u*v + 1/4*v^2
\end{Verbatim}

Note that \texttt{t[1,1, c_uv]} is the component of the tensor \texttt{t} with respect to the coordinate frame associated to the chart \((x, y)\) expressed in terms of the coordinates \((u, v)\). Indeed, \texttt{t[1,1, c_uv]} is a shortcut for \texttt{t.comp(c_xy.frame())[[1,1]].coord_function(c_uv)}:

\begin{Verbatim}
\texttt{sage: t[1,1, c_uv] is t.comp(c_xy.frame())[[1,1]].coord_function(c_uv)}
True
\end{Verbatim}

Similarly, \texttt{t[1,1]} is a shortcut for \texttt{t.comp(c_xy.frame())[[1,1]].coord_function(c_xy)}:

\begin{Verbatim}
\texttt{sage: t[1,1] is t.comp(c_xy.frame())[[1,1]].coord_function(c_xy)}
True
\end{Verbatim}
All the components can be set at once via [:]:

```sage
t[:]=[[1,-x],[x*y,2]]
t[:]
```

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')
t.set_comp(e)[1,1] = x+y
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
t[e,2,1], t[e,2,2] = y, -3*x
t.comp(e)
```

All the components in some frame can be set at once, via the operator [:]:

```sage
t[e,:] = [[[x+y, 0], [y, -3*x]]
t[e,:] # same as above:
```

Equivalently, one can initialize the components in e at the tensor field construction:

```sage
t = M.tensor_field(1, 1, [[x+y, 0], [y, -3*x]], frame=e, name='T')
t[e,:] # same as above:
```

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, name='T') # Let us restart
t[:] = [[[1, -x], [x*y, 2]] # by first setting the components in the frame c_xy.
```

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`: 2.8. Tensor Fields
sage: t.display()  # expansion in the manifold's default frame
T = ∂/∂x⊗dx - x ∂/∂x⊗dy + x*y ∂/∂y⊗dx + 2 ∂/∂y⊗dy
sage: t.display(e)
T = (x + y) e_1⊗e^1 + y e_2⊗e^1 - 3*x e_2⊗e^2

See \texttt{display()} for more examples.

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(1, x, name='a')
sage: v = M.vector_field(y, 2, name='V')
sage: t(a,v)
Scalar field \(T(a,V)\) on the 2-dimensional differentiable manifold \(M\)

sage: t(a,v).display()
\(T(a,V)\): \(M \rightarrow \mathbb{R}\)
\((x, y) \mapsto x^2*y^2 + 2*x + y\)
\((u, v) \mapsto 1/16*u^4 - 1/8*u^2*v^2 + 1/16*v^4 + 3/2*u + 1/2*v\)

sage: latex(t(a,v))
\(T(a,V)\)

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] \
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 \texttt{DiffScalarField} for more details on scalar fields.

A vector field (rank-1 contravariant tensor field):

sage: v = M.vector_field(-x, y, name='V'); v
Vector field \(v\) on the 2-dimensional differentiable manifold \(M\)

sage: v.tensor_type()
\((1, 0)\)

sage: v.display()
v = -x ∂/∂x + y ∂/∂y

A field of symmetric bilinear forms:

sage: q = M.sym_bilin_form_field(name='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
definitions
q[1,1], q[1,2], q[2,2] = (0, -x, y) # no need to set the component (2,1)
type(q.comp())
<class 'sage.tensor.modules.comp.CompFullySym'>
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
definitions
t = M.tensor_field(0, 4, name='T', sym=(0,1), antisym=(2,3))
t[1,2,1,2] = 3
t[2,1,1,2] # check of the symmetry with respect to the first 2 indices
3
t[1,2,2,1] # check of the antisymmetry with respect to the last 2 indices
-3
```

```python
class sage.manifolds.differentiable.tensorfield_paral.TensorFieldParal(
  vector_field_module,
  tensor_type,
  name=None,
  latex_name=None,
  sym=None,
  anti_sym=None)
```

Bases: `FreeModuleTensor`, `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

$$\Phi : U \rightarrow M,$$

a `tensor field of type $(k,l)$ along $U $ with values on $M` is a differentiable map

$$t : U \rightarrow T^{(k,l)}M$$

(where $T^{(k,l)}M$ is the tensor bundle of type $(k,l)$ over $M$) such that

$$t(p) \in T^{(k,l)}(T_q M)$$

for all $p \in U$, i.e. $t(p)$ is a tensor of type $(k,l)$ on the tangent space $T_q M$ at the point $q = \Phi(p)$. That is to say a multilinear map

$$t(p) : T^*_q M \times \cdots \times T^*_q M \times T_q M \times \cdots \times T_q M \rightarrow K,$$

2.8. Tensor Fields
where \( T^*_q M \) is the dual vector space to \( T_q M \) 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 \( \mathbb{R} \)).

**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 \( \Phi : U \to M \) (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, name='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
```

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
e = M.vector_frame('e') # a new frame defined on M, in addition to e
t.set_comp(e)[0,0] = -3
t.comp(e)
2-indices components w.r.t. Vector frame (M, (e_0,e_1,e_2))
t.comp(e)[0,0]
-3
t.comp(e)[:]
# 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, (e_0,e_1,e_2)): 2-indices components w.r.t. Vector frame (M, (e_0,e_1,e_2))}
```

To keep the other components, one must use the method `add_comp()`:

```sage
t = M.tensor_field(2, 0, name='T') # let us restart
t[0,0] = 2 # sets the components in the frame e
t.add_comp(f)[0,0] = -3
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
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:
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:

```python
sage: r = M.tensor_field(0, 4, name='T', antisym=[(0,1), (2,3)])
sage: r[0,1,2,0] = 3
sage: r[0,1,2,0]  # 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:

```python
sage: a = M.tensor_field(2, 0)
sage: a[0,0], a[1,1], a[2,2] = (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] [4 0 7] [ 5 2 10]
[0 0 0] [0 10 0] [ 2 5 0]
[0 0 0], [0 0 12], [ 3 0 6])
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:

```python
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()
(continues on next page)
```
Let us now make \( b \) symmetric:

\[
\begin{aligned}
\text{sage: } & b = M.tensor_field(2, 0, \text{sym}=(0,1)) \\
& b[0,0], \, b[1,1], \, b[2,2], \, b[0,2] = (4,5,6,7) \\
\text{sage: } & a = a + b \\
\end{aligned}
\]

\[
\begin{aligned}
\text{sage: } & s = a + b \\
\text{sage: } & a[:], \, b[:], \, s[:] \\
& (\begin{bmatrix} 1 & 2 & 3 \\ 2 & 0 & 0 \\ 3 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 4 & 0 & 7 \\ 0 & 5 & 0 \\ 7 & 0 & 6 \end{bmatrix}, \begin{bmatrix} 2 & 10 \\ 2 & 5 \\ 10 & 6 \end{bmatrix}) \\
\text{sage: } & s.\text{symmetries()} \quad \# \text{s is symmetric because both } a \text{ and } b \text{ are} \\
& \text{symmetry: } (0, 1); \text{ no antisymmetry} \\
\end{aligned}
\]

The tensor product is taken with the operator `*`:

\[
\begin{aligned}
\text{sage: } & c = a*b ; \, c \\
\end{aligned}
\]

Tensor field of type (4,0) on the 3-dimensional differentiable manifold \( M \)

\[
\text{sage: } c.\text{symmetries()} \quad \# \text{since } a \text{ and } b \text{ are both symmetric, } a*b \text{ has two symmetries:} \\
& \text{symmetries: } [(0, 1), (2, 3)]; \text{ no antisymmetry} \\
\]

The tensor product of two fully contravariant tensors is not symmetric in general:

\[
\text{sage: } a*b == b*a \\
\text{False} \\
\]

The tensor product of a fully contravariant tensor by a fully covariant one is symmetric:

\[
\begin{aligned}
\text{sage: } & d = M.\text{diff_form}(2) \quad \# \text{a fully covariant tensor field} \\
\text{sage: } & d[0,1], \, d[0,2], \, d[1,2] = (3, 2, 1) \\
\text{sage: } & s = a*d ; \, s \\
\end{aligned}
\]

Tensor field of type (2,2) on the 3-dimensional differentiable manifold \( M \)

\[
\text{sage: } s.\text{symmetries()} \\
& \text{symmetry: } (0, 1); \text{ antisymmetry: } (2, 3) \\
\text{sage: } s1 = d*a ; \, s1 \\
\end{aligned}
\]

Tensor field of type (2,2) on the 3-dimensional differentiable manifold \( M \)

\[
\text{sage: } s1.\text{symmetries()} \\
& \text{symmetry: } (0, 1); \text{ antisymmetry: } (2, 3) \\
\text{sage: } d*a == a*d \\
\text{True} \\
\]

Example of tensor field associated with a non-trivial differentiable map \( \Phi \): tensor field along a curve in \( M \):

\[
\begin{aligned}
\text{sage: } & R = \text{Manifold}(1, 'R') \quad \# \text{R as a 1-dimensional manifold} \\
\text{sage: } & T.<t> = R.chart() \quad \# \text{canonical chart on } R \\
\text{sage: } & \Phi = R.\text{diff_map}(M, [\cos(t), \sin(t), t], \text{name}='\Phi') ; \Phi \\
\end{aligned}
\]

(continues on next page)
Manifolds, Release 10.3

(continued from previous page)

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) ∂/∂x⊗∂/∂x + t^2 ∂/∂x⊗∂/∂y + sin(t) ∂/∂z⊗∂/∂x

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, 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, (∂/∂x,∂/∂y))
sage: t.add_comp(e_xy)[1,0] = 2
sage: t.display(e_xy)
t = 2 ∂/∂y⊗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 ∂/∂y⊗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')
The components previously defined are kept:

\[ t = (y + 1) f_0 \otimes f^1 \]

**along** *(mapping)*

Return the tensor field deduced from \( \text{self} \) via a differentiable map, the codomain of which is included in the domain of \( \text{self} \).

More precisely, if \( \text{self} \) is a tensor field \( t \) on \( M \) and if \( \Phi : U \rightarrow M \) is a differentiable map from some differentiable manifold \( U \) to \( M \), the returned object is a tensor field \( \tilde{t} \) along \( U \) with values on \( M \) such that

\[ \forall p \in U, \tilde{t}(p) = t(\Phi(p)). \]

**INPUT:**

- mapping – differentiable map \( \Phi : U \rightarrow M \)

**OUTPUT:**

- tensor field \( \tilde{t} \) along \( U \) defined above.

**EXAMPLES:**

Let us consider the map \( \Phi \) between the interval \( U = (0, 2\pi) \) and the Euclidean plane \( M = \mathbb{R}^2 \) defining the lemniscate of Gerono:

\[
\begin{align*}
\text{sage: } & M = \text{Manifold(2, 'M')} \\
\text{sage: } & X.<x,y> = M.chart() \\
\text{sage: } & t = \text{var('t', domain='real')} \\
\text{sage: } & \Phi = M.curve({X: [\sin(t), \sin(2*t)/2]}, (t, 0, 2*pi), \\
& \hspace{1cm} \text{name='\Phi')} \\
\text{sage: } & U = \Phi.domain(); U \\
& \text{Real interval (0, 2*pi)}
\end{align*}
\]

and a vector field on \( M \):

\[
\text{sage: } v = M.\text{vector_field}(-y, x, \text{name='v')} 
\]

We have then:

\[
\begin{align*}
\text{sage: } & vU = v.\text{along}(\Phi); vU \\
& \text{Vector field v along the Real interval (0, 2*pi) with values on} \\
& \text{the 2-dimensional differentiable manifold M} \\
\text{sage: } & vU.\text{display()} \\
& v = -\cos(t)\sin(t) \partial/\partial x + \sin(t) \partial/\partial y \\
\text{sage: } & vU.\text{parent()} \\
& \text{Free module X((0, 2*pi),\Phi) of vector fields along the Real} \\
& \text{interval (0, 2*pi) mapped into the 2-dimensional differentiable} \\
& \text{manifold M}
\end{align*}
\]
We check that the defining relation \( \tilde{t}(p) = t(\Phi(p)) \) holds:

```python
sage: p = U(t); # a generic point of U
sage: vU.at(p) == v.at(Phi(p))
True
```

Case of a tensor field of type \((0,2)\):

```python
sage: a = M.tensor_field(0, 2)
sage: a[0,0], a[0,1], a[1,1] = x+y, x*y, x^2-y^2
sage: aU = a.along(Phi); aU
Tensor field of type (0,2) along the Real interval (0, 2*pi) with values on the 2-dimensional differentiable manifold M
sage: aU.display()
(\cos(t) + 1)\sin(t) \, dx \otimes dx + \cos(t)\sin(t)^2 \, dx \otimes dy + \sin(t)^4 \, dy \otimes dy
sage: aU.parent()
Free module T^{0,2}((0, 2*pi),\Phi) of type-(0,2) tensors fields along the Real interval (0, 2*pi) mapped into the 2-dimensional differentiable manifold M
sage: aU.at(p) == a.at(Phi(p))
True
```

**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* \( p \) 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:

```python
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(y, x^2, name='v')
sage: v.display()
v = y \partial/\partial x + x^2 \partial/\partial y
sage: vp = v.at(p) ; vp
```
Tangent vector $v$ at Point $p$ on the 2-dimensional differentiable manifold $M$

```python
sage: vp.parent()
```

Tangent space at Point $p$ on the 2-dimensional differentiable manifold $M$

```python
sage: vp.display()
```

$v = 3 \partial/\partial x + 4 \partial/\partial y$

A 1-form gives birth to a linear form in the tangent space:

```python
sage: w = M.one_form(-x, 1+y, name='w')
sage: w.display()
w = -x \, dx + (y + 1) \, dy
```

Linear form $w$ on the Tangent space at Point $p$ on the 2-dimensional differentiable manifold $M$

```python
sage: wp = w.at(p); wp
```

Dual of the Tangent space at Point $p$ on the 2-dimensional differentiable manifold $M$

```python
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:

```python
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) \partial/\partial x \otimes dx + x*y \partial/\partial x \otimes dy + (-y + 1) \partial/\partial y \otimes dy
```

Free module of type-(1,1) tensors on the Tangent space at Point $p$ on the 2-dimensional differentiable manifold $M$

```python
sage: tp = t.at(p); tp
```

A 2-form yields an alternating form of degree 2 in the tangent space:

```python
sage: a = M.diff_form(2, name='a')
sage: a[0,1] = x*y
sage: a.display()
a = x*y \, dx \wedge dy
```

Alternating form $a$ of degree 2 on the Tangent space at Point $p$ on the 2-dimensional differentiable manifold $M$

```python
sage: ap = a.at(p); ap
```

Example with a non-trivial map $\Phi$:

```python
sage: U = Manifold(1, 'U')  # (0,2*pi) as a 1-dimensional manifold
sage: T.<t> = U.chart(rt:(0,2*pi))  # canonical chart on U
sage: Phi = U.diff_map(M, [cos(t), sin(t)], name='Phi'),
```

(continues on next page)
....: latex_name=r'\Phi')
sage: v = U.vector_field(1+t, t^2, 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.display()
v = (t + 1) \partial/\partial x + t^2 \partial/\partial y
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) \partial/\partial x + 1/36*pi^2 \partial/\partial y

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 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
t: 3-indices components w.r.t. Coordinate frame (M, (\partial/\partial x,\partial/\partial y))
sage: t.comp() # the default frame is X.frame()
t: 3-indices components w.r.t. Coordinate frame (M, (\partial/\partial x,\partial/\partial y))
sage: t.comp([]) [[0, 0], [x*y, 0]], [[0, 0, 0], [0, 0]]
sage: e = M.vector_frame('e')
sage: t[e, 2,1,1] = x-3
t: 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: a = M.tensor_field(2,0, [[1+x, 2], [y, -x^2]], name='a')
sage: b = M.tensor_field(1,1, [-y, 1], [x, x+y], name='b')
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 ∂/∂x⊗∂/∂x + (x^2 + x*y + y^2 + x) ∂/∂x⊗∂/∂y
+ (-x^2 - 2*y) ∂/∂y⊗∂/∂x + (-x^3 - x^2*y + 2*x) ∂/∂y⊗∂/∂y
```

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
```
```
sage: s.display()
-(-x + 1)*y + 2) ∂/∂x⊗∂/∂x + (x^2 + 3*x + 2*y) ∂/∂x⊗∂/∂y
+ (-x^2 - y^2) ∂/∂y⊗∂/∂x + (-x^3 - (x^2 - x)*y) ∂/∂y⊗∂/∂y
```

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 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:
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:

```python
sage: M = Manifold(2, 'M', start_index=1)
sage: c_xy.<x,y> = M.chart()
sage: v = M.vector_field(-y, x, name='v')
sage: w = M.vector_field(2*x+y, x*y)
sage: w.lie_derivative(v)
Vector field on the 2-dimensional differentiable manifold M
evaluating on 1-form:

sage: w.lie_derivative(v).display()
((x - 2)*y + x) \partial/\partial x + (x^2 - y^2 - 2*x - y) \partial/\partial y
```

The result is cached:

```python
sage: w.lie_derivative(v) is w.lie_derivative(v)
True
```

An alias is `lie_der`:

```python
sage: w.lie_der(v) is w.lie_derivative(v)
True
```

The Lie derivative is antisymmetric:

```python
sage: w.lie_der(v) == -v.lie_der(w)
True
```

For vectors, it coincides with the commutator:

```python
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:

```python
sage: om = M.one_form(y^2*sin(x), x^3*cos(y))
sage: om.lie_derivative(v)
1-form on the 2-dimensional differentiable manifold M
evaluating on 1-form:

sage: om.lie_derivative(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:

```python
sage: Parallelism().set('tensor', nproc=2)
sage: om.lie_derivative(v)
1-form on the 2-dimensional differentiable manifold M
evaluating on 1-form:

sage: om.lie_derivative(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
```

Check of Cartan identity:
\textbf{lie_derivative (vector)}

Compute the Lie derivative with respect to a vector field.

**INPUT:**
- \texttt{vector} – vector field with respect to which the Lie derivative is to be taken

**OUTPUT:**
- the tensor field that is the Lie derivative of \texttt{self} with respect to \texttt{vector}

**EXAMPLES:**

Lie derivative of a vector:

\begin{verbatim}
sage: M = Manifold(2, 'M', start_index=1)
sage: c_xy.<x,y> = M.chart()
sage: v = M.vector_field(-y, x, name='v')
sage: w = M.vector_field(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) \partial/\partial x + (x^2 - y^2 - 2*x - y) \partial/\partial y
\end{verbatim}

The result is cached:

\begin{verbatim}
sage: w.lie_derivative(v) is w.lie_derivative(v)
True
\end{verbatim}

An alias is \texttt{lie_der}:

\begin{verbatim}
sage: w.lie_der(v) is w.lie_derivative(v)
True
\end{verbatim}

The Lie derivative is antisymmetric:

\begin{verbatim}
sage: w.lie_der(v) == -v.lie_der(w)
True
\end{verbatim}

For vectors, it coincides with the commutator:

\begin{verbatim}
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
\end{verbatim}

Lie derivative of a 1-form:

\begin{verbatim}
sage: om = M.one_form(y^2*sin(x), x^3*cos(y))
sage: om.lie_derivative(v)
1-form on the 2-dimensional differentiable manifold M
sage: om.lie_derivative(v).display()
\end{verbatim}
Parallel computation:

```python
sage: Parallelism().set('tensor', nproc=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:

```python
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 \( \Psi : U \to V \), 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 \( S \to M \) associated with the tensor field

**OUTPUT:**

- instance of *TensorFieldParal* representing the restriction

**EXAMPLES:**

Restriction of a vector field defined on \( \mathbb{R}^2 \) to a disk:

```python
sage: M = Manifold(2, 'R^2')
sage: _cart.<x,y> = M.chart()  # Cartesian coordinates on R^2
sage: v = M.vector_field(x+y, -1+x^2, name='v')
sage: D = M.open_subset('D')  # the unit open disc
sage: _cart_D = _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) ∂/∂x + (x^2 - 1) ∂/∂y
```

The symbolic expressions of the components with respect to Cartesian coordinates are equal:

```python
sage: bool( v_D[1].expr() == v[1].expr() )
True
```

but neither the chart functions representing the components (they are defined on different charts):
nor the scalar fields representing the components (they are defined on different open subsets):

```
False
```

The restriction of the vector field to its own domain is of course itself:

```
sage: v.restrict(M) is v
True
```

**series_expansion**(symbol, order)

Expand the tensor field in power series with respect to a small parameter.

If the small parameter is \( \epsilon \) and \( T \) is \( \text{self} \), the power series expansion to order \( n \) is

\[
T = T_0 + \epsilon T_1 + \epsilon^2 T_2 + \cdots + \epsilon^n T_n + O(\epsilon^{n+1}),
\]

where \( T_0, T_1, \ldots, T_n \) are \( n+1 \) tensor fields of the same tensor type as \( \text{self} \) and do not depend upon \( \epsilon \).

INPUT:

- `symbol` – symbolic variable (the “small parameter” \( \epsilon \)) with respect to which the components of \( \text{self} \) are expanded in power series
- `order` – integer; the order \( n \) of the expansion, defined as the degree of the polynomial representing the truncated power series in `symbol`

OUTPUT:

- list of the tensor fields \( T_i \) (size \( \text{order+1} \))

EXAMPLES:

```python
sage: M = Manifold(4, 'M', structure='Lorentzian')
sage: C.<t,x,y,z> = M.chart()
sage: e = var('e')
sage: g = M.metric()
sage: h1 = M.tensor_field(0,2,sym=(0,1))
sage: h2 = M.tensor_field(0,2,sym=(0,1))
sage: g[0, 0], g[ 1, 1], g[ 2, 2], g[ 3, 3] = -1, 1, 1, 1
sage: h1[0, 1], h1[ 1, 2], h1[ 2, 3] = 1, 1, 1
sage: h2[0, 2], h2[ 1, 3] = 1, 1
sage: g.set(g + e*h1 + e^2*h2)
sage: g_ser = g.series_expansion(e, 2); g_ser

[Field of symmetric bilinear forms on the 4-dimensional Lorentzian manifold M, 
Field of symmetric bilinear forms on the 4-dimensional Lorentzian manifold M, 
Field of symmetric bilinear forms on the 4-dimensional Lorentzian manifold M]
sage: g_ser[0][:]
[-1 0 0 0]
[ 0 1 0 0]
[ 0 0 1 0]
[ 0 0 0 1]
sage: g_ser[1][:]
[0 1 0 0]
[1 0 1 0]
[0 1 0 1]
```

(continues on next page)
[0 0 1 0]
\[\text{sage: } \text{g\_ser[2][:]}\]
[0 0 1 0]
[0 0 0 1]
[1 0 0 0]
[0 1 0 0]
\[\text{sage: all([g\_ser[1] == h1, g\_ser[2] == h2])}\]
True

\textbf{set_calc_order} (\textit{symbol, order, truncate=False})

Trigger a power series expansion with respect to a small parameter in computations involving the tensor field. This property is propagated by usual operations. The internal representation must be \texttt{SR} for this to take effect.

If the small parameter is $\epsilon$ and $T$ is \texttt{self}, the power series expansion to order $n$ is

$$T = T_0 + \epsilon T_1 + \epsilon^2 T_2 + \cdots + \epsilon^n T_n + O(\epsilon^{n+1}),$$

where $T_0, T_1, \ldots, T_n$ are $n+1$ tensor fields of the same tensor type as \texttt{self} and do not depend upon $\epsilon$.

\textbf{INPUT:}

- \texttt{symbol} – symbolic variable (the “small parameter” $\epsilon$) with respect to which the components of \texttt{self} are expanded in power series
- \texttt{order} – integer; the order $n$ of the expansion, defined as the degree of the polynomial representing the truncated power series in \texttt{symbol}
- \texttt{truncate} – (default: \texttt{False}) determines whether the components of \texttt{self} are replaced by their expansions to the given order

\textbf{EXAMPLES:}

\begin{verbatim}
\texttt{sage: M = Manifold(4, 'M', structure='Lorentzian')}
\texttt{sage: C.<t,x,y,z> = M.chart()}
\texttt{sage: e = var('e')}
\texttt{sage: g = M.metric()}
\texttt{sage: h1 = M.tensor_field(0, 2, sym=(0,1))}
\texttt{sage: h2 = M.tensor_field(0, 2, sym=(0,1))}
\texttt{sage: g[0, 0], g[ 1, 1], g[ 2, 2], g[ 3, 3] = -1, 1, 1, 1}
\texttt{sage: h1[0, 1], h1[ 1, 2], h1[ 2, 3] = 1, 1, 1}
\texttt{sage: h2[0, 2], h2[1, 3] = 1, 1}
\texttt{sage: g.set(g + e*h1 + e^2*h2)}
\texttt{sage: g.set_calc_order(e, 1)}
\texttt{sage: g[:]}  # [-1 e e^2 0]
\texttt{[ e 1 e e^2]}
\texttt{[e^2 e 1 e]}
\texttt{[ 0 e^2 e 1]}
\texttt{sage: g.set_calc_order(e, 1, truncate=\texttt{True})}
\texttt{sage: g[:]}  # [-1 e 0 0]
\texttt{[ e 1 e 0]}
\texttt{[ 0 e 1 e]}
\texttt{[ 0 0 e 1]}
\end{verbatim}

\textbf{set_comp} (\textit{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 \texttt{add_comp()} instead.

**INPUT:**

- \texttt{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 \texttt{Components}; if such components did not exist previously, they are created

**EXAMPLES:**

```sage
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, (∂/∂x,∂/∂y))
sage: t.set_comp(e_xy)[1,0] = 2
sage: t.display(e_xy)
t = 2 ∂/∂y⊗dx
```

Setting components in a new frame (\texttt{e}):

```sage
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 \texttt{e_xy} have been erased:

```sage
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, (∂/∂x,∂/∂y))
```

Setting components in a frame defined on a subdomain deletes previously defined components as well:

```sage
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
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))
```

**\texttt{truncate} (symbol, order)**

Return the tensor field truncated at a given order in the power series expansion with respect to some small parameter.
If the small parameter is $\epsilon$ and $T$ is self, the power series expansion to order $n$ is

$$T = T_0 + \epsilon T_1 + \epsilon^2 T_2 + \cdots + \epsilon^n T_n + O(\epsilon^{n+1}),$$

where $T_0, T_1, \ldots, T_n$ are $n + 1$ tensor fields of the same tensor type as self and do not depend upon $\epsilon$.

INPUT:

- symbol – symbolic variable (the “small parameter” $\epsilon$) with respect to which the components of self are expanded in power series
- order – integer; the order $n$ of the expansion, defined as the degree of the polynomial representing the truncated power series in symbol

OUTPUT:

- the tensor field $T_0 + \epsilon T_1 + \epsilon^2 T_2 + \cdots + \epsilon^n T_n$

EXAMPLES:

```python
sage: M = Manifold(4, 'M', structure='Lorentzian')
sage: C.<t,x,y,z> = M.chart()
sage: e = var('e')
sage: g = M.metric()
sage: h1 = M.tensor_field(0,2,sym=(0,1))
sage: h2 = M.tensor_field(0,2,sym=(0,1))
sage: g[0, 0], g[1, 1], g[2, 2], g[3, 3] = -1, 1, 1, 1
sage: h1[0, 1], h1[1, 2], h1[2, 3] = 1, 1, 1
sage: h2[0, 2], h2[1, 3] = 1, 1
sage: g.set(g + e*h1 + e^2*h2)
sage: g[:]
[-1  e  e^2  0]
[ e  1  e^2]
[e^2  e  1  e]
[ 0 e^2  e  1]
sage: g.truncate(e, 1)[:]
[-1  e  0  0]
[ e  1  e  0]
[ 0 e  1 e]
[ 0 0 e  1]
```

## 2.9 Differential Forms

### 2.9.1 Differential Form Modules

The set $\Omega^p(U, \Phi)$ of $p$-forms along a differentiable manifold $U$ with values on a differentiable manifold $M$ via a differentiable map $\Phi : U \to M$ (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 $\Omega^p(U, \Phi)$:

- `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 $M$ (the subclass `VectorFieldDualFreeModule` implements the special case of differential 1-forms on a parallelizable manifold $M$)

AUTHORS:

- Eric Gourgoulhon (2015): initial version
REFERENCES:

• [KN1963]
• [Lee2013]

\[
\text{class } \text{sage.manifolds.differentiable.diff_form_module.DiffFormFreeModule(vec-tor\_field\_module, degree)}
\]

Bases: ExtPowerDualFreeModule

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 \( M \) of dimension \( n \), the set \( \Omega^p(U, \Phi) \) of \( p \)-forms along \( U \) with values on \( M \) is a free module of rank \( \binom{n}{p} \) over \( C^k(U) \), the commutative algebra of differentiable scalar fields on \( U \) (see \text{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 \( \mathbb{R} \)).

Note: This class implements \( \Omega^p(U, \Phi) \) in the case where \( M \) is parallelizable; \( \Omega^p(U, \Phi) \) is then a free module. If \( M \) is not parallelizable, the class \text{DiffFormModule} must be used instead.

For the special case of 1-forms, use the class \text{VectorFieldDualFreeModule}.

INPUT:

• \text{vector\_field\_module} – free module \( \mathfrak{X}(U, \Phi) \) of vector fields along \( U \) associated with the map \( \Phi : U \rightarrow V \)

• \text{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 Omega^2(M) of 2-forms on the 3-dimensional differentiable manifold M
sage: latex(A)
\Omega^{2}
\left(M\right)
```

\( \Lambda \) is nothing but the second exterior power of the dual of \( XM \), i.e. we have \( \Omega^2(M) = \Lambda^2(\mathfrak{X}(M)^*) \) (see \text{ExtPowerDualFreeModule}):

```
sage: A is XM.dual_exterior_power(2)
True
```

\( \Omega^2(M) \) is a module over the algebra \( C^k(M) \) of (differentiable) scalar fields on \( M \):

2.9. Differential Forms 691
Elements can be constructed from $A$. In particular, $0$ yields the zero element of $A$:

```sage
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
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 afterward:

```sage
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 $\Omega^1(M)$ is nothing but the dual of $\mathfrak{X}(M)$ (the free module of vector fields on $M$):

```sage
sage: L1 = M.diff_form_module(1) ; L1
Free module Omega^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, it is also equal to the set $T^{(0,1)}(M)$ of such tensors to $\Omega^1(M)$:

```sage
sage: T01 = M.tensor_field_module((0,1)) ; T01
Free module Omega^1(M) of 1-forms on the 3-dimensional differentiable manifold M
sage: L1 is T01
True
```
For a degree $p \geq 2$, the coercion holds only in the direction $\Omega^p(M) \to T^{(0,p)}(M)$:

```python
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 $\Omega^2(M) \to T^{(0,2)}(M)$ in action:

```python
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:

```python
sage: U = M.open_subset('U', coord_def=(X: x^2+y^2<1))
sage: B = U.diff_form_module(2) ; B
Free module $\Omega^2(U)$ 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`

```python
class sage.manifolds.differentiable.diff_form_module.DiffFormModule(vector_field_module, degree)

Bases: UniqueRepresentation, 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 $\Phi : U \to M$ to a differentiable manifold $M$, the set $\Omega^p(U, \Phi)$ 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 $\mathbb{R}$).

**Note:** This class implements $\Omega^p(U, \Phi)$ in the case where $M$ is not assumed to be parallelizable; the module
Ω^p(U, Φ) is then not necessarily free. If M is parallelizable, the class `DiffFormFreeModule` must be used instead.

**INPUT:**

- `vector_field_module` – module X(U, Φ) of vector fields along U with values on M via the map Φ: U → M
- `degree` – positive integer; the degree p of the differential forms

**EXAMPLES:**

Module of 2-forms on a non-parallelizable 2-dimensional manifold:

```python
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 Omega^2(M) of 2-forms on the 2-dimensional differentiable manifold M
sage: latex(A)
\Omega^{2}\left(M\right)
A is nothing but the second exterior power of the dual of XM, i.e. we have Ω^2(M) = Λ^2(X(M)^*):

```python
sage: A.is XM.dual_exterior_power(2)
True
```

Modules of differential forms are unique:

```python
sage: A.is M.diff_form_module(2)
True
```

Ω^2(M) is a module over the algebra C^k(M) of (differentiable) scalar fields on M:

```python
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() is XM
True
```
Elements can be constructed from A(). In particular, 0 yields the zero element of A:

```sage
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
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 initializ. 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
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 $\Omega^1(M)$ is nothing but the dual of $\mathcal{X}(M)$ (the module of vector fields on $M$):

```sage
sage: L1 = M.diff_form_module(1) ; L1
Module Omega^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 $\Omega^1(M)$:

```sage
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
sage: T01.has_coerce_map_from(L1)
True
```

For a degree $p \geq 2$, the coercion holds only in the direction $\Omega^p(M) \rightarrow T^{(0,p)}(M)$:
A differentiable manifold $M$

```
sage: T02.has_coerce_map_from(A)
True
sage: A.has_coerce_map_from(T02)
False
```

The coercion map $T^{(0,1)}(M) \rightarrow \Omega^1(M)$ 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 $\Omega^1(M) \rightarrow T^{(0,1)}(M)$ 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
```

The coercion map $\Omega^2(M) \rightarrow T^{(0,2)}(M)$ in action:

```
sage: ta = T02(a) ; ta
Tensor field $a$ of type $(0,2)$ on the 2-dimensional differentiable manifold $M$
sage: a.display(eU)
a = 3*x \, dx\wedge dy - 3*x \, dy\wedge dx
sage: a.display(eV)
a = 3*x \, dx\wedge dy
sage: ta.display(eU)
a = (-3/4*u - 3/4*v) \, du\wedge dv + (3/4*u + 3/4*v) \, dv\wedge du
sage: ta.display(eV)
a = (-3/4*u - 3/4*v) \, du\wedge 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 $\Omega^2(U)$ of 2-forms on the Open subset $U$ of the 2-dimensional differentiable manifold $M$
sage: L2U.has_coerce_map_from(A)
(continues on next page)
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 self is constructed.

OUTPUT:

• a VectorFieldModule representing the module on which self is defined

EXAMPLES:

sage: M = Manifold(3, 'M')
sage: A2 = M.diff_form_module(2) ; A2
Module Omega^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 Omega^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 self.

OUTPUT:

• integer p such that self 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

tensor(*others)

Return the tensor product of self and others.

EXAMPLES:
```python
sage: M = FiniteRankFreeModule(QQ, 2)
sage: M.tensor_product(M)
Free module of type-(2,0) tensors on the 2-dimensional vector space over the Rational Field
sage: M.tensor_product(M.dual())
Free module of type-(1,1) tensors on the 2-dimensional vector space over the Rational Field
sage: M.dual().tensor_product(M, M.tensor_module(1,2))
Free module of type-(1,2) tensors on the 2-dimensional vector space over the Rational Field
sage: M.tensor_product(M.tensor_module(1,2))
Free module of type-(2,2) tensors on the 2-dimensional vector space over the Rational Field
sage: M.tensor_module(1,2).tensor_product(M)
Free module of type-(2,2) tensors on the 2-dimensional vector space over the Rational Field
sage: M.tensor_module(1,1).tensor_product(M.tensor_module(1,2))
Free module of type-(2,3) tensors on the 2-dimensional vector space over the Rational Field
sage: Sym2M = M.tensor_module(2, 0, sym=range(2)); Sym2M
Free module of fully symmetric type-(2,0) tensors on the 2-dimensional vector space over the Rational Field
sage: Sym01x23M = Sym2M.tensor_product(Sym2M); Sym01x23M
Free module of type-(4,0) tensors on the 2-dimensional vector space over the Rational Field,
with symmetry on the index positions (0, 1), with symmetry on the index positions (2, 3)
sage: Sym01x23M._index_maps
((0, 1), (2, 3))
sage: N = M.tensor_module(3, 3, sym=[1, 2], antisym=[3, 4]); N
Free module of type-(3,3) tensors on the 2-dimensional vector space over the Rational Field,
with symmetry on the index positions (1, 2), with antisymmetry on the index positions (3, 4)
sage: NxN = N.tensor_product(N); NxN
Free module of type-(6,6) tensors on the 2-dimensional vector space over the Rational Field,
with symmetry on the index positions (1, 2), with symmetry on the index positions (4, 5),
with antisymmetry on the index positions (6, 7), with antisymmetry on the index positions (9, 10)
sage: NxN._index_maps
((0, 1, 2, 6, 7, 8), (3, 4, 5, 9, 10, 11))
tensor_product(*others)
Return the tensor product of self and others.
EXAMPLES:
```
Manifolds, Release 10.3

2.9. Differential Forms

\texttt{tensor_type}()

Return the tensor type of \texttt{self} if \texttt{self} is a module of 1-forms.

In this case, the pair (0, 1) is returned, indicating that the module is identified with the dual of the base module.

For differential forms of other degrees, an exception is raised.

EXAMPLES:

\texttt{sage: M = Manifold(3, 'M')}
\texttt{sage: M.diff_form_module(1).tensor_type()}
\texttt{(0, 1)}
\texttt{Traceback (most recent call last):}
\texttt{...}
\texttt{NotImplementedError}
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

class sage.manifolds.differentiable.diff_form_module.VectorFieldDualFreeModule(vector_field_module)

Bases: DiffFormFreeModule

Free module of differential 1-forms along a differentiable manifold $U$ with values on a parallelizable manifold $M$.

Given a differentiable manifold $U$ and a differentiable map $\Phi : U \to M$ to a parallelizable manifold $M$ of dimension $n$, the set $\Omega^1(U, \Phi)$ of 1-forms along $U$ with values on $M$ is a free module of rank $n$ over $C^k(U)$, the commutative algebra of differentiable scalar fields on $U$ (see DiffScalarFieldAlgebra). The standard case of 1-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 $\mathbb{R}$).

Note: This class implements $\Omega^1(U, \Phi)$ in the case where $M$ is parallelizable; $\Omega^1(U, \Phi)$ is then a free module. If $M$ is not parallelizable, the class DiffFormModule must be used instead.

INPUT:

    • vector_field_module – free module $X(U, \Phi)$ of vector fields along $U$ associated with the map $\Phi : U \to V$

EXAMPLES:

Free module of 1-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(1) ; A
    Free module $\Omega^1(M)$ of 1-forms on the 3-dimensional differentiable manifold $M$
    sage: latex(A)
    \Omega^{1}\left(M\right)
    A is nothing but the dual of $XM$ (the free module of vector fields on $M$) and thus also equal to the 1st exterior power of the dual, i.e. we have $\Omega^1(M) = \Lambda^1(X(M)^*) = X(M)^*$ (See ExtPowerDualFreeModule):

    sage: A.is XM.dual_exterior_power(1)
    True

$\Omega^1(M)$ 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
    (continues on next page)
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)
1-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 = [3*x,-z,4]
sage: a = A(comp, frame=X.frame(), name='a') ; a
1-form a on the 3-dimensional differentiable manifold M
sage: a.display()
a = 3*x dx - z dy + 4 dz
```

An alternative is to construct the 1-form from an empty list of components and to set the nonzero nonredundant components afterwards:
```
sage: a = A([], name='a')
sage: a[0] = 3*x # component in the manifold’s default frame
sage: a[1] = -z
sage: a[2] = 4
sage: a.display()
a = 3*x dx - z dy + 4 dz
```

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 $\Omega^1(M)$:
```
sage: T01 = M.tensor_field_module((0,1)) ; T01
Free module Omega^1(M) of 1-forms on the 3-dimensional differentiable manifold M
sage: A.has_coerce_map_from(T01)
True
```

There is also a coercion map in the reverse direction:
```
sage: T01.has_coerce_map_from(A)
True
```

The coercion map $T^{(0,1)}(M) \rightarrow \Omega^1(M)$ in action:
sage: b = T01([-x,2,3*y], name = 'b'); b
1-form b on the 3-dimensional differentiable manifold M
sage: b.display()
b = -x dx + 2 dy + 3*y dz
sage: lb = A(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 $\Omega^1(M) \to T^{(0,1)}(M)$ in action:

sage: tlb = T01(lb); tlb
1-form b on the 3-dimensional differentiable manifold M
sage: tlb == b
True

tensor_type()

Return the tensor type of self.

EXAMPLES:

sage: M = Manifold(3, 'M')
sage: X.<x,y,z> = M.chart()
sage: A = M.vector_field_module().dual(); A
Free module $\Omega^1(M)$ of 1-forms on the 3-dimensional differentiable manifold M
sage: A.tensor_type()
(0, 1)

2.9.2 Differential Forms

Let $U$ and $M$ be two differentiable manifolds. Given a positive integer $p$ and a differentiable map $\Phi : U \to M$, 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 $\mathbb{R}$).

Two classes implement differential forms, depending whether the manifold $M$ is parallelizable:

- `DiffFormParal` when $M$ is parallelizable
- `DiffForm` when $M$ is not assumed parallelizable.

AUTHORS:

- Joris Vankerschaver (2010): developed a previous class, `DifferentialForm` (cf. github issue #24444), 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: 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

\[
\forall x \in U, \quad a(x) \in \Lambda^p(T^*_{\Phi(x)}M),
\]

where \( T^*_{\Phi(x)}M \) is the dual of the tangent space to \( M \) at \( \Phi(x) \) and \( \Lambda^p \) stands for the exterior power of degree \( p \) (cf. \texttt{ExtPowerDualFreeModule}). In other words, \( a(x) \) is an alternating multilinear form of degree \( p \) of the tangent vector space \( T_{\Phi(x)}M \).

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 \( \mathbb{R} \)).

**Note:** If \( M \) is parallelizable, the class \texttt{DiffFormParal} must be used instead.

**INPUT:**

- \texttt{vector_field_module} – module \( \mathfrak{X}(U, \Phi) \) of vector fields along \( U \) with values on \( M \) via the map \( \Phi \)
- \texttt{degree} – the degree of the differential form (i.e. its tensor rank)
- \texttt{name} – (default: None) name given to the differential form
- \texttt{latex_name} – (default: None) \LaTeX\ symbol to denote the differential form; if none is provided, the \LaTeX\ symbol is set to \texttt{name}

**EXAMPLES:**

Differential form of degree 2 on a non-parallelizable 2-dimensional manifold:

```python
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()  # \( \Phi \)
...: 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 = c_xy.transition_map(c_uv, (x+y, x-y), intersection_name = 'W',
...: restrictions1= x>0, restrictions2= u+v>0)
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 \( \Omega^2(M) \) of 2-forms on the 2-dimensional differentiable manifold M
sage: a.degree()
2
```

Setting the components of \( a \):
A 1-form on $M$:

```python
sage: a = M.one_form(name='a') ; a
1-form a on the 2-dimensional differentiable manifold M
sage: a.parent()
Module Omega^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:

```python
sage: a[eU,0,1] = [-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
```

It is also possible to set the components at the 1-form definition, via a dictionary whose keys are the vector frames:

```python
sage: a1 = M.one_form({eU: [-y, x], eV: [v/2, -u/2]}, name='a')  
sage: a1 == a
True
```

The exterior derivative of the 1-form is a 2-form:

```python
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
```

The exterior derivative can also be obtained by applying the function `diff` to a differentiable form:

```python
sage: diff(a) is a.exterior_derivative()
True
```

Another 1-form defined by its components in $eU$:

```python
sage: b = M.one_form(1+x*y, x^2, frame=eU, name='b')
```

Since $eU$ is the default vector frame on $M$, it can be omitted in the definition:

```python
sage: b = M.one_form(1+x*y, x^2, name='b')
sage: b.add_comp_by_continuation(eV, W, c_uv)
```

Adding two 1-forms results in another 1-form:
The exterior product of two 1-forms is a 2-form:

\[
\mathbf{a} \wedge \mathbf{b} = (-2x^2y - x) \, dx \wedge dy
\]

\[
\mathbf{a} \wedge \mathbf{b} = (1/8u^3 - 1/8u^2v^2 - 1/8v^3 + 1/8(u^2 + 2)v + 1/4u) \, du \wedge dv
\]

Examples with SymPy as the symbolic engine

From now on, we ask that all symbolic calculus on manifold \( M \) are performed by SymPy:
Another 1-form:

```
sage: b = M.one_form(1+x*y, x^2, name='b')
sage: b.add_comp_by_continuation(eV, W, c_uv)
```

Adding two 1-forms:

```
sage: s = a + b
sage: s.display(eU)
a+b = (x*y - y + 1) dx + x*(x + 1) dy
sage: s.display(eV)
a+b = (u**2/4 + u*v/4 + v/2 + 1/2) du + (-u*v/4 - u/2 - v**2/4 + 1/2) dv
```

The exterior product of two 1-forms:

```
sage: s = a.wedge(b)
sage: s.display(eU)
a∧b = x*(-2*x*y - 1) dx∧dy
sage: s.display(eV)
a∧b = (u**3/8 + u**2*v/8 - u*v**2/8 + u/4 - v**3/8 + v/4) du∧dv
```

Multiplying a 1-form by a scalar field:

```
sage: f = M.scalar_field({c_xy: (x+y)^2, c_uv: u^2}, name='f')
sage: s = f*a
sage: s.display(eU)
f*a = y*(-x**2 - 2*x*y - y**2) dx + x*(x**2 + 2*x*y + y**2) dy
sage: s.display(eV)
f*a = u**2*v/2 du - u**3/2 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
```

**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:

```python
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)
```

The 1-form:

```python
sage: a = M.one_form({e_xy: [-y^2, x^2]}, name='a')
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
```

Its exterior derivative:

```python
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
```

The result is cached, i.e. is not recomputed unless a is changed:

```python
sage: a.exterior_derivative() is da
True
```

Instead of invoking the method `exterior_derivative()`, one may use the global function `diff`:

```python
sage: diff(a) is a.exterior_derivative()
True
```

Let us check Cartan’s identity:

```python
sage: v = M.vector_field({e_xy: [-y, x]}, name='v')
sage: v.add_comp_by_continuation(e_uv, U.intersection(V), c_uv)
sage: a.lie_der(v) == v.contract(diff(a)) + diff(a(v))  # long time
True
```

**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:

```plaintext
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:

```plaintext
sage: a = M.one_form({e_xy: [-y^2, x^2]}, name='a')
```

Its exterior derivative:

```plaintext
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\wedge 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\wedge dv
```

The result is cached, i.e. is not recomputed unless `a` is changed:

```plaintext
sage: a.exterior_derivative() is da
True
```

Instead of invoking the method `exterior_derivative()`, one may use the global function `diff`:

```plaintext
sage: diff(a) is a.exterior_derivative()
True
```

Let us check Cartan’s identity:

```plaintext
sage: v = M.vector_field({e_xy: [-y, x]}, name='v')
```

```plaintext
hodge_dual (nongenerate_tensor=None, minus_eigenvalues_convention=False)
```
Compute the Hodge dual of the differential form with respect to some non-degenerate bilinear form (Riemannian metric or symplectic form).
If the differential form is a $p$-form $A$, its Hodge dual with respect to the non-degenerate form $g$ is the $(n-p)$-form $*A$ defined by

$$ *A_{i_1 \ldots i_{n-p}} = \frac{1}{p!} A^{k_1 \ldots k_p} \epsilon_{k_1 \ldots k_p i_1 \ldots i_{n-p}} $$

where $n$ is the manifold’s dimension, $\epsilon$ is the volume $n$-form associated with $g$ (see `volume_form()`) and the indices $k_1, \ldots, k_p$ are raised with $g$. If $g$ is a pseudo-Riemannian metric, sometimes an additional multiplicative factor of $(-1)^s$ is introduced on the right-hand side, where $s$ is the number of negative eigenvalues of $g$. This convention can be enforced by setting the option `minus_eigenvalues_convention`.

**INPUT:**

- `nondegenerate_tensor`: a non-degenerate bilinear form defined on the same manifold as the current differential form; must be an instance of `PseudoRiemannianMetric` or `SymplecticForm`. If none is provided, the ambient domain of `self` is supposed to be endowed with a default metric and this metric is then used.

- `minus_eigenvalues_convention` – if `true`, a factor of $(-1)^s$ is introduced with $s$ being the number of negative eigenvalues of the `nondegenerate_tensor`.

**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 $S^2$ 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
c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart() # stereographic coord. \(\to\) (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
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
```

We endow $S^2$ with the orientation defined by the stereographic frame from the North pole, i.e. $eU; eV$ is then left-handed and in order to define an orientation on the whole manifold, we introduce a vector frame on $V$ by swapping $eV$'s vectors:

```sage
f = V.vector_frame('f', (eV[2], eV[1]))
sage: M.set_orientation([eU, f])
```

Then we construct the 1-form and take its Hodge dual w.r.t. $g$:

```sage
a = M.one_form({eU: [-y, x]}, name='a')
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
```

(continues on next page)
Instead of calling the method \emph{hodge_dual()} on the differential form, one can invoke the method \emph{hodge_star()} of the metric:

\begin{verbatim}
sage: a.hodge_dual(g) == g.hodge_star(a)
True
\end{verbatim}

For a 1-form and a Riemannian metric in dimension 2, the Hodge dual applied twice is minus the identity:

\begin{verbatim}
sage: ssa = sa.hodge_dual(g); ssa
1-form \(\ast a\) on the 2-dimensional differentiable manifold \(S^2\)

sage: ssa == -a
True
\end{verbatim}

The Hodge dual of the metric volume 2-form is the constant scalar field 1 (considered as a 0-form):

\begin{verbatim}
sage: eps = g.volume_form(); eps
2-form \(\varepsilon_g\) on the 2-dimensional differentiable manifold \(S^2\)

sage: eps.display(eU)
\(\varepsilon_g = 4/(x^4 + y^4 + 2*(x^2 + 1)*y^2 + 2*x^2 + 1)\, dx\wedge dy\)

sage: eps.display(eV)
\(\varepsilon_g = -4/(u^4 + v^4 + 2*(u^2 + 1)*v^2 + 2*u^2 + 1)\, du\wedge dv\)

sage: seps = eps.hodge_dual(g); seps
Scalar field \(\varepsilon_g\) on the 2-dimensional differentiable manifold \(S^2\)

sage: seps.display()
\(\varepsilon_g: S^2 \to \mathbb{R}\)

on \(U\): \((x, y) \mapsto 1\)

on \(V\): \((u, v) \mapsto 1\)
\end{verbatim}

Hodge dual of a 1-form in the Euclidean space \(\mathbb{R}^3\):

\begin{verbatim}
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: var('Ax Ay Az')
(Ax, Ay, Az)
sage: a = M.one_form(Ax, Ay, Az, name='A')
sage: sa = a.hodge_dual(g) ; sa
2-form \(A\) on the 3-dimensional differentiable manifold \(M\)

sage: sa.display()
\(A = Az\, dx\wedge dy - Ay\, dx\wedge dz + Ax\, dy\wedge dz\)

sage: ssa = sa.hodge_dual(g) ; ssa
1-form \(\ast A\) on the 3-dimensional differentiable manifold \(M\)

sage: ssa.display()
\(\ast A = Ax\, dx + Ay\, dy + Az\, dz\)

sage: ssa == a # must hold for a Riemannian metric in dimension 3
True
\end{verbatim}

See the documentation of \emph{hodge_star()} for more examples.
interior_product(qvect)

Interior product with a multivector field.

If self is a differential form $A$ of degree $p$ and $B$ is a multivector field of degree $q \geq p$ on the same manifold, the interior product of $A$ by $B$ is the multivector field $\iota_A B$ of degree $q - p$ defined by

$$(\iota_A B)^{i_1 \ldots i_q}_{p} = A^{k_1 \ldots k_p} B^{k_1 \ldots k_p i_1 \ldots i_q}_{p}$$

Note: $A$.interior_product($B$) yields the same result as $A$.contract(0, ..., p-1, $B$, 0, ..., p-1) (cf. contract()), but interior_product is more efficient, the alternating character of $A$ being not used to reduce the computation in contract().

INPUT:

- `qvect` – multivector field $B$ (instance of MultivectorField); the degree of $B$ must be at least equal to the degree of `self`

OUTPUT:

- scalar field (case $p = q$) or `MultivectorField` (case $p < q$) representing the interior product $\iota_A B$, where $A$ is `self`

See also:

`interior_product()` for the interior product of a multivector field with a differential form

EXAMPLES:

Interior product of a 1-form with a 2-vector field on the 2-sphere:

\begin{verbatim}
sage: M = Manifold(2, 'S^2', start_index=1) # the 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() # stereographic coord. North sage: c_uv.<u,v> = V.chart() # stereographic coord. 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.one_form({e_xy: [y, x]}, name='a') sage: a.add_comp_by_continuation(e_uv, W, c_uv) sage: b = M.multivector_field(2, name='b') sage: b[e_xy,1,2] = x*y sage: b.add_comp_by_continuation(e_uv, W, c_uv) sage: s = a.interior_product(b); s Vector field i_a b on the 2-dimensional differentiable manifold S^2 sage: s.display(e_xy) i_a b = -x^2*y ∂/∂x + x*y^2 ∂/∂y sage: s.display(e_uv) i_a b = (u^4*v - 3*u^2*v^3)/(u^6 + 3*u^4*v^2 + 3*u^2*v^4 + v^6) ∂/∂u + (3*u^3*v^2 - u*v^4)/(u^6 + 3*u^4*v^2 + 3*u^2*v^4 + v^6) ∂/∂v sage: s == a.contract(b) True
\end{verbatim}

Interior product of a 2-form with a 2-vector field:
sage: a = M.diff_form(2, name='a')
sage: a[e_xy,1,2] = 4/(x^2+y^2+1)^2  # the standard area 2-form
sage: a.add_comp_by_continuation(e_uv, W, c_uv)
sage: s = a.interior_product(b); s
Scalar field i_a b on the 2-dimensional differentiable manifold S^2
sage: s.display()
i_a b: S^2 → ℝ
on U: (x, y) ↦ 8*x*y/(x^4 + y^4 + 2*(x^2 + 1)*y^2 + 2*x^2 + 1)
on V: (u, v) ↦ 8*u*v/(u^4 + v^4 + 2*(u^2 + 1)*v^2 + 2*u^2 + 1)

Some checks:

sage: s == a.contract(0, 1, b, 0, 1)
True
sage: s.restrict(U) == 2 * a[e_xy,1,2] * b[e_xy,1,2]
True
sage: s.restrict(V) == 2 * a[e_uv,1,2] * b[e_uv,1,2]
True

wedge (other)
Exterior product with another differential form.

INPUT:

• other — another differential form (on the same manifold)

OUTPUT:

• instance of DiffForm representing 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.one_form({e_xy: [y, x]}, name='a')
sage: a.add_comp_by_continuation(e_uv, W, c_uv)
sage: b = M.one_form({e_xy: [x ^2 + y^2, y]}, name='b')
... 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

If one of the two operands is unnamed, the result is unnamed too:
Manifolds, Release 10.3

```python
sage: b1 = M.diff_form(1)  # no name set
sage: b1[e_xy,:] = x^2 + y^2, y
sage: b1.add_comp_by_continuation(e_uv, W, c_uv)
sage: c1 = a.wedge(b1); c1
2-form on the 2-dimensional differentiable manifold S^2
sage: c1.display(e_xy)
(-x^3 - (x - 1)*y^2) dx∧dy
```

To give a name to the result, one shall use the method \texttt{set_name()}:  
```
sage: c1.set_name('c'); c1
2-form c on the 2-dimensional differentiable manifold S^2
sage: c1.display(e_xy)
c = (-x^3 - (x - 1)*y^2) dx∧dy
```

Wedging with scalar fields yields the multiplication from right:  
```
sage: f = M.scalar_field(x, name = f)
sage: f.add_expr_by_continuation(c_uv, W)
sage: t = a.wedge(f)
sage: t.display()
f*a = x*y dx + x^2 dy
```

```python

class \texttt{sage.manifolds.differentiable.diff_form.DiffFormParal}(vector_field_module:\ FreeModuleAltForm, TensorFieldParal, DiffForm)

Bases: \texttt{FreeModuleAltForm}, \texttt{TensorFieldParal}, \texttt{DiffForm}

Differential form with values on a parallelizable manifold.

Given a differentiable manifold \(U\), a differentiable map \(\Phi : U \to M\) to a parallelizable manifold \(M\) and a positive integer \(p\), a \textit{differential form of degree} \(p\) (or \(p\)-form) along \(U\) with values on \(M \supset \Phi(U)\) is a differentiable map

\[
a : U \to T^{(0,p)}M
\]

\((T^{(0,p)}M)\) being the tensor bundle of type \((0, p)\) over \(M\) such that

\[
\forall x \in U, \quad a(x) \in \Lambda^p(T^*_\Phi(x)M),
\]

where \(T^*_\Phi(x)M\) is the dual of the tangent space to \(M\) at \(\Phi(x)\) and \(\Lambda^p\) stands for the exterior power of degree \(p\) (cf. \texttt{ExtPowerDualFreeModule}). In other words, \(a(x)\) is an alternating multilinear form of degree \(p\) of the tangent vector space \(T^*_\Phi(x)M\).

The standard case of a differential form \textit{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 \(\mathbb{R}\)).

\textbf{Note:} If \(M\) is not parallelizable, the class \texttt{DiffForm} must be used instead.

INPUT:

- \texttt{vector_field_module} – free module \(\mathcal{X}(U, \Phi)\) of vector fields along \(U\) with values on \(M\) via the map \(\Phi\)
- \texttt{degree} – the degree of the differential form (i.e. its tensor rank)
```

2.9. Differential Forms 713
• 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, name='a') ; a
2-form a on the 4-dimensional differentiable manifold M
sage: a.parent()
Free module Omega^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, (∂/∂t,∂/∂x,∂/∂y,∂/∂z))
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[1,2] = 0 # OK, albeit useless
sage: a[2,1] = 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
```

(continues on next page)
A 2-form on the 4-dimensional differentiable manifold $M$:

```sage
s = a - b ; s
```

An example of a 3-form is the volume element on $\mathbb{R}^3$ in Cartesian coordinates:

```sage
M = Manifold(3, 'R3', latex_name=r'\mathbb{R}^3', start_index=1)
c_cart.<x,y,z> = M.chart()
eps = M.diff_form(3, name =epsilon, latex_name =r'\epsilon')
eps[1,2,3] = 1  # the only independent component
eps[:]
```

Spherical components of the volume element from the tensorial change-of-frame formula:

```sage
c_spher.<r,th,ph> = M.chart(rr:[0,+oo) th:[0,pi): \theta ph:[0,2*pi): \phi)
spher_to_cart = c_spher.transition_map(c_cart, ....:
[ r*sin(th)*cos(ph), r*sin(th)*sin(ph), r*cos(th)])
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))
```

As a shortcut of the above command, one can pass just the chart $c_spher$ to `display`, the vector frame being then assumed to be the coordinate frame associated with the chart:
The exterior product of two differential forms is performed via the method \texttt{wedge()}: 

\begin{verbatim}
sage: a = M.one_form(x*y*z, -z*x, y*z, name='A')
sage: b = M.one_form(cos(z), sin(x), cos(y), name='B')
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
\end{verbatim}

Let us check the formula relating the exterior product to the tensor product for 1-forms:

\begin{verbatim}
sage: a.wedge(b) == a*b - b*a
True
\end{verbatim}

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:

\begin{verbatim}
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 2 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 2 arguments:
no symmetry; antisymmetry: (0, 1)
\end{verbatim}

The exterior derivative of a differential form is obtained by means of the method \texttt{exterior_derivative()}: 

\begin{verbatim}
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
sage: dab.display()
dA∧B = -(x + 1)*z dx∧dy + (x + z) dy∧dz + x*y dx∧dz - x*y*z*cos(z) dy∧dz + (x*y*z*sin(x) + x*z*cos(z)) dy∧dx

or by applying the function \texttt{diff} to the differential form:

\begin{verbatim}
sage: diff(a) is a.exterior_derivative()
True
\end{verbatim}

As a 3-form over a 3-dimensional manifold, \(d(A∧B)\) is necessarily proportional to the volume 3-form:

\begin{verbatim}
sage: dab == dab[[1,2,3]]/eps[[1,2,3]]*eps
True
\end{verbatim}
We may also check that the classical anti-derivation formula is fulfilled:

```python
sage: dab == da.wedge(b) - a.wedge(db)
True
```

The Lie derivative of a 2-form is a 2-form:

```python
sage: v = M.vector_field(y*z, -x*z, x*y, name='v')
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:

```python
sage: ab.lie_der(v) == (v.contract(ab.exterior_derivative()) + v.contract(ab).exterior_derivative())
True
```

A 1-form on a $\mathbb{R}^3$:

```python
sage: om = M.one_form(name='omega', latex_name=r'\omega'); om
1-form omega on the 3-dimensional differentiable manifold R3
```

A 1-form is of course a differential form:

```python
sage: isinstance(om, sage.manifolds.differentiable.diff_form.DiffFormParal)
True
sage: om.parent()
Free module Omega^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:

```python
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:

```python
sage: v = M.vector_field(x, 2*y, 3*z, name='V')
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(V)
```

The tensor product of two 1-forms is a tensor field of type (0, 2):

```python
sage: a = M.one_form(1, 2, 3, name='A')
sage: b = M.one_form(6, 5, 4, name='B')
sage: c = a*b ; c
```

(continues on next page)
Tensor field $A \otimes B$ of type $(0,2)$ on the 3-dimensional differentiable manifold $\mathbb{R}^3$

```
sage: c[:]
[ 6  5  4]
[12 10  8]
[18 15 12]
sage: c.symmetries()  # c has no symmetries:
no symmetry; no antisymmetry
```

def 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(t*x*y*z, z*y**2, x*z**2, x**2 + y**2, name='A')
sage: da = a.exterior_derivative() ; da
2-form $dA$ on the 4-dimensional differentiable manifold $M$
sage: da.display()
\mathrm{d}A = -t*y*z \ dt \wedge dx - t*x*z \ dt \wedge dy - t*x*y \ dt \wedge dz
+ (-2*y*z + z^2) \ dx \wedge dy + (-y^2 + 2*x) \ dx \wedge dz
+ (-2*x*z + 2*y) \ dy \wedge 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 `diff`:

```
sage: diff(a) is a.exterior_derivative()
True
```

The exterior derivative is nilpotent:

```
sage: dda = da.exterior_derivative() ; dda
3-form $d^2A$ on the 4-dimensional differentiable manifold $M$
sage: dda.display()
d^2A = 0
sage: dda == 0
True
```

Let us check Cartan's identity:

```
sage: v = M.vector_field(-y, x, t, z, name='v')
sage: a.lie_der(v) == v.contract(diff(a)) + diff(a(v))  # long time
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:

```python
sage: M = Manifold(4, 'M')
sage: c_txyz.<t,x,y,z> = M.chart()
sage: a = M.one_form(t*x*y*z, z*y**2, x*z**2, x**2 + y**2, name='A')
sage: da = a.exterior_derivative() ; da
2-form dA on the 4-dimensional differentiable manifold M
sage: da.display()
\mathrm{d}A = -t*y*z \, dt \wedge dx - t*x*z \, dt \wedge dy - t*x*y \, dt \wedge dz
+ (-2*y*z + z^2) \, dx \wedge dy + (-y^2 + 2*x) \, dx \wedge dz
+ (-2*x*z + 2*y) \, dy \wedge dz
sage: latex(da)
\mathrm{d}A
```

The result is cached, i.e. is not recomputed unless `a` is changed:

```python
sage: a.exterior_derivative() is da
True
```

Instead of invoking the method `exterior_derivative()`, one may use the global function `diff`:

```python
sage: diff(a) is a.exterior_derivative()
True
```

The exterior derivative is nilpotent:

```python
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:

```python
sage: v = M.vector_field(-y, x, t, z, name='v')
sage: a.lie_der(v) == v.contract(diff(a)) + diff(a(v)) # long time
True
```

**interior_product(qvect)**

Interior product with a multivector field.

If `self` is a differential form `A` of degree `p` and `B` is a multivector field of degree `q \geq p` on the same manifold, the interior product of `A` by `B` is the multivector field \( \iota_A B \) of degree `q - p` defined by

\[
(\iota_A B)^{i_1 \ldots i_{q-p}} = A_{k_1 \ldots k_p} B^{k_1 \ldots k_p i_1 \ldots i_{q-p}}
\]

**Note:** `A.interior_product(B)` yields the same result as `A.contract(0, \ldots, p-1, B, 0, \ldots, p-1)` (cf. `contract()`), but `interior_product` is more efficient, the alternating character
of $A$ being not used to reduce the computation in contract()

INPUT:

- qvect – multivector field $B$ (instance of MultivectorFieldParal); the degree of $B$ must be at least equal to the degree of self

OUTPUT:

- scalar field (case $p = q$) or MultivectorFieldParal (case $p < q$) representing the interior product $\iota_A B$, where $A$ is self

See also:

interior_product() for the interior product of a multivector field with a differential form

EXAMPLES:

Interior product of a 1-form with a 2-vector field 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(2, 1+x, y*z, name =a)
sage: b = M.multivector_field(2, name =b)
sage: b[1,2], b[1,3], b[2,3] = y^2, z+x, -z^2
sage: s = a.interior_product(b); s
Vector field i_a b on the 3-dimensional differentiable manifold M
sage: s.display()
i_a b = -(x + 1)*y^2 - x*y*z - y*z^2) ∂/∂x
+ (y*z^3 + 2*y^2) ∂/∂y
+ (-(x + 1)*z^2 + 2*x + 2*z) ∂/∂z
sage: s == a.contract(b)
True
```

Interior product of a 2-form with a 2-vector field:

```
sage: a = M.diff_form(2, name =a)
sage: a[1,2], a[1,3], a[2,3] = x*y, -3, z
sage: s = a.interior_product(b); s
Scalar field i_a b on the 3-dimensional differentiable manifold M
sage: s.display()
i_a b: M → ℝ
(x, y, z) ↦ 2*x*y^3 - 2*z^3 - 6*x - 6*z
sage: s == a.contract(b)
True
```

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:
Manifolds, Release 10.3

```python
sage: M = Manifold(3, 'M', start_index=1)
sage: X.<x,y,z> = M.chart()
sage: a = M.one_form(2, 1+x, y*z, name='a')
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
```

Check:

```python
sage: s[1,2,3] == a[1]*b[2,3] + a[2]*b[3,1] + a[3]*b[1,2]
True
```

Wedging with scalar fields yields the multiplication from right:

```python
sage: f = M.scalar_field(x, name='f')
sage: t = a.wedge(f)
sage: t.display()
f*a = 2*x dx + (x^2 + x) dy + x*y*z dz
```

### 2.10 Mixed Differential Forms

#### 2.10.1 Graded Algebra of Mixed Differential Forms

Let $M$ and $N$ be differentiable manifolds and $\varphi : M \to N$ a differentiable map. The space of *mixed differential forms along* $\varphi$, denoted by $\Omega^*(M, \varphi)$, is given by the direct sum $\bigoplus_{j=0}^n \Omega^j(M, \varphi)$ of differential form modules, where $n = \dim(N)$. With the wedge product, $\Omega^*(M, \varphi)$ inherits the structure of a graded algebra. See MixedFormAlgebra for details.

This algebra is endowed with a natural chain complex structure induced by the exterior derivative. The corresponding homology is called *de Rham cohomology*. See DeRhamCohomologyRing for details.

AUTHORS:
- Michael Jung (2019): initial version

class sage.manifolds.differentiable.mixed_form_algebra.MixedFormAlgebra(vector_field_module)

Bases: Parent, UniqueRepresentation

An instance of this class represents the graded algebra of mixed forms. That is, if $\varphi : M \to N$ is a differentiable map between two differentiable manifolds $M$ and $N$, the *graded algebra of mixed forms* $\Omega^*(M, \varphi)$ along $\varphi$ is defined via the direct sum $\bigoplus_{j=0}^n \Omega^j(M, \varphi)$ consisting of differential form modules (cf. DiffFormModule), where $n$ is the dimension of $N$. Hence, $\Omega^*(M, \varphi)$ is a module over $C^k(M)$ and a vector space over $\mathbb{R}$ or $\mathbb{C}$. Furthermore notice, that

$$\Omega^*(M, \varphi) \cong C^k \left( \bigoplus_{j=0}^n \Lambda^j(\varphi^* T^* N) \right),$$
where $C^k$ denotes the global section functor for differentiable sections of order $k$ here. The wedge product induces a multiplication on $\Omega^*(M, \varphi)$ and gives it the structure of a graded algebra since
\[
\Omega^k(M, \varphi) \wedge \Omega^l(M, \varphi) \subset \Omega^{k+l}(M, \varphi).
\]
Moreover, $\Omega^*(M, \varphi)$ inherits the structure of a chain complex, called de Rham complex, with the exterior derivative as boundary map, that is
\[
0 \to \Omega^0(M, \varphi) \xrightarrow{d_0} \Omega^1(M, \varphi) \xrightarrow{d_1} \ldots \xrightarrow{d_{n-1}} \Omega^n(M, \varphi) \xrightarrow{d_n} 0.
\]
The induced cohomology is called de Rham cohomology, see `cohomology()` or `DeRhamCohomologyRing` respectively.

**INPUT:**
- `vector_field_module` -- module $\mathfrak{X}(M, \varphi)$ of vector fields along $M$ associated with the map $\varphi : M \to N$

**EXAMPLES:**

Graded algebra of mixed forms on a 3-dimensional manifold:

```python
sage: M = Manifold(3, 'M')
sage: X.<x,y,z> = M.chart()
sage: Omega = M.mixed_form_algebra(); Omega
Graded algebra Omega^*(M) of mixed differential forms on the 3-dimensional differentiable manifold M

sage: Omega.category()
Join of Category of graded algebras over Symbolic Ring and Category of chain complexes over Symbolic Ring

sage: Omega.base_ring()
Symbolic Ring

sage: Omega.vector_field_module()
Free module X(M) of vector fields on the 3-dimensional differentiable manifold M
```

Elements can be created from scratch:

```python
sage: A = Omega(0); A
Mixed differential form zero on the 3-dimensional differentiable manifold M
sage: A is Omega.zero()
True

sage: B = Omega(1); B
Mixed differential form one on the 3-dimensional differentiable manifold M
sage: B is Omega.one()
True

sage: C = Omega([2,0,0,0]); C
Mixed differential form on the 3-dimensional differentiable manifold M
```

There are some important coercions implemented:

```python
sage: Omega0 = M.scalar_field_algebra(); Omega0
Algebra of differentiable scalar fields on the 3-dimensional differentiable manifold M
sage: Omega.has_coerce_map_from(Omega0)
True
```

(continues on next page)
Element

alias of MixedForm

cohomology(*args, **kwargs)

Return the de Rham cohomology of the de Rham complex self.

The \( k \)-th de Rham cohomology is given by

\[
H^k_{\text{dR}}(M, \varphi) = \ker(d_k)/\text{im}(d_{k-1}) .
\]

The corresponding ring is given by

\[
H^*_{\text{dR}}(M, \varphi) = \bigoplus_{k=0}^n H^k_{\text{dR}}(M, \varphi),
\]

endowed with the cup product as multiplication induced by the wedge product.

See also:

See DeRhamCohomologyRing for details.

EXAMPLES:

sage: M = Manifold(3, 'M', latex_name=r'\mathcal{M}')
sage: A = M.mixed_form_algebra()
sage: A.cohomology()
De Rham cohomology ring on the 3-dimensional differentiable manifold M

differential (degree=None)

Return the differential of the de Rham complex self given by the exterior derivative.

INPUT:

- degree – (default: None) degree of the differential operator; if none is provided, the differential operator on self is returned.

EXAMPLES:

sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: C = M.de_rham_complex()

(continues on next page)
sage: d = C.differential(); d
Generic endomorphism of Graded algebra Omega^*(M) of mixed
differential forms on the 2-dimensional differentiable manifold M
sage: d0 = C.differential(0); d0
Generic morphism:
  From: Algebra of differentiable scalar fields on the
  2-dimensional differentiable manifold M
  To:  Free module Omega^1(M) of 1-forms on the 2-dimensional
  differentiable manifold M
sage: f = M.scalar_field(x, name ='f'); f.display()
  f: M → ℝ
    (x, y) ↦ x
sage: d0(f).display()
  df = dx

homology (*args, **kwargs)
Return the de Rham cohomology of the de Rham complex self.
The $k$-th de Rham cohomology is given by
$$H^k_{\text{dR}}(M, \varphi) = \ker(d_k) / \text{im}(d_{k-1}).$$
The corresponding ring is given by
$$H^*_\text{dR}(M, \varphi) = \bigoplus_{k=0}^n H^k_{\text{dR}}(M, \varphi),$$
endowed with the cup product as multiplication induced by the wedge product.
See also:
See DeRhamCohomologyRing for details.
EXAMPLES:

```sage
sage: M = Manifold(3, 'M', latex_name=r'\mathcal{M}')
sage: A = M.mixed_form_algebra()
sage: A.cohomology()
De Rham cohomology ring on the 3-dimensional differentiable
manifold M
```

irange (start=None)
Single index generator.

INPUT:
- start – (default: None) initial value $i_0$ of the index between 0 and $n$, where $n$ is the manifold’s
dimension; if none is provided, the value 0 is assumed

OUTPUT:
- an iterable index, starting from $i_0$ and ending at $n$, where $n$ is the manifold’s dimension

EXAMPLES:

```sage
sage: M = Manifold(3, 'M')
sage: A = M.mixed_form_algebra()
sage: list(A.irange())
[0, 1, 2, 3]
```
sage: list(A.irange(2))
[2, 3]

**lift_from_homology**(*x*)

Lift a cohomology class to the algebra of mixed differential forms.

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: C = M.de_rham_complex()
sage: H = C.cohomology()
sage: alpha = M.diff_form(1, [1,1], name='alpha')
sage: alpha.display()
alpha = dx + dy
sage: a = H(alpha); a
[alpha]
sage: C.lift_from_homology(a)
Mixed differential form alpha on the 2-dimensional differentiable manifold M
```

**one()**

Return the one of self.

EXAMPLES:

```
sage: M = Manifold(3, 'M')
sage: A = M.mixed_form_algebra()
sage: A.one()
Mixed differential form one on the 3-dimensional differentiable manifold M
```

**vector_field_module()**

Return the underlying vector field module.

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: N = Manifold(3, 'N')
sage: Phi = M.diff_map(N, name='Phi'); Phi
Differentiable map Phi from the 2-dimensional differentiable manifold M to the 3-dimensional differentiable manifold N
sage: A = M.mixed_form_algebra(Phi); A
Graded algebra Omega^*(M,Phi) of mixed differential forms along the 2-dimensional differentiable manifold M mapped into the 3-dimensional differentiable manifold N via Phi
sage: A.vector_field_module()
Module X(M,Phi) of vector fields along the 2-dimensional differentiable manifold M mapped into the 3-dimensional differentiable manifold N
```

**zero()**

Return the zero of self.

EXAMPLES:
2.10.2 Mixed Differential Forms

Let \( M \) and \( N \) be differentiable manifolds and \( \varphi : M \to N \) a differentiable map. A \emph{mixed differential form along} \( \varphi \) is an element of the graded algebra represented by \texttt{MixedFormAlgebra}. Its homogeneous components consist of differential forms along \( \varphi \). Mixed forms are useful to represent characteristic classes and perform computations of such.

AUTHORS:

- Michael Jung (2019): initial version

```python
class sage.manifolds.differentiable.mixed_form.MixedForm (parent, name=None, latex_name=None)

Bases: AlgebraElement, ModuleElementWithMutability
```

An instance of this class is a mixed form along some differentiable map \( \varphi : M \to N \) between two differentiable manifolds \( M \) and \( N \). More precisely, a mixed form \( a \) along \( \varphi : M \to N \) can be considered as a differentiable map

\[
a : M \to \bigoplus_{k=0}^n T^{(0,k)} N,
\]

where \( T^{(0,k)} \) denotes the tensor bundle of type \((0,k)\), \( \bigoplus \) the Whitney sum and \( n \) the dimension of \( N \), such that

\[
\forall x \in M, \quad a(x) \in \bigoplus_{k=0}^n \Lambda^k \left( T^*_{\varphi(x)} N \right),
\]

where \( \Lambda^k \left( T^*_{\varphi(x)} N \right) \) is the \( k \)-th exterior power of the dual of the tangent space \( T^*_{\varphi(x)} N \). Thus, a mixed differential form \( a \) consists of homogeneous components \( a_i, i = 0,1,\ldots,n \), where the \( i \)-th homogeneous component represents a differential form of degree \( i \).

The standard case of a mixed form on \( M \) corresponds to \( M = N \) with \( \varphi = \text{Id}_M \).

INPUT:

- \texttt{parent} – graded algebra of mixed forms represented by \texttt{MixedFormAlgebra} where the mixed form \texttt{self} shall belong to
- \texttt{comp} – (default: None) homogeneous components of the mixed form as a list; if none is provided, the components are set to innocent unnamed differential forms
- \texttt{name} – (default: None) name given to the mixed form
- \texttt{latex_name} – (default: None) \LaTeX\ symbol to denote the mixed form; if none is provided, the \LaTeX\ symbol is set to \texttt{name}

EXAMPLES:

Initialize a mixed form on a 2-dimensional parallelizable differentiable manifold:

```python
sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart()
sage: e_xy = c_xy.frame()
sage: A = M.mixed_form(name='A'); A
```
Mixed differential form $A$ on the 2-dimensional differentiable manifold $M$

```sage```
A.parent()
```
Graded algebra $\Omega^\ast(M)$ of mixed differential forms on the
2-dimensional differentiable manifold $M$

The default way to specify the $i$-th homogeneous component of a mixed form is by accessing it via $A[i]$ or using
```
set_comp():
```
```sage```
A = M.mixed_form(name='A')
A[0].set_expr(x)  # scalar field
A.set_comp(1)[0] = y^2 x
A.display()  # display names
A = A_0 + A_1 + A_2
A.display_expansion()  # display expansion in basis
A = x + x*y dx + x*y^2 dx\wedge dy
```

Another way to define the homogeneous components is using predefined differential forms:
```sage```
f = M.scalar_field(x, name = 'f'); f
Scalar field $f$ on the 2-dimensional differentiable manifold $M$
```
omega = M.diff_form(1, name = 'omega'); omega
1-form $\omega$ on the 2-dimensional differentiable manifold $M$
```
omega[e_xy,0] = y*x; omega.display()
omega = x*y dx
```
```
eta = M.diff_form(2, name = 'eta'); eta
2-form $\eta$ on the 2-dimensional differentiable manifold $M$
```
eta[e_xy,0,1] = y^2*x; eta.display()
eta = x*y^2 dx\wedge dy
```

The components of a mixed form $B$ can then be set as follows:
```sage```
B = M.mixed_form(name='B')
B[:,] = [f, omega, eta]; B.display()  # display names
B = f + omega + eta
```
```
B.display_expansion()  # display in coordinates
B = x + x*y dx + x^2 dx\wedge dy
```
```
B[0]
Scalar field $f$ on the 2-dimensional differentiable manifold $M$
```
```
B[1]
1-form $\omega$ on the 2-dimensional differentiable manifold $M$
```
```
B[2]
2-form $\eta$ on the 2-dimensional differentiable manifold $M$
```
```
As we can see, the names are applied. However note that the differential forms are different instances:
```
sage: f is B[0]
False
```
Alternatively, the components can be determined from scratch:
```sage```
B = M.mixed_form([[f, omega, eta], name='B'])
B.display()
B = f + omega + eta
```
```
Mixed forms are elements of an algebra so they can be added, and multiplied via the wedge product:
Manifolds, Release 10.3

```
sage: C = x*A; C
Mixed differential form x∧A on the 2-dimensional differentiable
  manifold M
sage: C.display_expansion()
x∧A = x^2 + x^2*y dx + x^2*y^2 dx∧dy
sage: D = A+C; D
Mixed differential form A+x∧A on the 2-dimensional differentiable
  manifold M
sage: D.display_expansion()
A+x∧A = x^2 + x + (x^2 + x)*y dx + (x^2 + x)*y^2 dx∧dy
sage: E = A*C; E
Mixed differential form A∧(x∧A) on the 2-dimensional differentiable
  manifold M
sage: E.display_expansion()
A∧(x∧A) = x^3 + 2*x^3*y dx + 2*x^3*y^2 dx∧dy
Coercions are fully implemented:
```
sage: F = omega*A
sage: F.display_expansion()
omega∧A = x^2*y dx
sage: G = omega+A
sage: G.display_expansion()
omega+A = x + 2*x*y dx + x*y^2 dx∧dy
Moreover, it is possible to compute the exterior derivative of a mixed form:
```
sage: dA = A.exterior_derivative(); dA.display()
dA = zero + dA_0 + dA_1
sage: dA.display_expansion()
dA = dx - x dx∧dy
Initialize a mixed form on a 2-dimensional non-parallelizable differentiable 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: e_xy = c_xy.frame(); e_uv = c_uv.frame() # define frames
sage: A = M.mixed_form(name='A')
sage: A[0].set_expr(x, c_xy)
sage: A[0].display()
A_0: M → ℝ
  on U: (x, y) ↦ x
  on W: (u, v) ↦ 1/2*u + 1/2*v
sage: A[1][0] = y*x; A[1].display(e_xy)
A_1 = x*y dx
sage: A[2][e_uv,0,1] = u*v^2; A[2].display(e_uv)
A_2 = u*v^2 du dv
sage: A.add_comp_by_continuation(e_uv, W, c_uv)
sage: A.display_expansion(e_uv)
A = 1/2*u + 1/2*v + (1/8*u^2 - 1/8*v^2) du + (1/8*u^2 - 1/8*v^2) dv + u*v^2 du∧dv
(continues on next page)
Since zero and one are special elements, their components cannot be changed:

```python
sage: z = M.mixed_form_algebra().zero()
sage: z[0] = 1
Traceback (most recent call last):
  ...:
ValueError: the components of an immutable element cannot be changed
sage: one = M.mixed_form_algebra().one()
sage: one[0] = 0
Traceback (most recent call last):
  ...:
ValueError: the components of an immutable element cannot be changed
```

`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:**

Mixed form defined by differential forms with components on different parts of the 2-sphere:

```python
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: e_xy = c_xy.frame(); e_uv = c_uv.frame()
sage: F = M.mixed_form(name='F') # No predefined components, here
sage: F[0] = M.scalar_field(x, name='f')
sage: F[1] = M.diff_form(1, {e_xy: [x, 0]}, name='omega')
sage: F[2].set_name(name='eta')
sage: F[2][e_uv,0,1] = u*v
sage: F.add_comp_by_continuation(e_uv, W, c_uv)
sage: F.add_comp_by_continuation(e_xy, W, c_xy) # Now, F is fully defined
sage: F.display_expansion(e_xy)
```

(continues on next page)
F = x + x dx - x*y/(x^8 + 4*x^6*y^2 + 6*x^4*y^4 + 4*x^2*y^6 + y^8) dx dy

\begin{verbatim}
sage: F.display_expansion(e_uv)
F = u/(u^2 + v^2) - (u^3 - u*v^2)/(u^6 + 3*u^4*v^2 + 3*u^2*v^4 + v^6) du -
    2*u^2*v/(u^6 + 3*u^4*v^2 + 3*u^2*v^4 + v^6) dv + u*v du dv
\end{verbatim}

**copy** *(name=None, latex_name=None)*

Return an exact copy of `self`.

**Note:** The name and names of the components are not copied.

**INPUT:**

- `name` – (default: `None`) name given to the copy
- `latex_name` – (default: `None`) \LaTeX{} symbol to denote the copy; if none is provided, the \LaTeX{} symbol is set to `name`

**EXAMPLES:**

Initialize a 2-dimensional manifold and differential forms:

\begin{verbatim}
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: e_xy = c_xy.frame(); e_uv = c_uv.frame()
sage: f = M.scalar_field(x, name='f', chart=c_xy)
sage: f.add_expr_by_continuation(c_uv, W)
sage: f.display()
f: M → ℝ
on U: (x, y) ↦ x
on V: (u, v) ↦ 1/2*u + 1/2*v
sage: omega = M.diff_form(1, name='omega')
sage: omega[e_xy,0] = x
sage: omega.add_comp_by_continuation(e_uv, W, c_uv)
sage: omega.display()
omega = x dx
sage: A = M.mixed_form([f, omega, 0], name='A'); A.display()
A = f + omega + zero
sage: A.display_expansion(e_uv)
A = 1/2*u + 1/2*v + (1/4*u + 1/4*v) du + (1/4*u + 1/4*v) dv
\end{verbatim}

An exact copy is made. The copy is an entirely new instance and has a different name, but has the very same values:

\begin{verbatim}
sage: B = A.copy(); B.display()
(unnamed scalar field) + (unnamed 1-form) + (unnamed 2-form)
sage: B.display_expansion(e_uv)
1/2*u + 1/2*v + (1/4*u + 1/4*v) du + (1/4*u + 1/4*v) dv
sage: A == B
True
\end{verbatim}
\textbf{derivative()}

Compute the exterior derivative of \texttt{self}.

More precisely, the \textit{exterior derivative} on \(\Omega^k(M, \varphi)\) is a linear map

\[
d_k : \Omega^k(M, \varphi) \to \Omega^{k+1}(M, \varphi),
\]

where \(\Omega^k(M, \varphi)\) denotes the space of differential forms of degree \(k\) along \(\varphi\) (see \texttt{exterior_derivative()} for further information). By linear extension, this induces a map on \(\Omega^*(M, \varphi)\):

\[
d : \Omega^*(M, \varphi) \to \Omega^*(M, \varphi).
\]

\textbf{OUTPUT:}

- a \texttt{MixedForm} representing the exterior derivative of the mixed form

\textbf{EXAMPLES:}

Exterior derivative of a mixed form on a 3-dimensional manifold:

\begin{verbatim}
sage: M = Manifold(3, 'M', start_index =1)
sage: c_xyz.<x,y,z> = M.chart()
sage: f = M.scalar_field(z^2, name =f)
sage: f.disp()
f: M → ℝ
  (x, y, z) ↦ z^2
sage: a = M.diff_form(2, name =a)
sage: a[1,2], a[1,3], a[2,3] = z+y^2, z+x, x^2
sage: a.disp()
a = (y^2 + z) dx∧dy + (x + z) dx∧dz + x^2 dy∧dz
sage: F = M.mixed_form([f, 0, a, 0], name =F); F .display()
F = f + zero + a + zero
sage: dF = F.exterior_derivative()
sage: dF.display()
dF = zero + df + dzero + da
sage: dF = F.exterior_derivative()
sage: dF.display_expansion()
dF = 2*z dz + (2*x + 1) dx∧dy∧dz
\end{verbatim}

Due to long calculation times, the result is cached:

\begin{verbatim}
sage: F.exterior_derivative() is dF
True
\end{verbatim}

\textbf{disp()}

Display the homogeneous components of the mixed form.

The output is either text-formatted (console mode) or \texttt{LaTeX}-formatted (notebook mode).

\textbf{EXAMPLES:}

\begin{verbatim}
sage: M = Manifold(2, 'M')
sage: f = M.scalar_field(name='f')
sage: omega = M.diff_form(1, name='omega')
sage: eta = M.diff_form(2, name='eta')
\end{verbatim}

(continues on next page)
Manifolds, Release 10.3

sage: F = M.mixed_form([f, omega, eta], name = 'F'); F
Mixed differential form F on the 2-dimensional differentiable manifold M
sage: F.display()  # display names of homogeneous components
F = f + omega + eta

disp_exp(frame=None, chart=None, from_chart=None)

Display the expansion in a particular basis and chart of mixed forms.

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

INPUT:

- **frame** – (default: None) vector frame with respect to which the mixed form is expanded; if None, only the names of the components are displayed
- **chart** – (default: None) chart with respect to which the components of the mixed form in the selected frame are expressed; if None, the default chart of the vector frame domain is assumed

EXAMPLES:

Display the expansion of a mixed form on a 2-dimensional non-parallelizable differentiable 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();
    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: e_xy = c_xy.frame(); e_uv = c_uv.frame()  # define frames
sage: omega = M.diff_form(1, name='omega')
sage: omega[e_xy,0] = x; omega.display(e_xy)
omega = x \, dx
sage: omega.add_comp_by_continuation(e_uv, W, c_uv)  # continuation onto M
sage: eta = M.diff_form(2, name='eta')
sage: eta[e_uv,0,1] = u*v; eta.display(e_uv)
eta = u \, v \, du \wedge dv
sage: eta.add_comp_by_continuation(e_xy, W, c_xy)  # continuation onto M
sage: F = M.mixed_form([0, omega, eta], name='F'); F
Mixed differential form F on the 2-dimensional differentiable manifold M
sage: F.display()  # display names of homogeneous components
F = zero + omega + eta
sage: F.display_expansion(e_uv)
F = (1/4*u + 1/4*v) \, du + (1/4*u + 1/4*v) \, dv + u*v \, du \wedge dv
sage: F.display_expansion(e_xy)
F = x \, dx + (2*x^2 - 2*y^2) \, dx \wedge dy

display()

Display the homogeneous components of the mixed form.

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

EXAMPLES:
sage: M = Manifold(2, 'M')
sage: f = M.scalar_field(name='f')
sage: omega = M.diff_form(1, name='omega')
sage: eta = M.diff_form(2, name='eta')
Mixed differential form F on the 2-dimensional differentiable manifold M
sage: F = M.mixed_form([f, omega, eta], name='F'); F
sage: F.display()  # display names of homogeneous components
F = f + omega + eta

\texttt{display_exp} (frame=None, chart=None, from_chart=None)

Display the expansion in a particular basis and chart of mixed forms.

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

INPUT:

- \texttt{frame} – (default: None) vector frame with respect to which the mixed form is expanded; if None, only the names of the components are displayed
- \texttt{chart} – (default: None) chart with respect to which the components of the mixed form in the selected frame are expressed; if None, the default chart of the vector frame domain is assumed

EXAMPLES:

Display the expansion of a mixed form on a 2-dimensional non-parallelizable differentiable 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()

....;
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: e_xy = c_xy.frame(); e_uv = c_uv.frame()  # define frames
sage: omega = M.diff_form(1, name='omega')

sage: omega[e_xy,0] = x; omega.display(e_xy)
omega = x \, dx
sage: omega.add_comp_by_continuation(e_uv, W, c_uv)  # continuation onto M
sage: eta = M.diff_form(2, name='eta')
sage: eta[e_uv,0,1] = u*v; eta.display(e_uv)
eta = u*v \, du∧dv
sage: eta.add_comp_by_continuation(e_xy, W, c_xy)  # continuation onto M
sage: F = M.mixed_form([0, omega, eta], name='F'); F
Mixed differential form F on the 2-dimensional differentiable manifold M
sage: F.display()  # display names of homogeneous components
F = zero + omega + eta
sage: F.display_expansion(e_uv)
F = (1/4*u + 1/4*v) \, du + (1/4*u + 1/4*v) \, dv + u*v \, du∧dv
sage: F.display_expansion(e_xy)
F = x \, dx + (2*x^2 - 2*y^2) \, dx∧dy

\texttt{display_expansion} (frame=None, chart=None, from_chart=None)

Display the expansion in a particular basis and chart of mixed forms.

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

- **frame** – (default: None) vector frame with respect to which the mixed form is expanded; if None, only the names of the components are displayed

- **chart** – (default: None) chart with respect to which the components of the mixed form in the selected frame are expressed; if None, the default chart of the vector frame domain is assumed

EXAMPLES:

Display the expansion of a mixed form on a 2-dimensional non-parallelizable differentiable manifold:

```sage
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()
....:
      intersection_name='W', restrictions1= x>0,
      restrictions2= u+v>0)
sage: inv = transf.inverse()
sage: W = U.intersection(V)
sage: e_xy = c_xy.frame(); e_uv = c_uv.frame()       # define frames
sage: omega = M.diff_form(1, name='omega')
sage: omega[e_xy,0] = x; omega.display(e_xy)
omega = x dx
sage: omega.add_comp_by_continuation(e_uv, W, c_uv) # continuation onto M
sage: eta = M.diff_form(2, name='eta')
sage: eta[e_uv,0,1] = u*v; eta.display(e_uv)
eta = u*v du dv
sage: F = M.mixed_form([0, omega, eta], name='F'); F
Mixed differential form F on the 2-dimensional differentiable manifold M
sage: F.display()  # display names of homogeneous components
F = zero + omega + eta
sage: F.display_expansion(e_uv)
F = (1/4*u + 1/4*v) du + (1/4*u + 1/4*v) dv + u*v du dv
sage: F.display_expansion(e_xy)
F = x dx + (2*x^2 - 2*y^2) dx dy
```

**exterior_derivative()**

Compute the exterior derivative of self.

More precisely, the **exterior derivative** on $\Omega^k(M, \varphi)$ is a linear map

\[ d_k : \Omega^k(M, \varphi) \to \Omega^{k+1}(M, \varphi), \]

where $\Omega^k(M, \varphi)$ denotes the space of differential forms of degree $k$ along $\varphi$ (see **exterior_derivative()** for further information). By linear extension, this induces a map on $\Omega^*(M, \varphi)$:

\[ d : \Omega^*(M, \varphi) \to \Omega^*(M, \varphi). \]

OUTPUT:

- a **MixedForm** representing the exterior derivative of the mixed form

EXAMPLES:

Exterior derivative of a mixed form on a 3-dimensional manifold:
```
sage: M = Manifold(3, 'M', start_index=1)
sage: c_xyz.<x,y,z> = M.chart()
sage: f = M.scalar_field(z^2, name='f')
sage: f.disp()
f: M → ℝ
  (x, y, z) ↦ z^2
sage: a = M.diff_form(2, 'a')
sage: a[1,2], a[1,3], a[2,3] = z+y^2, z+x, x^2
sage: a.disp()
a = (y^2 + z) dx∧dy + (x + z) dx∧dz + x^2 dy∧dz
sage: F = M.mixed_form([f, 0, a, 0], name='F'); F.display()
F = f + zero + a + zero
sage: dF = F.exterior_derivative()
sage: dF.display()
dF = 2*z dz + (2*x + 1) dx∧dy∧dz
```

Due to long calculation times, the result is cached:
```
sage: F.exterior_derivative() is dF
True
```

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

Single index generator.

**INPUT:**

- start – (default: None) initial value $i_0$ of the index between 0 and $n$, where $n$ is the manifold’s dimension; if none is provided, the value 0 is assumed

**OUTPUT:**

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

**EXAMPLES:**

```
sage: M = Manifold(3, 'M')
sage: a = M.mixed_form(name='a')
sage: list(a.irange())
[0, 1, 2, 3]
sage: list(a.irange(2))
[2, 3]
```

**restrict** *(subdomain, dest_map=None)*

Return the restriction of `self` to some subdomain.

**INPUT:**

- subdomain = *DifferentiableManifold*; open subset $U$ of the domain of `self`
- dest_map = *DiffMap* (default: None); destination map $\Psi : U \rightarrow V$, where $V$ is an open subset of the manifold $N$ where the mixed form takes it values; if None, the restriction of $\Phi$ to $U$ is used, $\Phi$ being the differentiable map $S \rightarrow M$ associated with the mixed form

**OUTPUT:**

- *MixedForm* representing the restriction
EXAM PLES:

Initialize 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: e_xy = c_xy.frame(); e_uv = c_uv.frame()
```

And predefine some forms:

```
sage: f = M.scalar_field(x^2, name = f, chart = c_xy)
sage: f.add_expr_by_continuation(c_uv, W)
sage: omega = M.diff_form(1, name = 'omega')
sage: omega[e_xy,0] = y^2
sage: omega.add_comp_by_continuation(e_uv, W, c_uv)
sage: eta = M.diff_form(2, name = 'eta')
sage: eta[e_xy,0,1] = x^2*y^2
sage: eta.add_comp_by_continuation(e_uv, W, c_uv)
```

Now, a mixed form can be restricted to some subdomain:

```
sage: F = M.mixed_form([f, omega, eta], name = 'F')
sage: FV = F.restrict(V); FV
Mixed differential form F on the Open subset V of the 2-dimensional
differentiable manifold M
sage: FV[:]
[Scalar field f on the Open subset V of the 2-dimensional
differentiable manifold M,
1-form omega on the Open subset V of the 2-dimensional
differentiable manifold M,
2-form eta on the Open subset V of the 2-dimensional
differentiable manifold M]
sage: FV.display_expansion(e_uv)
F = u^2/(u^4 + 2*u^2*v^2 + v^4) - (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 - 2*u*v^3/(u^8 + 4*u^6*v^2 + 6*u^4*v^4 + 4*u^2*v^6 + v^8) dv - u^2*v^2/(u^12 + 6*u^10*v^2 + 15*u^8*v^4 + 20*u^6*v^6 + 15*u^4*v^8 + 6*u^2*v^10 + v^12) du∧dv
```

set_comp (i)

Return the i-th homogeneous component for assignment.

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: A = M.mixed_form(name = 'A')
sage: A.set_comp(0).set_expr(x^2) # scalar field
sage: A.set_comp(1)[:,] = [-y, x]
sage: A.set_comp(2)[:,0,1] = x-y
```

(continues on next page)
sage: A.display()  
A = A_0 + A_1 + A_2  
sage: A.display_expansion()  
A = x^2 - y dx + x dy + (x - y) dx dy

**set_immutable()**

Set self and homogeneous components of self immutable.

**EXAMPLES:**

sage: M = Manifold(2, 'M')  
sage: X.<x,y> = M.chart()  
sage: f = M.scalar_field(x^2, name='f')  
sage: A = M.mixed_form([f, 0, 0], name='A')  
sage: A.set_immutable()  
sage: A.is_immutable()  
True  
sage: A[0].is_immutable()  
True  
sage: f.is_immutable()  
False

**set_name** *(name=None, latex_name=None, apply_to_comp=True)*

Redefine the string and LaTeX representation of the object.

**INPUT:**

- **name** – (default: None) name given to the mixed form
- **latex_name** – (default: None) \( \LaTeX \) symbol to denote the mixed form; if none is provided, the \( \LaTeX \) symbol is set to name
- **apply_to_comp** – (default: True) if True all homogeneous components will be renamed accordingly; if False only the mixed form will be renamed

**EXAMPLES:**

Rename a mixed form:

sage: M = Manifold(4, 'M')  
sage: F = M.mixed_form(name='dummy', latex_name=r'\text{ugly}'); F  
Mixed differential form dummy on the 4-dimensional differentiable manifold M  
sage: latex(F)  
\text{ugly}  
sage: F.set_name(name='F', latex_name=r'\mathcal{F}'); F  
Mixed differential form F on the 4-dimensional differentiable manifold M  
sage: latex(F)  
\mathcal{F}

If not stated otherwise, all homogeneous components are renamed accordingly:

sage: F.display()  
F = F_0 + F_1 + F_2 + F_3 + F_4

Setting the argument **set_all** to **False** prevents the renaming in the homogeneous components:
To rename a homogeneous component individually, we simply access the homogeneous component and use its `set_name()` method:

```sage
F[0].set_name(name='g'); F.display()
```

\[
\eta = g + F_1 + F_2 + F_3 + F_4
\]

### set_restriction (rst)

Set a (component-wise) restriction of `self` to some subdomain.

**INPUT:**

- `rst` — `MixedForm` of the same type as `self`, defined on a subdomain of the domain of `self`

**EXAMPLES:**

Initialize the 2-sphere:

```sage
M = Manifold(2, 'M') # the 2-dimensional sphere S^2
U = M.open_subset('U') # complement of the North pole
\( c_{xy}.\langle x,y \rangle = U.\text{chart}() \) # stereographic coordinates from the North pole
V = M.open_subset('V') # complement of the South pole
\( c_{uv}.\langle u,v \rangle = V.\text{chart}() \) # stereographic coordinates from the South pole
M.declare_union(U,V) # S^2 is the union of U and V
xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),

\( \text{intersection name='W', restrictions1= x^2+y^2!=0,}
\)

\( \text{restrictions2= u^2+v^2!=0) }
\)

\( \text{W = U.intersection(V)}
\)

\( \text{e_xy = c_xy.frame(); e_uv = c_uv.frame()}
\)

And define some forms on the subset `U`:

```sage
f = U.scalar_field(x, name='f', chart=c_xy)
omega = U.diff_form(1, name='omega')
omega[e_xy,0] = y
AU = U.mixed_form([f, omega, 0], name='A'); AU
```

A mixed form on `M` can be specified by some mixed form on a subset:

```sage
A = M.mixed_form(name='A'); A
```

```sage
A.set_restriction(AU)
```

```sage
A.display_expansion(e_xy)
```

\[
A = x + y \, dx
\]

```sage
A.add_comp_by_continuation(e_uv, W, c_uv)
```

```sage
A.display_expansion(e_uv)
```

\[
A = u/(u^2+v^2)*(u^2+v^2)/(u^6+3*u^4*v^2+3*u^2*v^4+v^6) \, du - 2*u*v^2/(u^6+3*u^4*v^2+3*u^2*v^4+v^6) \, dv
\]

```sage
A.restrict(U) == AU
```

True
wedge (other)

Wedge product on the graded algebra of mixed forms.

More precisely, the wedge product is a bilinear map

\[ \wedge : \Omega^k(M, \varphi) \times \Omega^l(M, \varphi) \rightarrow \Omega^{k+l}(M, \varphi), \]

where \( \Omega^k(M, \varphi) \) denotes the space of differential forms of degree \( k \) along \( \varphi \). By bilinear extension, this induces a map

\[ \wedge : \Omega^*(M, \varphi) \times \Omega^*(M, \varphi) \rightarrow \Omega^*(M, \varphi)^\wedge \]

and equips \( \Omega^*(M, \varphi) \) with a multiplication such that it becomes a graded algebra.

INPUT:

- other — mixed form in the same algebra as self

OUTPUT:

- the mixed form resulting from the wedge product of self with other

EXAMPLES:

Initialize a mixed form on a 3-dimensional manifold:

```
sage: M = Manifold(3, 'M')
sage: c_xyz.<x,y,z> = M.chart()
sage: f = M.scalar_field(x, name='f')
sage: f.display()
f: M → ℝ
  (x, y, z) ↦ x
sage: g = M.scalar_field(y, name='g')
sage: g.display()
g: M → ℝ
  (x, y, z) ↦ y
sage: omega = M.diff_form(1, name='omega')
sage: omega[0] = x
sage: omega.display()
omega = x dx
sage: eta = M.diff_form(1, name='eta')
sage: eta[1] = y
sage: eta.display()
eta = y dy
sage: mu = M.diff_form(2, name='mu')
sage: mu[0,2] = z
sage: mu.display()
mu = z dx∧dz
sage: A = M.mixed_form([f, omega, mu, 0], name='A')
sage: A.display_expansion()
A = x + x dx + z dx∧dz
sage: B = M.mixed_form([g, eta, mu, 0], name='B')
sage: B.display_expansion()
B = y + y dy + z dx∧dz
```

The wedge product of A and B yields:

```
sage: C = A.wedge(B); C
Mixed differential form A∧B on the 3-dimensional differentiable manifold M
```

(continues on next page)
2.11 De Rham Cohomology

Let \( M \) and \( N \) be differentiable manifolds and \( \varphi : M \to N \) be a differentiable map. Then the associated de Rham complex is given by

\[
0 \to \Omega^0(M, \varphi) \xrightarrow{d_0} \Omega^1(M, \varphi) \xrightarrow{d_1} \cdots \xrightarrow{d_{n-1}} \Omega^n(M, \varphi) \xrightarrow{d_n} 0,
\]

where \( \Omega^k(M, \varphi) \) is the module of differential forms of degree \( k \), and \( d_k \) is the associated exterior derivative. Then the \( k \)-th de Rham cohomology group is given by

\[
H^k_{\text{DR}}(M, \varphi) = \ker(d_k)/\text{im}(d_{k-1}),
\]

and the corresponding ring is obtained by

\[
H^*_{\text{DR}}(M, \varphi) = \bigoplus_{k=0}^n H^k_{\text{DR}}(M, \varphi).
\]

The de Rham cohomology ring is implemented via \texttt{DeRhamCohomologyRing}. Its elements, the cohomology classes, are represented by \texttt{DeRhamCohomologyClass}.

AUTHORS:
- Michael Jung (2021): initial version
Bases: `AlgebraElement`

Define a cohomology class in the de Rham cohomology ring.

**INPUT:**

- `parent` – de Rham cohomology ring represented by an instance of `DeRhamCohomologyRing`
- `representative` – a closed (mixed) differential form representing the cohomology class

**Note:** The current implementation only provides basic features. Comparison via exact forms are not supported at the time being.

**EXAMPLES:**

```python
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: C = M.de_rham_complex()
sage: H = C.cohomology()
sage: omega = M.diff_form(1, [1,1], name='omega')
sage: u = H(omega); u
[omega]
```

Cohomology classes can be lifted to the algebra of mixed differential forms:

```python
sage: u.lift()
Mixed differential form omega on the 2-dimensional differentiable manifold M
```

However, comparison of two cohomology classes is limited at the time being:

```python
sage: eta = M.diff_form(1, [1,-1], name='eta')
sage: H(eta) == u
True
e
NotImplementedError: comparison via exact forms is currently not supported
```

`cup(other)`

Cup product of two cohomology classes.

**INPUT:**

- `other` – another cohomology class in the de Rham cohomology

**EXAMPLES:**

```python
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: C = M.de_rham_complex()
sage: H = C.cohomology()
sage: omega = M.diff_form(1, [1,1], name='omega')
sage: eta = M.diff_form(1, [1,-1], name='eta')
sage: H(omega).cup(H(eta))
[omega∧eta]
```
lift()  
Return a representative of self in the associated de Rham complex.

EXAMPLES:

```plaintext
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: C = M.de_rham_complex()
sage: H = C.cohomology()
sage: omega = M.diff_form(2, name =omega)
sage: omega[0,1] = x
sage: omega.display()
  omega = x dx∧dy
sage: u = H(omega); u
  [omega]
sage: u.representative()
 Mixed differential form omega on the 2-dimensional differentiable
  manifold M
```

representative()  
Return a representative of self in the associated de Rham complex.

EXAMPLES:

```plaintext
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: C = M.de_rham_complex()
sage: H = C.cohomology()
sage: omega = M.diff_form(2, name =omega)
sage: omega[0,1] = x
sage: omega.display()
  omega = x dx∧dy
sage: u = H(omega); u
  [omega]
sage: u.representative()
 Mixed differential form omega on the 2-dimensional differentiable
  manifold M
```

class sage.manifolds.differentiable.de_rham_cohomology.DeRhamCohomologyRing(de_rham_complex)

Bases: Parent, UniqueRepresentation

The de Rham cohomology ring of a de Rham complex.

This ring is naturally endowed with a multiplication induced by the wedge product, called cup product, see DeRhamCohomologyClass.cup().

Note: The current implementation only provides basic features. Comparison via exact forms are not supported at the time being.

INPUT:

- de_rham_complex – a de Rham complex, typically an instance of MixedFormAlgebra

EXAMPLES:

We define the de Rham cohomology ring on a parallelizable manifold $M$: 
De Rham cohomology ring on the 2-dimensional differentiable manifold $M$:

Cohomology classes can be lifted to the algebra of mixed differential forms:

The ring admits a zero and unit element:

Element

alias of $\text{DeRhamCohomologyClass}$

one()

Return the one element of self.

EXAMPLES:

zero()

Return the zero element of self.

EXAMPLES:

(continues on next page)
Mixed differential form zero on the 2-dimensional differentiable manifold M

2.12 Alternating Multivector Fields

2.12.1 Multivector Field Modules

The set $A^p(U, \Phi)$ of $p$-vector fields along a differentiable manifold $U$ with values on a differentiable manifold $M$ via a differentiable map $\Phi : U \to M$ (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 $A^p(U, \Phi)$:

- $\text{MultivectorModule}$ for $p$-vector fields with values on a generic (in practice, not parallelizable) differentiable manifold $M$
- $\text{MultivectorFreeModule}$ for $p$-vector fields with values on a parallelizable manifold $M$

AUTHORS:
- Eric Gourgoulhon (2017): initial version

REFERENCES:

class sage.manifolds.differentiable.multivector_module.MultivectorFreeModule(vector_field_module, degree)

Bases: ExtPowerFreeModule

Free module of multivector fields of a given degree $p$ ($p$-vector fields) along a differentiable manifold $U$ with values on a parallelizable manifold $M$.

Given a differentiable manifold $U$ and a differentiable map $\Phi : U \to M$ to a parallelizable manifold $M$ of dimension $n$, the set $A^p(U, \Phi)$ of $p$-vector fields (i.e. alternating tensor fields of type $(p, 0)$) along $U$ with values on $M$ is a free module of rank $\binom{n}{p}$ over $C^k(U)$, the commutative algebra of differentiable scalar fields on $U$ (see $\text{DiffScalarFieldAlgebra}$). The standard case of $p$-vector fields 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 $\mathbb{R}$).

Note: This class implements $A^p(U, \Phi)$ in the case where $M$ is parallelizable; $A^p(U, \Phi)$ is then a free module. If $M$ is not parallelizable, the class $\text{MultivectorModule}$ must be used instead.

INPUT:
- vector_field_module – free module $\mathcal{X}(U, \Phi)$ of vector fields along $U$ associated with the map $\Phi : U \to V$
- degree – positive integer; the degree $p$ of the multivector fields
EXAMPLES:

Free module of 2-vector fields on a parallelizable 3-dimensional manifold:

```python
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.multivector_module(2) ; A
Free module A^2(M) of 2-vector fields on the 3-dimensional
differentiable manifold M
sage: latex(A)
A^{2}(M)
A is nothing but the second exterior power of XM, i.e. we have A^2(M) = \Lambda^2(X(M)) (see ExtPowerFreeModule):

```python
sage: A is XM.exterior_power(2)
True
A^2(M) is a module over the algebra C^k(M) of (differentiable) scalar fields on M:

```python
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:

```python
sage: A(0)
2-vector field 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:

```python
sage: comp = [[0,3*x,-z],[-3*x,0,4],[z,-4,0]]
sage: a = A(comp, frame=X.frame(), name='a') ; a
2-vector field a on the 3-dimensional differentiable manifold M
sage: a.display()
a = 3*x \partial/\partial x \wedge \partial y - z \partial/\partial x \wedge \partial z + 4 \partial/\partial y \wedge \partial z
```

2.12. Alternating Multivector Fields
An alternative is to construct the 2-vector field 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 ∂/∂x∧∂/∂y - z ∂/∂x∧∂/∂z + 4 ∂/∂y∧∂/∂z
```

The module $A^1(M)$ is nothing but $\mathcal{X}(M)$ (the free module of vector fields on $M$):

```
sage: A1 = M.multivector_module(1) ; A1
Free module $\mathcal{X}(M)$ of vector fields on the 3-dimensional differentiable manifold $M$
sage: A1 is XM
True
```

There is a coercion map $A^p(M) \to T^{(p,0)}(M)$:

```
sage: T20 = M.tensor_field_module((2,0)); T20
Free module $T^{(2,0)}(M)$ of type-(2,0) tensors fields on the 3-dimensional differentiable manifold $M$
sage: T20.has_coerce_map_from(A)
True
```

but of course not in the reverse direction, since not all contravariant tensor field is alternating:

```
sage: A.has_coerce_map_from(T20)
False
```

The coercion map $A^2(M) \to T^{(2,0)}(M)$ in action:

```
sage: T20 = M.tensor_field_module((2,0)) ; T20
Free module $T^{(2,0)}(M)$ of type-(2,0) tensors fields on the 3-dimensional differentiable manifold $M$
sage: ta = T20(a) ; ta
Tensor field $a$ of type $(2,0)$ on the 3-dimensional differentiable manifold $M$
sage: ta.display()
a = 3*x ∂/∂x⊗∂/∂y - z ∂/∂x⊗∂/∂z - 3*x ∂/∂y⊗∂/∂x + 4 ∂/∂y⊗∂/∂z + z ∂/∂z⊗∂/∂x - 4 ∂/∂z⊗∂/∂y
```

There is also coercion to subdomains, which is nothing but the restriction of the multivector field to some subset of its domain:

```
sage: U = M.open_subset('U', coord_def={X: x^2+y^2<1})
sage: B = U.multivector_module(2) ; B
Free module $A^2(U)$ of 2-vector fields 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-vector field $a$ on the Open subset $U$ of the 3-dimensional
```
differentiable manifold M
sage: a_U.display()
a = 3*x ∂/∂x∧∂/∂y - z ∂/∂x∧∂/∂z + 4 ∂/∂y∧∂/∂z

Element
  alias of MultivectorFieldParal
class sage.manifolds.differentiable.multivector_module.MultivectorModule(vertex_field_module, degree)

Bases: UniqueRepresentation, Parent

Module of multivector fields of a given degree \( p \) (\( p \)-vector fields) along a differentiable manifold \( U \) with values on a differentiable manifold \( M \).

Given a differentiable manifold \( U \) and a differentiable map \( \Phi : U \to M \) to a differentiable manifold \( M \), the set \( A^p(U, \Phi) \) of \( p \)-vector fields (i.e. alternating tensor fields of type \( (p,0) \) 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 \( p \)-vector fields 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 \( \mathbb{R} \)).

**Note:** This class implements \( A^p(U, \Phi) \) in the case where \( M \) is not assumed to be parallelizable; the module \( A^p(U, \Phi) \) is then not necessarily free. If \( M \) is parallelizable, the class MultivectorFreeModule must be used instead.

**INPUT:**

- \( \text{vector_field_module} \) – module \( X(U, \Phi) \) of vector fields along \( U \) with values on \( M \) via the map \( \Phi : U \to M \)
- \( \text{degree} \) – positive integer; the degree \( p \) of the multivector fields

**EXAMPLES:**

Module of 2-vector fields on a non-parallelizable 2-dimensional manifold:

```sage
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()
    ....: 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.multivector_module(2) ; A
Module A^2(M) of 2-vector fields on the 2-dimensional differentiable manifold M
sage: latex(A)
A^{(2)}\left(M\right)
```

\( A \) is nothing but the second exterior power of \( XM \), i.e. we have \( A^2(M) = \Lambda^2(X(M)) \):
Manifolds, Release 10.3

```
sage: A is XM.exterior_power(2)
True
```

Modules of multivector fields are unique:

```
sage: A is M.multivector_module(2)
True
```

$A^2(M)$ 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() is XM
True
```

Elements can be constructed from $A()$. In particular, 0 yields the zero element of $A$:

```
sage: z = A(0) ; z
2-vector field 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-vector field a on the 2-dimensional differentiable manifold M
sage: a.add_comp_by_continuation(eV, W, c_uv) # finishes initializ. of a
sage: a.display(eU)
a = 3*x ∂/∂x∧∂/∂y
sage: a.display(eV)
a = (-3*u - 3*v) ∂/∂u∧∂/∂v
```

An alternative is to construct the 2-vector field 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 ∂/∂x∧∂/∂y
sage: a.display(eV)
a = (-3*u - 3*v) ∂/∂u∧∂/∂v
```
The module $A^1(M)$ is nothing but the dual of $\mathfrak{X}(M)$ (the module of vector fields on $M$):

\begin{Verbatim}
sage: A1 = M.multivector_module(1) ; A1
Module X(M) of vector fields on the 2-dimensional differentiable manifold M
sage: A1 is XM
True
\end{Verbatim}

There is a coercion map $A^p(M) \rightarrow T^{(p,0)}(M)$:

\begin{Verbatim}
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
sage: T20.has_coerce_map_from(A)
True
\end{Verbatim}

but of course not in the reverse direction, since not all contravariant tensor field is alternating:

\begin{Verbatim}
sage: A.has_coerce_map_from(T20)
False
\end{Verbatim}

The coercion map $A^2(M) \rightarrow T^{(2,0)}(M)$ in action:

\begin{Verbatim}
sage: ta = T20(a) ; ta
Tensor field a of type (2,0) on the 2-dimensional differentiable manifold M
sage: ta.display(eU)
a = 3*x ∂/∂x⊗∂/∂y - 3*x ∂/∂y⊗∂/∂x
sage: a.display(eU)
a = 3*x ∂/∂x∂y
sage: ta.display(eV)
a = (-3*u - 3*v) ∂/∂u⊗∂/∂v + (3*u + 3*v) ∂/∂v⊗∂/∂u
sage: a.display(eV)
a = (-3*u - 3*v) ∂/∂u∂v
\end{Verbatim}

There is also coercion to subdomains, which is nothing but the restriction of the multivector field to some subset of its domain:

\begin{Verbatim}
sage: A2U = U.multivector_module(2) ; A2U
Free module A^2(U) of 2-vector fields on the Open subset U of the 2-dimensional differentiable manifold M
sage: A2U.has_coerce_map_from(A)
True
sage: a_U = A2U(a) ; a_U
2-vector field a on the Open subset U of the 2-dimensional differentiable manifold M
sage: a_U.display(eU)
a = 3*x ∂/∂x∧∂/∂y
\end{Verbatim}

Element

alias of MultivectorField

base_module()

Return the vector field module on which the multivector field module self is constructed.

OUTPUT:

- a VectorFieldModule representing the module on which self is defined
EXAMPLES:

```python
sage: M = Manifold(3, 'M')
sage: A2 = M.multivector_module(2) ; A2
Module A^2(M) of 2-vector fields 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.multivector_module(2) ; A2U
Module A^2(U) of 2-vector fields 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 multivector fields in `self`.

**OUTPUT:**

- integer `p` such that `self` is a set of `p`-vector fields

**EXAMPLES:**

```python
sage: M = Manifold(3, 'M')
sage: M.multivector_module(2).degree()
2
sage: M.multivector_module(3).degree()
3
```

**zero()**

Return the zero of `self`.

**EXAMPLES:**

```python
sage: M = Manifold(3, 'M')
sage: A2 = M.multivector_module(2)
sage: A2.zero()
2-vector field zero on the 3-dimensional differentiable
manifold M
```

## 2.12.2 Multivector Fields

Let $U$ and $M$ be two differentiable manifolds. Given a positive integer $p$ and a differentiable map $\Phi : U \to M$, a **multivector field of degree $p$**, or **$p$-vector field**, along $U$ with values on $M$ is a field along $U$ of alternating contravariant tensors of rank $p$ in the tangent spaces to $M$. The standard case of a multivector 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 $\mathbb{R}$).

Two classes implement multivector fields, depending whether the manifold $M$ is parallelizable:

- **MultivectorFieldParal** when $M$ is parallelizable
- **MultivectorField** when $M$ is not assumed parallelizable.
AUTHORS:
• Eric Gourgoulhon (2017): initial version

REFERENCES:
• R. L. Bishop and S. L. Goldberg (1980) [BG1980]
• C.-M. Marle (1997) [Mar1997]

class sage.manifolds.differentiable.multivectorfield.MultivectorField(vector_field_module, degree, name=None, latex_name=None)

Bases: TensorField

Multivector field 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 multivector field of degree $p$ (or $p$-vector field) along $U$ with values on $M \supset \Phi(U)$ is a differentiable map

$$a : U \longrightarrow T^{(p,0)} M$$

($T^{(p,0)} M$ being the tensor bundle of type $(p,0)$ over $M$) such that

$$\forall x \in U, \ a(x) \in \Lambda^p(T_{\Phi(x)} M),$$

where $T_{\Phi(x)} M$ is the vector space tangent to $M$ at $\Phi(x)$ and $\Lambda^p$ stands for the exterior power of degree $p$ (cf. ExtPowerFreeModule). In other words, $a(x)$ is an alternating contravariant tensor of degree $p$ of the tangent vector space $T_{\Phi(x)} M$.

The standard case of a multivector field 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 $\mathbb{R}$).

Note: If $M$ is parallelizable, the class MultivectorFieldParal must be used instead.

INPUT:

• vector_field_module – module $\mathfrak{X}(U, \Phi)$ of vector fields along $U$ with values on $M$ via the map $\Phi$
• degree – the degree of the multivector field (i.e. its tensor rank)
• name – (default: None) name given to the multivector field
• latex_name – (default: None) LaTeX symbol to denote the multivector field; if none is provided, the LaTeX symbol is set to name

EXAMPLES:

Multivector field of degree 2 on a non-parallelizable 2-dimensional manifold:

```sage
M = Manifold(2, 'M')
U = M.open_subset('U') ; V = M.open_subset('V')
M.declare_union(U,V) # M is the union of U and V
c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart()
xy_to_uv = c_xy.transition_map(c_uv, (x+y, x-y),
intersection_name='W',
```

(continues on next page)
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.multivector_field(2, name='a'); a
2-vector field a on the 2-dimensional differentiable manifold M
sage: a.parent()
Module A^2(M) of 2-vector fields 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) \partial/\partial x \wedge \partial/\partial y
sage: a.display(eV)
a = (-1/4*u^3 + 1/4*u*v^2 - 1/4*v^3 + 1/4*(u^2 - 8)*v - 2*u) \partial/\partial u \wedge \partial/\partial v

It is also possible to set the components while defining the 2-vector field definition, via a dictionary whose keys are the vector frames:

sage: a1 = M.multivector_field(2, {eU: [[0, x*y^2 + 2*x], [-x*y^2 - 2*x, 0]], name='a1')
sage: a1.add_comp_by_continuation(eV, W, c_uv)
sage: a1 == a
True

The exterior product of two vector fields is a 2-vector field:

sage: a = M.vector_field({eU: [-y, x]}, name='a')
sage: b = M.vector_field({eU: [1+x*y, x^2]}, name='b')
sage: s = a.wedge(b) ; s
2-vector field a\wedge b on the 2-dimensional differentiable manifold M
sage: s.display(eU)
a\wedge b = (-2*x^2*y - x) \partial/\partial x \wedge \partial/\partial y
sage: s.display(eV)
a\wedge b = (1/2*u^3 - 1/2*u*v^2 - 1/2*v^3 + 1/2*(u^2 + 2)*v + u) \partial/\partial u \wedge \partial/\partial v

Multiplying a 2-vector field by a scalar field results in another 2-vector field:

sage: f = M.scalar_field({c_xy: (x + y)^2, c_uv: u^2}, name='f')
sage: s = f*s ; s
2-vector field f*(a\wedge b) on the 2-dimensional differentiable manifold M
sage: s.display(eU)
f*(a\wedge b) = (-2*x^2*y^3 - x^3 - (4*x^2 + x)*y^2 - 2*(x^4 + x^2)*y) \partial/\partial x \wedge \partial/\partial y
sage: s.display(eV)
f*(a\wedge b) = (1/2*u^5 - 1/2*u^3*v^2 - 1/2*u^2*v^3 + u^3 + 1/2*(u^4 + 2*u^2)*v) \partial/\partial u \wedge \partial/\partial v

\textbf{bracket} (other)

Return the Schouten-Nijenhuis bracket of self with another multivector field.
The Schouten-Nijenhuis bracket extends the Lie bracket of vector fields (cf. `bracket()`) to multivector fields.

Denoting by $A^p(M)$ the $C^k(M)$-module of $p$-vector fields on the $C^k$-differentiable manifold $M$ over the field $K$ (cf. `MultivectorModule`), the Schouten-Nijenhuis bracket is a $K$-bilinear map

$$A^p(M) \times A^q(M) \rightarrow A^{p+q-1}(M)$$

which obeys the following properties:

- if $p = 0$ and $q = 0$, (i.e. $a$ and $b$ are two scalar fields), $[a, b] = 0$
- if $p = 0$ (i.e. $a$ is a scalar field) and $q \geq 1$, $[a, b] = -\iota_\delta b$ (minus the interior product of the differential of $a$ by $b$)
- if $p = 1$ (i.e. $a$ is a vector field), $[a, b] = \mathcal{L}_a b$ (the Lie derivative of $b$ along $a$)
- if $p, q > 0$, $[a, b] = \iota_{\delta a} b + (-1)^{p+1} \iota_{\delta b} a$
- for any multivector field $c$ and $(a, b) \in A^p(M) \times A^q(M)$, $[a, c]$ obeys the graded Leibniz rule
  $$[a, b \wedge c] = [a, b] \wedge c + (-1)^{p-1} q b \wedge [a, c]$$
- for $(a, b, c) \in A^p(M) \times A^q(M) \times A^{r}(M)$, the graded Jacobi identity holds:
  $$(-1)^{(p-1)(r-1)} [a, [b, c]] + (-1)^{(q-1)(r-1)} [b, [c, a]] + (-1)^{(p-1)(q-1)} [c, [a, b]] = 0$$

**Note:** There are two definitions of the Schouten-Nijenhuis bracket in the literature, which differ from each other when $p$ is even by an overall sign. The definition adopted here is that of [Mar1997], [Kos1985] and Wikipedia article Schouten-Nijenhuis_bracket. The other definition, adopted e.g. by [Nij1955], [Lic1977] and [Vai1994], is $[a, b]' = (-1)^{p+1} [a, b]$.

**INPUT:**
- other – a multivector field

**OUTPUT:**
- instance of `MultivectorField` (or of `DiffScalarField` if $p = 1$ and $q = 0$) representing the Schouten-Nijenhuis bracket $[a, b]$, where $a$ is self and $b$ is other

**EXAMPLES:**
Bracket of two vector fields on the 2-sphere:

```
sage: M = Manifold(2, 'S^2', start_index=1) # the 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() # stereographic coord. North
sage: c_uv.<u,v> = V.chart() # stereographic coord. 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()
```

(continues on next page)
sage: b.add_comp_by_continuation(e_uv, W, c_uv)
sage: s = a.bracket(b); s
Vector field [a,b] on the 2-dimensional differentiable manifold S^2
sage: s.display(e_xy)
[a,b] = (x^2 + y^2 - x + y) \partial/\partial x + (-x - 1)*y - x) \partial/\partial y

For two vector fields, the bracket coincides with the Lie derivative:

sage: s == b.lie_derivative(a)
True

Schouten-Nijenhuis bracket of a 2-vector field and a 1-vector field:

sage: c = a.wedge(b); c
2-vector field a∧b on the 2-dimensional differentiable manifold S^2
sage: s = c.bracket(a); s
2-vector field [a∧b,a] on the 2-dimensional differentiable manifold S^2
sage: s.display(e_xy)
[a∧b,a] = (x^3 + (2*x - 1)*y^2 - x^2 + 2*x*y) \partial/\partial x∧\partial/\partial y

Since \(a\) is a vector field, we have in this case:

sage: s == - c.lie_derivative(a)
True

See also: `MultivectorFieldParal.bracket()` for more examples and check of standards identities involving the Schouten-Nijenhuis bracket

**degree()**

Return the degree of `self`.

**OUTPUT:**

- integer \(p\) such that `self` is a \(p\)-vector field

**EXAMPLES:**

```python
sage: M = Manifold(3, 'M')
sage: a = M.multivector_field(2); a
2-vector field on the 3-dimensional differentiable manifold M
sage: a.degree()
2
sage: b = M.vector_field(); b
Vector field on the 3-dimensional differentiable manifold M
sage: b.degree()
1
```

**interior_product(form)**

Interior product with a differential form.

If `self` is a multivector field \(A\) of degree \(p\) and \(B\) is a differential form of degree \(q \geq p\) on the same manifold as \(A\), the interior product of \(A\) by \(B\) is the differential form \(\iota_A B\) of degree \(q - p\) defined by

\[
(\iota_A B)_{i_1...i_{q-p}} = A^{k_1...k_p} B_{k_1...k_p i_1...i_{q-p}}
\]
Note: \( A.\text{interior\_product}(B) \) yields the same result as \( A.\text{contract}(0, \ldots, p-1, B, 0, \ldots, p-1) \) (cf. \text{contract}()), but \text{interior\_product} is more efficient, the alternating character of \( A \) being not used to reduce the computation in \text{contract}().

**INPUT:**

- \( \text{form} \) – differential form \( B \) (instance of \text{DiffForm}); the degree of \( B \) must be at least equal to the degree of \( self \)

**OUTPUT:**

- scalar field (case \( p = q \)) or \text{DiffForm} (case \( p < q \)) representing the interior product \( \iota_A B \), where \( A \) is \( self \)

**See also:**

\text{interior\_product()} for the interior product of a differential form with a multivector field

**EXAMPLES:**

Interior product of a vector field \( (p = 1) \) with a 2-form \( (q = 2) \) on the 2-sphere:

```python
sage: M = Manifold(2, 'S^2', start_index=1) # the 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() # stereographic coord. North
sage: c_uv.<u,v> = V.chart() # stereographic coord. South
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),   # the standard area 2-form
.....:     intersection_name='W', restrictions1= x^2+y^2!=0,
.....:     restrictions2= u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
```

Example with \( p = 2 \) and \( q = 2 \):

```python
sage: a = M.multivector_field(2, name='a')
sage: a[e_xy,1,2] = x*y
```

(continues on next page)
sage: a.display(e_xy)
a = x*y ∂/∂x∧∂/∂y
sage: a.display(e_uv)
a = -u*v ∂/∂u∧∂/∂v
sage: s = a.interior_product(b); s
Scalar field i_a b on the 2-dimensional differentiable manifold S^2
sage: s.display()
i_a b: S^2 → ℝ
on U: (x, y) ↦ 8*x*y/(x^4 + y^4 + 2*(x^2 + 1)*y^2 + 2*x^2 + 1)
on V: (u, v) ↦ 8*u*v/(u^4 + v^4 + 2*(u^2 + 1)*v^2 + 2*u^2 + 1)

Some checks:
sage: s == a.contract(0, 1, b, 0, 1)
True
sage: s.restrict(U) == 2 * a[[e_xy,1,2]] * b[[e_xy,1,2]]
True
sage: s.restrict(V) == 2 * a[[e_uv,1,2]] * b[[e_uv,1,2]]
True

wedge (other)
Exterior product with another multivector field.

INPUT:

• other – another multivector field (on the same manifold)

OUTPUT:

• instance of MultivectorField representing the exterior product self ∧ other

EXAMPLES:

Exterior product of two vector fields on the 2-sphere:
sage: M = Manifold(2, 'S^2', start_index =1) # the 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() # stereographic coord. North
sage: c_uv.<u,v> = V.chart() # stereographic coord. 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.vector_field({e_xy: [y, x]}, name='a')
sage: a.add_comp_by_continuation(e_uv, W, c_uv)
sage: b = M.vector_field({e_xy: [x ^2 + y^2, y]}, name='b')
sage: b.add_comp_by_continuation(e_uv, W, c_uv)
sage: c = a.wedge(b); c
2-vector field a∧b on the 2-dimensional differentiable manifold S^2
sage: c.display(e_xy)
a∧b = (-x^3 - (x - 1)*y^2) ∂/∂x∧∂/∂y
sage: c.display(e_uv)
a∧b = (-v^2 + u) ∂/∂u∧∂/∂v
class sage.manifolds.differentiable.multivectorfield.MultivectorFieldParal (vector_field_module, degree, name=None, latex_name=None)

Bases: AlternatingContrTensor, TensorFieldParal

Multivector field 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 multivector field of degree $p$ (or $p$-vector field) along $U$ with values on $M \supset \Phi(U)$ is a differentiable map

$$ a : U \rightarrow T^{(p,0)}M $$

($T^{(p,0)}M$ being the tensor bundle of type $(p,0)$ over $M$) such that

$$ \forall x \in U, \ a(x) \in \Lambda^p(T_{\Phi(x)}M), $$

where $T_{\Phi(x)}M$ is the vector space tangent to $M$ at $\Phi(x)$ and $\Lambda^p$ stands for the exterior power of degree $p$ (cf. ExtPowerFreeModule). In other words, $a(x)$ is an alternating contravariant tensor of degree $p$ of the tangent vector space $T_{\Phi(x)}M$.

The standard case of a multivector field 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 $\mathbb{R}$).

**Note:** If $M$ is not parallelizable, the class MultivectorField 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 multivector field (i.e. its tensor rank)
- `name` – (default: None) name given to the multivector field
- `latex_name` – (default: None) LaTeX symbol to denote the multivector field; if none is provided, the LaTeX symbol is set to `name`

**EXAMPLES:**

A 2-vector field on a 4-dimensional manifold:

```
sage: M = Manifold(4, 'M')
sage: c_txyz.<t,x,y,z> = M.chart()
sage: a = M.multivector_field(2, name='a') ; a
2-vector field a on the 4-dimensional differentiable manifold M
sage: a.parent()
Free module $\Lambda^2(M)$ of 2-vector fields on the 4-dimensional differentiable manifold M
```

A multivector field is a tensor field of purely contravariant type:
sage: a.tensor_type()
(2, 0)

It is antisymmetric, its components being CompFullyAntiSym:

sage: a.symmetries()
no symmetry; antisymmetry: (0, 1)
sage: a[0,1] = 2*x
sage: a[1,0] = -2*x
sage: a.comp()
Fully antisymmetric 2-indices components w.r.t. Coordinate frame
(M, (∂/∂t,∂/∂x,∂/∂y,∂/∂z))
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 multivector field with respect to a given frame is displayed via the method display():

sage: a.display()  # expansion w.r.t. the default frame
a = 2*x ∂/∂t∧∂/∂x + 3 ∂/∂x∧∂/∂y
sage: latex(a.display())  # output for the notebook
a = 2 \, x \frac{\partial}{\partial t} \wedge \frac{\partial}{\partial x} + 3 \frac{\partial}{\partial x} \wedge \frac{\partial}{\partial y}

Multivector fields can be added or subtracted:

sage: b = M.multivector_field(2)
sage: b[0,1], b[0,2], b[0,3] = y, 2, x+z
sage: s = a + b ; s
2-vector field on the 4-dimensional differentiable manifold M
sage: s.display()
(2*x + y) ∂/∂t∧∂/∂x + 2 ∂/∂t∧∂/∂y + (x + z) ∂/∂x∧∂/∂z + 3 ∂/∂x∧∂/∂y
sage: s = a - b ; s
2-vector field on the 4-dimensional differentiable manifold M
sage: s.display()
(2*x - y) ∂/∂t∧∂/∂x - 2 ∂/∂t∧∂/∂y + (-x - z) ∂/∂x∧∂/∂z + 3 ∂/∂x∧∂/∂y

An example of 3-vector field in $\mathbb{R}^3$ with Cartesian coordinates:

sage: M = Manifold(3, 'R3', latex_name=r'\mathbb{R}^3', start_index=1)
sage: c_cart.<x,y,z> = M.chart()
sage: a = M.multivector_field(3, name='a')
sage: a[1,2,3] = x^2+y^2+z^2  # the only independent component
sage: a[:]
# all the components are set from the previous line:
[[[0, 0, 0], [0, 0, x^2 + y^2 + z^2], [0, -x^2 - y^2 - z^2, 0]],
[[0, 0, -x^2 - y^2 - z^2], [0, 0, 0], [x^2 + y^2 + z^2, 0, 0]],
[[0, x^2 + y^2 + z^2, 0], [-x^2 - y^2 - z^2, 0, 0], [0, 0, 0]]]
### Spherical components from the tensorial change-of-frame formula:

```
sage: c_spher.<r,th,ph> = M.chart(rr:[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))
```

Check of the inverse coordinate transformation:

```
r == r  *passed*
th == arctan2(r*sin(th), r*cos(th))  **failed**
ph == arctan2(r*sin(ph)*sin(th), r*cos(ph)*sin(th))  **failed**
x == x  *passed*
y == y  *passed*
z == z  *passed*
```

NB: a failed report can reflect a mere lack of simplification.

```
sage: a.comp(c_spher.frame()) # computation of components w.r.t. spherical frame
Fully antisymmetric 3-indices components w.r.t. Coordinate frame (R3, (∂/∂r,∂/∂th,∂/∂ph))
sage: a.comp(c_spher.frame())[1,2,3, c_spher]
1/sin(th)
sage: a.display(c_spher.frame())
a = sqrt(x^2 + y^2 + z^2)/sqrt(x^2 + y^2) \partial/\partial r \wedge \partial/\partial \theta \wedge \partial/\partial \phi
sage: a.display(c_spher.frame(), c_spher)
a = 1/sin(th) \partial/\partial r \wedge \partial/\partial \theta \wedge \partial/\partial \phi
```

As a shortcut of the above command, on can pass just the chart `c_spher` to `display`, the vector frame being then assumed to be the coordinate frame associated with the chart:

```
sage: a.display(c_spher)
a = 1/sin(th) \partial/\partial r \wedge \partial/\partial \theta \wedge \partial/\partial \phi
```

The exterior product of two multivector fields is performed via the method `wedge()`:

```
sage: a = M.vector_field([x*y, -z*x, y], name = 'A')
sage: b = M.vector_field([y, z + y, x^2-z^2], name = 'B')
sage: ab = a.wedge(b) ; ab
2-vector field A\wedge B on the 3-dimensional differentiable manifold R3
sage: ab.display()
A\wedge B = (x*y^2 + 2*x*y*z) \partial/\partial x \wedge \partial/\partial y + (x^3*y - x*y*z^2 - y^2) \partial/\partial x \wedge \partial/\partial z + (x^2 - (x^3 + y)*z) \partial/\partial y \wedge \partial/\partial z
```

Let us check the formula relating the exterior product to the tensor product for vector fields:

```
sage: a.wedge(b) == a*b - b*a
True
```

The tensor product of a vector field and a 2-vector field is not a 3-vector field but a tensor field of type (3,0) with less symmetries:

```
sage: c = a*ab ; c
Tensor field A\otimes(A\wedge B) of type (3,0) on the 3-dimensional differentiable manifold R3
sage: c.symmetries() # the antisymmetry is only w.r.t. the last 2 arguments:
no symmetry; antisymmetry: (1, 2)
```

### 2.12. Alternating Multivector Fields
The Lie derivative of a 2-vector field is a 2-vector field:

```
sage: ab.lie_der(a)
2-vector field on the 3-dimensional differentiable manifold R3
```

**bracket (other)**

Return the Schouten-Nijenhuis bracket of `self` with another multivector field.

The Schouten-Nijenhuis bracket extends the Lie bracket of vector fields (cf. `bracket()`) to multivector fields.

Denoting by $A^p(M)$ the $C^k$-module of $p$-vector fields on the $C^k$-differentiable manifold $M$ over the field $K$ (cf. `MultivectorModule`), the Schouten-Nijenhuis bracket is a $K$-bilinear map

$$A^p(M) \times A^q(M) \rightarrow A^{p+q-1}(M)$$

which obeys the following properties:

- if $p = 0$ and $q = 0$, (i.e. $a$ and $b$ are two scalar fields), $[a, b] = 0$
- if $p = 0$ (i.e. $a$ is a scalar field) and $q \geq 1$, $[a, b] = -\mathcal{L}_a b$ (minus the interior product of the differential of $a$ by $b$)
- if $p = 1$ (i.e. $a$ is a vector field), $[a, b] = \mathcal{L}_a b$ (the Lie derivative of $b$ along $a$)
- $[a, b] = -(-1)^{(p-1)(q-1)}[b, a]$
- for any multivector field $c$ and $(a, b) \in A^p(M) \times A^q(M)$, $[a, .]$ obeys the *graded Leibniz rule*
  $$[a, b \wedge c] = [a, b] \wedge c + (-1)^{(p-1)q}b \wedge [a, c]$$
- for $(a, b, c) \in A^p(M) \times A^q(M) \times A^r(M)$, the *graded Jacobi identity* holds:
  $$(-1)^{(p-1)(r-1)}[a, [b, c]] + (-1)^{(q-1)(p-1)}[b, [c, a]] + (-1)^{(r-1)(q-1)}[c, [a, b]] = 0$$

**Note:** There are two definitions of the Schouten-Nijenhuis bracket in the literature, which differ from each other when $p$ is even by an overall sign. The definition adopted here is that of [Mar1997], [Kos1985] and Wikipedia article Schouten-Nijenhuis_bracket. The other definition, adopted e.g. by [Nij1955], [Lic1977] and [Vai1994], is $[a, b]' = (-1)^{p+1}[a, b]$.

**INPUT:**

- `other` – a multivector field

**OUTPUT:**

- instance of `MultivectorFieldParal` (or of `DiffScalarField` if $p = 1$ and $q = 0$) representing the Schouten-Nijenhuis bracket $[a, b]$, where $a$ is `self` and $b$ is `other`

**EXAMPLES:**

Let us consider two vector fields on a 3-dimensional manifold:

```
sage: M = Manifold(3, 'M')
sage: X.<x,y,z> = M.chart()
sage: a = M.vector_field([x*y+z, x+y-z, z-2*x+y], name='a')
sage: b = M.vector_field([y+2*z-x, x^2-y+z, z-x], name='b')
```

and form their Schouten-Nijenhuis bracket:
sage: s = a.bracket(b); s
Vector field \([a,b]\) on the 3-dimensional differentiable manifold \(M\)
sage: s.display()
\([a,b]\) = \((-x^3 + (x + 3)*y - y^2 - (x + 2*y + 1)*z - 2*x) \partial/\partial x
+ (2*x^2*y - x^2 + 2*x*z - 3*x) \partial/\partial y
+ (-x^2 - (x - 4)*y - 3*x + 2*z) \partial/\partial z

Check that \([a,b]\) is actually the Lie bracket:

sage: f = M.scalar_field({X: x+y*z}, name='f')
sage: s(f) == a(b(f)) - b(a(f))
True

Check that \([a,b]\) coincides with the Lie derivative of \(b\) along \(a\):

sage: s == b.lie_derivative(a)
True

Schouten-Nijenhuis bracket for \(p=0\) and \(q=1\):

sage: s = f.bracket(a); s
Scalar field \(-\text{d} f a\) on the 3-dimensional differentiable manifold \(M\)
sage: s.display()
\(-\text{d} f a\) : \(M \rightarrow \mathbb{R}\)
\((x, y, z) \mapsto x*y - y^2 - (x + 2*y + 1)*z + z^2

Check that \([f,a]\) is actually the Lie bracket:

sage: s == - f.differential()(a)
True

Schouten-Nijenhuis bracket for \(p=0\) and \(q=2\):

sage: c = M.multivector_field(2, name='c')
sage: c[0,1], c[0,2], c[1,2] = x+z+1, x*y+z, x-y
sage: s = f.bracket(c); s
Vector field \(-\text{d} f c\) on the 3-dimensional differentiable manifold \(M\)
sage: s.display()
\(-\text{d} f c\) = \((x*y^2 + (x + y + 1)*z + z^2) \partial/\partial x
+ (x*y - y^2 - x - z - 1) \partial/\partial y
+ (-x*y - (x - y + 1)*z) \partial/\partial z

Check that \([f,c]\) is actually the Lie bracket:

sage: s == - f.differential().interior_product(c)
True

Schouten-Nijenhuis bracket for \(p=1\) and \(q=2\):

sage: s = a.bracket(c); s
2-vector field \([a,c]\) on the 3-dimensional differentiable manifold \(M\)
sage: s.display()
\([a,c]\) = \(((x - 1)*y - (y - 2)*z - 2*x - 1) \partial/\partial x\wedge\partial y
+ ((x + 1)*y - (x + 1)*z - 3*x - 1) \partial/\partial x\wedge\partial z
+ (-5*x + y - z - 2) \partial/\partial y\wedge\partial z

Again, since \(a\) is a vector field, the Schouten-Nijenhuis bracket coincides with the Lie derivative:
Schouten-Nijenhuis bracket for $p = 2$ and $q = 2$:

```sage
sage: c = M.multivector_field(2, name='c')
sage: d = M.multivector_field(2, name='d')
sage: s = c.bracket(d); s.display()
3-vector field $[c,d]$ on the 3-dimensional differentiable manifold $M$

Let us check the component formula (with respect to the manifold's default coordinate chart, i.e. $X$) for $p = 2$, taking into account the tensor antisymmetries:

```sage
sage: s[0,1,2] == - sum(c[i,0]*d[1,2].diff(i) + c[i,1]*d[2,0].diff(i) + c[i,2]*d[0,1].diff(i) + d[i,0]*c[1,2].diff(i) + d[i,1]*c[2,0].diff(i) + d[i,2]*c[0,1].diff(i) for i in M.irange())
True
```

Schouten-Nijenhuis bracket for $p = 1$ and $q = 3$:

```sage
sage: e = M.multivector_field(3, name='e')
sage: e[0,1,2] = x*y*z+1
sage: s = e.bracket(a); s.display()
3-vector field $[a,e]$ on the 3-dimensional differentiable manifold $M$

Again, since $p = 1$, the bracket coincides with the Lie derivative:

```sage
sage: s == e.lie_derivative(a)
True
```

Schouten-Nijenhuis bracket for $p = 2$ and $q = 3$:

```sage
sage: s = c.bracket(e); s.display()
4-vector field $[c,e]$ on the 3-dimensional differentiable manifold $M$

Since on a 3-dimensional manifold, any 4-vector field is zero, we have:

```sage
sage: s.display()
[c,e] = 0
```

Let us check the graded commutation law $[a, b] = -(-1)^{(p-1)(q-1)}[b, a]$ for various values of $p$ and $q$:

```sage
sage: f.bracket(a) == - a.bracket(f)  # p=0 and q=1
True
sage: f.bracket(c) == c.bracket(f)  # p=0 and q=2
True
sage: a.bracket(b) == - b.bracket(a)  # p=1 and q=1
True
sage: a.bracket(c) == - c.bracket(a)  # p=1 and q=2
True
sage: c.bracket(d) == d.bracket(c)  # p=2 and q=2
True
```
Let us check the graded Leibniz rule for $p = 1$ and $q = 1$:

```python
sage: a.bracket(b.wedge(c)) == a.bracket(b).wedge(c) + b.wedge(a.bracket(c))
˓→# long time
True
```

as well as for $p = 2$ and $q = 1$:

```python
sage: c.bracket(a.wedge(b)) == c.bracket(a).wedge(b) - a.wedge(c.bracket(b))
˓→# long time
True
```

Finally let us check the graded Jacobi identity for $p = 1$, $q = 1$ and $r = 2$:

```python
sage: # long time
sage: a_bc = a.bracket(b.bracket(c))
sage: b_ca = b.bracket(c.bracket(a))
sage: c_ab = c.bracket(a.bracket(b))
sage: a_bc + b_ca + c_ab == 0
True
```

as well as for $p = 1$, $q = 2$ and $r = 2$:

```python
sage: # long time
sage: a_cd = a.bracket(c.bracket(d))
sage: c_da = c.bracket(d.bracket(a))
sage: d_ac = d.bracket(a.bracket(c))
sage: a_cd + c_da - d_ac == 0
True
```

**interior_product** *(form)*

Interior product with a differential form.

If `self` is a multivector field $A$ of degree $p$ and $B$ is a differential form of degree $q \geq p$ on the same manifold as $A$, the interior product of $A$ by $B$ is the differential form $\iota_A B$ of degree $q - p$ defined by

$$ (\iota_A B)_{i_1 \ldots i_{q-p}} = A^{k_1 \ldots k_p} B_{k_1 \ldots k_p i_1 \ldots i_{q-p}} $$

**Note:** $A.interior_product(B)$ yields the same result as $A.contract(0, \ldots, p-1, B, 0, \ldots, p-1)$ (cf. `contract()`), but `interior_product` is more efficient, the alternating character of $A$ being not used to reduce the computation in `contract()`

**INPUT:**

- `form` – differential form $B$ (instance of `DiffFormParal`); the degree of $B$ must be at least equal to the degree of `self`

**OUTPUT:**

- scalar field (case $p = q$) or `DiffFormParal` (case $p < q$) representing the interior product $\iota_A B$, where $A$ is `self`

**See also:**

`interior_product()` for the interior product of a differential form with a multivector field

**EXAMPLES:**

Interior product with $p = 1$ and $q = 1$ on 4-dimensional manifold:
sage: M = Manifold(4, 'M')
sage: X.<t,x,y,z> = M.chart()
sage: a = M.vector_field([x, 1+t^2, x*z, y-3], name='a')
sage: b = M.one_form([-z^2, 2, x, x-y], name='b')
sage: s = a.interior_product(b); s
Scalar field i_a b on the 4-dimensional differentiable manifold M
sage: s.display()
i_a b: M → ℝ
(t, x, y, z) ↦ x^2*z - x*z^2 + 2*t^2 + (x + 3)*y - y^2
- 3*x + 2
In this case, we have i_a b = i_a i_b = i_a (b) = b(a):

sage: all([s == a.contract(b), s == a(b), s == b(a)])
True

Case \( p = 1 \) and \( q = 3 \):

sage: c = M.diff_form(3, name='c')
sage: c[0,1,2], c[0,1,3] = x*y - z, -3*t
sage: c[0,2,3], c[1,2,3] = t+x, y
sage: s = a.interior_product(c); s
2-form i_a c on the 4-dimensional differentiable manifold M
sage: s.display()
i_a c = (x^2*y*z - x*z^2 - 3*t*y + 9*t) dt∧dx
+ (-t^2*x - t)*y + (t^2 + 1)*z - 3*t - 3*x) dt∧dy
+ (3*t^3 - (t*x + x^2)*z + 3*t) dt∧dz
+ ((x^2 - 3)*y + y^2 - x*z) dx∧dy
+ (-x*y*z - 3*t*x) dx∧dz + (t*x + x^2 + (t^2 + 1)*y) dy∧dz
sage: s == a.contract(c)
True

Case \( p = 2 \) and \( q = 3 \):

sage: d = M.multivector_field(2, name='d')
sage: d[0,1], d[0,2], d[0,3] = t-x, 2*z, y-1
sage: d[1,2], d[1,3], d[2,3] = z, y+t, 4
sage: s = d.interior_product(c); s
1-form i_d c on the 4-dimensional differentiable manifold M
sage: s.display()
i_d c = (2*x*y*z - 6*t^2 - 6*t*y - 2*z^2 + 8*t + 8*x) dt
+ (-4*x*y*z + 2*(3*t + 4)*y + 4*z^2 - 6*t) dx
+ (2*((t - 1)*x - x^2 - 2*t)*y - 2*y^2 - 2*(t - x)*z + 2*t
+ 2*x) dy + (-6*t^2 + 6*t*x + 2*(2*t + 2*x + y)*z) dz
sage: s == d.contract(0, 1, c, 0, 1)
True

wedge (other)

Exterior product of self with another multivector field.

INPUT:

- other – another multivector field

OUTPUT:

- instance of MultivectorFieldParal representing the exterior product self ∧ other

EXAMPLES:
Exterior product of a vector field and a 2-vector field on a 3-dimensional manifold:

```sage
sage: M = Manifold(3, 'M', start_index=1)
sage: X.<x,y,z> = M.chart()
sage: a = M.vector_field([2, 1+x, y*z], name='a')
sage: b = M.multivector_field(2, name='b')
sage: b[1,2], b[1,3], b[2,3] = y^2, z+x, z^2
sage: a.display()
a = 2 \partial/\partial x + (x + 1) \partial/\partial y + y*z \partial/\partial z
sage: b.display()
b = y^2 \partial/\partial x \wedge \partial/\partial y + (x + z) \partial/\partial x \wedge \partial/\partial z + z^2 \partial/\partial y \wedge \partial/\partial z
sage: s = a.wedge(b); s
3-vector field a\wedge b on the 3-dimensional differentiable manifold M
sage: s.display()
a\wedge b = (-x^2 + (y^3 - x - 1)*z + 2*z^2 - x) \partial/\partial x \wedge \partial/\partial y \wedge \partial/\partial z
```

Check:

```sage
sage: s[1,2,3] == a[1]*b[2,3] + a[2]*b[3,1] + a[3]*b[1,2]
True
```

Exterior product with a scalar field:

```sage
sage: f = M.scalar_field(x, name='f')
sage: s = b.wedge(f); s
2-vector field f*b on the 3-dimensional differentiable manifold M
sage: s.display()
f*b = x*y^2 \partial/\partial x \wedge \partial/\partial y + (x^2 + x*z) \partial/\partial x \wedge \partial/\partial z + x*z^2 \partial/\partial y \wedge \partial/\partial z
sage: s == f*b
True
sage: s == f.wedge(b)
True
```

### 2.13 Affine Connections

The class `AffineConnection` implements affine connections on smooth manifolds.

**AUTHORS:**
- Marco Mancini (2015) : parallelization of some computations
- Florentin Jaffredo (2018) : series expansion with respect to a given parameter

**REFERENCES:**
- [Lee1997]
- [KN1963]
- [ONe1983]

```python
class sage.manifolds.differentiable.affine_connection.AffineConnection(domain, name, latex_name=None)
```

**Bases:** `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 $K = \mathbb{R}$ or $K = \mathbb{C}$), let $C^\infty(M)$ be the algebra of smooth functions $M \to K$ (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

\[
\nabla : \mathcal{X}(M) \times \mathcal{X}(M) \longrightarrow \mathcal{X}(M)
\]

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: $\forall f \in C^\infty(M)$, $\nabla f u v = f \nabla u v$
- obeys Leibniz rule w.r.t. the second argument: $\forall f \in C^\infty(M)$, $\nabla(u f v) = d f( u ) v + f \nabla u v$

The affine connection $\nabla$ gives birth to the covariant derivative operator acting on tensor fields, denoted by the same symbol:

\[
\nabla : T^{(k,l)}(M) \longrightarrow T^{(k,l+1)}(M)
\]

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

\[
\forall u \in \mathcal{X}(M), \nabla_u t = \nabla t(\cdot \cdot \cdot , u)
\]

**Note:** The above convention means that, in terms of index notation, the “derivation index” in $\nabla t$ is the last one:

\[
\nabla_{\partial^{a_1 \ldots a_k}_{b_1 \ldots b_l}} = (\nabla t)^{a_1 \ldots a_k}_{b_1 \ldots b_l}
\]

**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:

```python
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:
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:

\[
\text{sage}: \text{nab}[1,1,2], \text{nab}[3,2,3] = x^2, yz \\
\text{sage}: \text{nab}._\text{coefficients}{\text{(Coordinate frame \(M, (\partial/\partial x, \partial/\partial y, \partial/\partial z)\)): 3-indices components w.r.t. Coordinate frame \(M, (\partial/\partial x, \partial/\partial y, \partial/\partial z)\))}
\]

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:

\[
\text{sage}: \text{nab}[c\_\text{xyz}.\text{frame()}, 1,1,2], \text{nab}[c\_\text{xyz}.\text{frame()},3,2,3] = x^2, yz \\
\text{sage}: \text{nab}._\text{coefficients}{\text{(Coordinate frame \(M, (\partial/\partial x, \partial/\partial y, \partial/\partial z)\)): 3-indices components w.r.t. Coordinate frame \(M, (\partial/\partial x, \partial/\partial y, \partial/\partial z)\))}
\]

Unset components are initialized to zero:

\[
\text{sage}: \text{nab}[:]{\text{list of coefficients relative to the manifold’s default vector frame}} \\
\rightarrow \text{frame} \\
\text{[[[0, x^2, 0], [0, 0, 0], [0, 0, 0]],} \\
\text{[[0, 0, 0], [0, 0, 0], [0, 0, 0]},} \\
\text{[[0, 0, 0], [0, 0, y*z], [0, 0, 0]]}
\]

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 ChartFunction, while the double square brackets return a scalar field:

\[
\text{sage}: \text{nab}[1,1,2] \\
x^2 \\
\text{sage}: \text{nab}[1,1,2].\text{display}() \\
(x, y, z) \mapsto x^2 \\
\text{sage}: \text{type}(\text{nab}[1,1,2]) <\text{class 'sage.manifolds.chart_func.ChartFunctionRing_with_category.element_class'>} \\
\text{sage}: \text{nab}[1,1,2] \text{ Scalar field on the 3-dimensional differentiable manifold M} \\
\text{sage}: \text{nab}[1,1,2].\text{display}() \\
M \rightarrow \mathbb{R} \\
(x, y, z) \mapsto x^2 \\
\text{sage}: \text{nab}[1,1,2].\text{coord_function()} \text{ is nab}[1,1,2] \text{ True}
\]

Action on a scalar field:

\[
\text{sage}: f = \text{M}.\text{scalar_field}(x^2 - y^2, \text{name}='f') \\
\text{sage}: \text{Df} = \text{nab}(f) ; \text{Df} \text{ 1-form df on the 3-dimensional differentiable manifold M} \\
\text{sage}: \text{Df[]} \text{[2*x, -2*y, 0]} \\
\text{sage}: \text{Df}==f.\text{differential}() \text{ True}
\]

The action of an affine connection on a scalar field must coincide with the differential:
A generic affine connection has some torsion:

\[
\nabla_j \nabla_i f - \nabla_i \nabla_j f = T^k_{ij} \nabla_k f,
\]

where the \( T^k_{ij} \)'s are the components of the connection's torsion tensor:

\[
\]
We may let it act on a vector field defined globally on $M$:

```python
sage: a = M.vector_field({eU: [-y,x]}, name='a')
sage: a.add_comp_by_continuation(eV, W, c_uv)
sage: a.display(eU)
a = -y \frac{\partial}{\partial x} + x \frac{\partial}{\partial y}
sage: a.display(eV)
a = v \frac{\partial}{\partial u} - u \frac{\partial}{\partial v}
```

A few tests:

```python
sage: nab(a.restrict(V)) == da.restrict(V) True
sage: nab(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) # long time True
```

Same examples with SymPy as the engine for symbolic calculus:

```python
sage: M.set_calculus_method('sympy')
sage: nab = M.affine_connection('nabla', r'nabla')
sage: nab[0,0,0], nab[ 1,0,1] = x, 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()
```

At this stage, the connection is fully defined on all the manifold:
We may let it act on a vector field defined globally on $M$:

```python
sage: a = M.vector_field({eU: [-y,x]}, name='a')
sage: a.add_comp_by_continuation(eV, W, c_uv)
sage: a.display(eU)
a = -y \frac{\partial}{\partial x} + x \frac{\partial}{\partial y}
sage: a.display(eV)
a = v \frac{\partial}{\partial u} - u \frac{\partial}{\partial v}
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 \frac{\partial}{\partial x} \otimes dx - \frac{\partial}{\partial x} \otimes dy + \frac{\partial}{\partial y} \otimes dx - x*y**2 \frac{\partial}{\partial y} \otimes dy
sage: da.display(eV)
nabla(a) = (-u**3/16 + u**2*v/16 - u**2/8 + u*v**2/16 - v**3/16 + v**2/8) \frac{\partial}{\partial u} \otimes du + (u**3/16 - u**2*v/16 - u**2/8 - u*v**2/16 + v**3/16 + v**2/8 + 1) \frac{\partial}{\partial v} \otimes dv + (u**3/16 - u**2*v/16 - u**2/8 - u*v**2/16 + v**3/16 + v**2/8 - 1) \frac{\partial}{\partial v} \otimes du + (-u**3/16 + u**2*v/16 - u**2/8 + u*v**2/16 - v**3/16 + v**2/8) \frac{\partial}{\partial u} \otimes dv
```

To make affine connections hashable, they have to be set immutable before:

```python
sage: nab.is_immutable()
False
sage: nab.set_immutable()

```

Immutable connections cannot be changed anymore:

```python
sage: nab.set_coef(eU)
Traceback (most recent call last):
  ... ValueError: the coefficients of an immutable element cannot be changed
```

However, they can now be used as keys for dictionaries:

```python
sage: {nab: 1}[nab]
1
```

The immutability process cannot be made undone. If a connection is needed to be changed again, a copy has to be created:

```python
sage: nab_copy = nab.copy('nablo'); nab_copy
Affine connection nablo on the 2-dimensional differentiable manifold M
sage: nab_copy is nab
False
sage: nab_copy == nab
True
sage: nab_copy.is_immutable()
False
```

`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.

770 Chapter 2. Differentiable Manifolds
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:

```python
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, (∂/∂x,∂/∂y))
sage: nab.add_coef(eX)
3-indices components w.r.t. Coordinate frame (M, (∂/∂x,∂/∂y))
sage: nab.add_coef(eX)[1,2,1] = x*y
sage: nab.display(eX)
\Gamma^{x\xi}_{\eta\xi} = x*y
```

Since `eX` is the manifold’s default vector frame, its mention may be omitted:

```python
sage: nab.add_coef()[1,2,1] = x*y
sage: nab.add_coef()
3-indices components w.r.t. Coordinate frame (M, (∂/∂x,∂/∂y))
sage: nab.add_coef()[1,2,1] = x*y
sage: nab.display()
\Gamma^{x\xi}_{\eta\xi} = x*y
```

Adding connection coefficients w.r.t. to another vector frame:

```python
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)
\Gamma^{2\,\xi\,1}_{1\,\xi\,1} = x + y
\Gamma^{2\,\xi\,2}_{1\,\xi\,2} = x - y
```

The coefficients w.r.t. the frame `e` have been kept:

```python
sage: nab.display(eX)
\Gamma^{x\xi}_{\eta\xi} = x*y
```

To delete them, use the method `set_coef()` instead.

```python
ccoef(frame=\text{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 $\Gamma^k_{ij}$ defined by

$$\nabla_{e_j} e_i = \Gamma^k_{ij} e_k$$

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:

```python
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, (∂/∂x,∂/∂y,∂/∂z))
sage: type(nab.coef())
<class 'sage.tensor.modules.comp.Components'>
sage: M.default_frame()
Coordinate frame (M, (∂/∂x,∂/∂y,∂/∂z))
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 $\text{DiffForm}$

EXEMPLARY:

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 $\text{set_coef()}$ or $\text{add_coef()}$):

```
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 $\omega^i_j = \Gamma^i_{jk}e^k$:

First on the manifold's default frame $(\partial/\partial x, \partial/\partial y, d/\partial z)$:

```
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) == \n    ....:             sum( nab[[i,j,k]]*dx[k] for k in M.irange() ) )
sage: check
[True, True, True, True, True, True, True, True, True]
```

Then 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():
    ....:     check.append( nab.connection_form(i,j) == \n    ....:         sum( nab[[i,j,k]]*ef[k] for k in M.irange() ) )
```

(continues on next page)
Check of the formula $\nabla_v e_j = \langle \omega^i_{\ j}, v \rangle e_i$:

```python
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  # long time
    nab(b[j]).contract(v) == \n    sum( nab.connection_form(i,j)(v)*b[i] for i in M.irange())
```

```python
sage: for j in M.irange():
    # check on frame e  # long time
    nab(e[j]).contract(v) == \n    sum( nab.connection_form(i,j,e)(v)*e[i] for i in M.irange())
```

**copy** *(name, latex_name=None)*

Return an exact copy of *self*.

**INPUT:**

- *name* – name given to the copy
- *latex_name* – (default: None) LaTeX symbol to denote the copy; if none is provided, the LaTeX symbol is set to *name*

**Note:** The name and the derived quantities are not copied.

**EXAMPLES:**

```python
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: nab.set_coef(eX)[1,2,2] = x + y
sage: nab.display()
Gam^x_yx = x*y
Gam^x_yy = x + y
sage: nab_copy = nab.copy(name='nabla_1', latex_name=r'\nabla_1')
sage: nab == nab_copy
True
sage: nab_copy.display()
Gam^x_yx = x*y
Gam^x_yy = x + y
```
\textbf{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. \textit{riemann()}), \((e^i)\) is the coframe dual to \((e_i)\) and \((u, v)\) is a generic pair of vectors.

\textbf{INPUT:}

- \(i, j\) – indices identifying the 2-form \(\Omega^i_j\)
- \textbf{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.

\textbf{OUTPUT:}

- the 2-form \(\Omega^i_j\), as an instance of \textit{DiffForm}

\textbf{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^2-z^2
sage: nab.curvature_form(1,1)  # long time
2-form curvature (1,1) of connection nabla w.r.t. Coordinate frame (M, (∂/∂x,∂/∂y,∂/∂z)) 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, (∂/∂x,∂/∂y,∂/∂z)) = (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: ef = e.coframe()
sage: ef[1].display(), ef[2].display(), ef[3].display()
(e_1 = 1/y dx, e_2 = z dy, e_3 = x dz)
sage: ef1.display()  # 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: ef1.display()  # 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
```

2.13. Affine Connections 775
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. \texttt{connection_form()}). Let us check it on the frame \( e \):

```python
sage: omega = nab.connection_form
sage: check = []
for i in M.irange():
    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 \( \text{frame} = \text{None} \)

Delete all the coefficients but those corresponding to \( \text{frame} \).

INPUT:

- \textbf{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:

```python
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
e = M.vector_frame('e')
sage: nab.add_coef(e)[2,1,1] = x+y
sage: nab.display(eX)
Gam^x_yx = x*y
e: Gam^2_11 = x + y
```

Let us delete the connection coefficients w.r.t. all frames except for frame \( eX \):

```python
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:

```python
sage: nab.display(e)
Traceback (most recent call last):
...
ValueError: no common frame found for the computation
```

display \( \text{frame} = \text{None, chart} = \text{None, symbol} = \text{None, latex_symbol} = \text{None, index_labels} = \text{None, index_latex_labels} = \text{None, coordinate_labels} = \text{True, only_nonzero} = \text{True, only_nonredundant} = \text{False} \)

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 domain of frame 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 }^{\, x \phantom{\, x} \, \rightarrow \phantom{\, y} } & = & x^2 \\
\Gamma_{\phantom{\, z} \, y \, z }^{\, z \phantom{\, y} \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:
\begin{verbatim}
M = Manifold(3, 'R^3', start_index=1)
c_spher.<r,th,ph> = M.chart(rr:(0,+oo) th:(0,pi): \theta ph:(0,2*pi): \phi)
g = M.metric(g)
g[1,1], g[2,2], g[3,3] = 1, r^2, (r*sin(th))^2
g.display()
g.connection().display(only_nonredundant=True)
\end{verbatim}

By default, the parameter only_nonredundant is set to False:

\begin{verbatim}
g.connection().display()
\end{verbatim}

\begin{verbatim}
M.default_frame() is c_xyz.frame()
True
sage: nab.display(coordinate_labels=False)
\begin{array}{lcl}
\Gamma_{12}^1 &=& x^2 \\
\Gamma_{23}^3 &=& y*z
\end{array}
\end{verbatim}

The index labels can also be customized:

\begin{verbatim}
sage: nab.display(index_labels=['(1)', '(2)', '(3)'])
\begin{array}{lcl}
\Gamma_{(1)}^{(1)} &=& x^2 \\
\Gamma_{(2)}^{(3)} &=& y*z
\end{array}
\end{verbatim}

The symbol ‘\(\Gamma\)’ can be changed:

\begin{verbatim}
sage: nab.display(symbol='C', latex_symbol='C')
C_{x\phantom{y}y}^x = x^2 \\
C_{y\phantom{x}z}^z = y*z
\end{verbatim}

Display of Christoffel symbols, skipping the redundancy associated with the symmetry of the last two indices:

\begin{verbatim}
sage: M = Manifold(3, 'R^3', start_index=1)
c_spher.<r,th,ph> = M.chart(rr:(0,+oo) th:(0,pi): \theta ph:(0,2*pi): \phi)
g = M.metric(g)
g[1,1], g[2,2], g[3,3] = 1, r^2, (r*sin(th))^2
g.display()
g.connection().display(only_nonredundant=True)
\end{verbatim}

\begin{verbatim}
\Gamma_{r\theta \theta} = -r \\
\Gamma_{r\phi \phi} = -r*sin(th)^2 \\
\Gamma_{\theta r \theta} = 1/r \\
\Gamma_{\theta \theta \phi} = -cos(th)*sin(th) \\
\Gamma_{\phi r \phi} = 1/r \\
\Gamma_{\phi \phi \theta} = cos(th)/sin(th)
\end{verbatim}

By default, the parameter only_nonredundant is set to False:

\begin{verbatim}
\Gamma_{r\theta \theta} = -r \\
\Gamma_{r\phi \phi} = -r*sin(th)^2 \\
\Gamma_{\theta r \theta} = 1/r \\
\Gamma_{\theta \theta \phi} = -cos(th)*sin(th) \\
\Gamma_{\phi r \phi} = 1/r \\
\Gamma_{\phi \phi \theta} = cos(th)/sin(th)
\end{verbatim}

\begin{verbatim}
domain()
\end{verbatim}

Return the manifold subset on which the affine connection is defined.

\begin{verbatim}
OUTPUT:
\end{verbatim}
• instance of class `DifferentiableManifold` representing the manifold on which `self` is defined.

**EXAMPLES:**

```python
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
```

**isImmutable()**

Return `True` if this object is immutable, i.e. its coefficients cannot be changed, and `False` if it is not.

To set an affine connection immutable, use `set_immutable()`.

**EXAMPLES:**

```python
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: nab.is_immutable()
False
sage: nab.set_immutable()

sage: nab.is_immutable()
True
```

**isMutable()**

Return `True` if this object is mutable, i.e. its coefficients can be changed, and `False` if it is not.

**EXAMPLES:**

```python
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: nab.is_mutable()
True
sage: nab.set_immutable()

sage: nab.is_mutable()
False
```

**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  \# Gamma^1_{12} = x^2, Gamma^3_{23} = \ldots
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} = \ldots
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]
```

ricci()
The result is cached (until the connection is modified via `set_coef()` or `add_coef()`):

```plaintext
sage: nab.riemann() is r
True
```

The `riemann()` function returns the connection’s Riemann curvature tensor.

**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:

```plaintext
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} = y*z
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):

```plaintext
sage: r.symmetries()
no symmetry; antisymmetry: (2, 3)
```

The components:

```plaintext
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]],
]
```

The result is cached (until the connection is modified via `set_coef()` or `add_coef()`):

```plaintext
sage: nab.riemann() is r
True
```
Another example: Riemann curvature tensor of some connection on a non-parallelizable 2-dimensional manifold:

```python
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'
abla')
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 # long time
Tensor field of type (1,3) on the 2-dimensional differentiable manifold M
sage: r.parent() # long time
Module T^(1,3)(M) of type-(1,3) tensors fields on the 2-dimensional differentiable manifold M
```

The same computation parallelized on 2 cores:

```python
sage: Parallelism().set(nproc=2)
sage: r_backup = r # long time
sage: nab = M.affine_connection('nabla', r'
abla')
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()
```

(continues on next page)
Manifolds, Release 10.3

(continued from previous page)

```
sage: r = nab.riemann() ; r  # long time
Tensor field of type (1,3) on the 2-dimensional differentiable
  manifold M
sage: r.parent()  # long time
Module T^(1,3)(M) of type-(1,3) tensors fields on the 2-dimensional
differentiable manifold M
sage: r == r_backup  # long time
True
sage: Parallelism().set(nproc=1)  # switch off parallelization
```

**set_calc_order** (*symbol*, *order*, *truncate=False*)

Trigger a series expansion with respect to a small parameter in computations involving *self*.

This property is propagated by usual operations. The internal representation must be SR for this to take effect.

**INPUT:**

- *symbol* – symbolic variable (the "small parameter" \( \epsilon \)) with respect to which the connection coefficients are expanded in power series

- *order* – integer; the order \( n \) of the expansion, defined as the degree of the polynomial representing the truncated power series in *symbol*

- *truncate* – (default: False) determines whether the connection coefficients are replaced by their expansions to the given order

**EXAMPLES:**

```
sage: M = Manifold(4, 'M', structure='Lorentzian')
sage: C.<t,x,y,z> = M.chart()
sage: e = var('e')
sage: g = M.metric()
sage: h = M.tensor_field(0, 2, sym=(0,1))
sage: g[0, 0], g[1, 1], g[2, 2], g[3, 3] = -1, 1, 1, 1
sage: h[0, 1] = x
sage: g.set(g + e*h)
sage: g[:]
[ -1*e*x  0  0]
[ e*x    1  0  0]
[  0    1  0]
[  0    0  1]
sage: nab = g.connection()
sage: nab[0, 1, 1]
-e/(e^2*x^2 + 1)
sage: nab.set_calc_order(e, 1, truncate=True)
sage: nab[0, 1, 1]
-e
```

**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.

2.13. Affine Connections 783
• 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:

```python
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, (∂/∂x,∂/∂y))
sage: nab.set_coef(eX)
3-indices components w.r.t. Coordinate frame (M, (∂/∂x,∂/∂y))
sage: nab.set_coef(eX)[1,2,1] = x*y
sage: nab.display(eX)
\Gamma^x_{yx} = x*y
```

Since `eX` is the manifold’s default vector frame, its mention may be omitted:

```python
sage: nab.set_coef()[1,2,1] = x*y
sage: nab.set_coef()
3-indices components w.r.t. Coordinate frame (M, (∂/∂x,∂/∂y))
sage: nab.set_coef()[1,2,1] = x*y
sage: nab.display()
\Gamma^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:

```python
sage: nab[1,2,1] = x*y
sage: nab.display()
\Gamma^x_{yx} = x*y
```

Setting the connection coefficients w.r.t. another vector frame:

```python
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)
\Gamma^2_{11} = x + y
\Gamma^2_{12} = x - y
```

The coefficients w.r.t. the frame `eX` have been deleted:

```python
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.

```python
set_immutable()
Set self and all restrictions of self immutable.
```
EXAMPLES:

An affine connection can be set immutable:

```
sage: M = Manifold(2, 'M', start_index=1)
sage: X.<x,y> = M.chart()
sage: U = M.open_subset('U', coord_def={X: x^2+y^2<1})
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: nab.is_immutable()  # False
sage: nab.set_immutable()

sage: nab.is_immutable()  # True
```

The coefficients of immutable elements cannot be changed:

```
sage: nab.add_coef(eX)[2,1,1] = x+y
Traceback (most recent call last):
...  # ValueError: the coefficients of an immutable element cannot be changed
```

The restriction are set immutable as well:

```
sage: nabU = nab.restrict(U)
sage: nabU.is_immutable()  # True
```

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} = y*z
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()
(\partial/\partial y \otimes dx \otimes dz + (x + 1) \partial/\partial y \otimes dz \otimes 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: 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,:]
```

(continues on next page)
\[
\begin{bmatrix}
0, x - y, [-x + y, 0], [0, -x*y], [x*y, 0]\n\end{bmatrix}
\]

\[
\begin{bmatrix}
[0, 1/8*u^2 - 1/8*v^2 - 1/2*v], [-1/8*u^2 + 1/8*v^2 + 1/2*v, 0],
[0, -1/8*u^2 + 1/8*v^2 - 1/2*v], [1/8*u^2 - 1/8*v^2 + 1/2*v, 0]\
\end{bmatrix}
\]

Check of the torsion formula:

\[
\begin{align*}
sage: f & = M.scalar_field({c_xy: (x + y)^2, c_uv: u^2}, name='f') \\
sage: DDF & = nab nab(f) ; DDF \\
\text{Tensor field } \nabla(f) \text{ of type (0,2) on the 2-dimensional} \\
\text{differentiable manifold } M \\
sage: DDF.antisymmetrize().display(eU) \\
& = (-x^2*y - (x + 1)*y^2 + x^2) \, dx \wedge dy \\
sage: DDF.antisymmetrize().display(eV) \\
& = (1/8*u^3 - 1/8*u*v^2 - 1/2*u*v) \, du \wedge dv \\
sage: 2*DDF.antisymmetrize() & == \nabla(f).contract(nab torsion()) \\
& \text{True}
\end{align*}
\]

\texttt{torsion\_form} \(i, \text{frame}=\text{None}\)

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. \texttt{torsion()}), \((e^i)\) is the coframe dual to \((e_i)\) and \((u, v)\) is a generic pair of vectors.

INPUT:

\begin{itemize}
\item \(i\) – index identifying the 2-form \(\theta^i\)
\item \texttt{frame} – (default: \text{None}) vector frame relative to which the torsion 2-forms are defined; if \text{None}, the default frame of the connection’s domain is assumed.
\end{itemize}

OUTPUT:

\begin{itemize}
\item the 2-form \(\theta^i\), as an instance of \texttt{DiffForm}
\end{itemize}

EXAMPLES:

Torsion 2-forms on a 3-dimensional manifold:

\[
\begin{align*}
sage: M & = \text{Manifold}(3, 'M', \text{start\_index=1}) \\
sage: c_xyz.<x,y,z> & = M.chart() \\
sage: nab & = M.affine\_connection('\text{nabla}', r'\nabla') \\
sage: nab[1,1,1], nab[1,1,2], nab[1,1,3] & = x^y*z, x^2, -y^z \\
sage: nab[1,2,3], nab[1,3,1], nab[1,3,2] & = -x^3, y^2*z, y^2-x^2 \\
sage: nab[2,1,1], nab[2,1,2], nab[2,2,1] & = z^2, x*y^2*z, -x^2 \\
sage: nab[2,3,1], nab[2,3,2], nab[3,1,2] & = x^2+y^2+z^2, y^2-z^2, x*y+z^2 \\
sage: nab[3,2,1], nab[3,2,2], nab[3,3,3] & = x^y+z, z^3 -y^2, x^2 - z*y^2 \\
sage: nab.torsion\_form(1) \\
\text{2-form torsion (1) of connection } \text{nabla} \text{ w.r.t. Coordinate frame} \\
\text{(M, (\partial/\partial x, \partial/\partial y, \partial/\partial z)) on the 3-dimensional differentiable manifold } M \\
sage: nab.torsion\_form(1)[i::] \\
\begin{bmatrix}
0, -x^2, (y^2 + y)*z) \\
x^2, 0, x^3 - x^2 + y^2) \\
-(y^2 + y)*z, -x^3 + x^2 - y^2, 0
\end{bmatrix}
\end{align*}
\]

Torsion 2-forms w.r.t. a non-holonomic frame:
Cartan’s first structure equation is

$$\theta^i = d\epsilon^i + \omega^i_j \wedge \epsilon^j$$

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
```

2.14 Submanifolds of differentiable manifolds

Given two differentiable manifolds $N$ and $M$, an immersion $\phi$ is a differentiable map $N \to M$ whose differential is everywhere injective. One then says that $N$ is an immersed submanifold of $M$, via $\phi$.

If in addition, $\phi$ is a differentiable embedding (i.e. $\phi$ is an immersion that is a homeomorphism onto its image), then $N$ is called an embedded submanifold of $M$ (or simply a submanifold).

$\phi$ can also depend on one or multiple parameters. As long as the differential of $\phi$ remains injective in these parameters, it represents a foliation. The dimension of the foliation is defined as the number of parameters.

AUTHORS:

- Florentin Jaffredo (2018): initial version
- Matthias Koeppe (2021): open subsets of submanifolds

REFERENCES:

- J. M. Lee: *Introduction to Smooth Manifolds* [Lee2013]
Manifolds, Release 10.3

class sage.manifolds.differentiable.differentiable_submanifold.DifferentiableSubmanifold(n, name, field, structure, ambient=None, base_manifold=None, diff_degree=+Infinity, latex_name=None, start_index=0, category=None, unique_tag=None)

Bases: DifferentiableManifold, TopologicalSubmanifold

Submanifold of a differentiable manifold.

Given two differentiable manifolds \( N \) and \( M \), an immersion \( \phi \) is a differentiable map \( N \to M \) whose differential is everywhere injective. One then says that \( N \) is an immersed submanifold of \( M \), via \( \phi \).

If in addition, \( \phi \) is a differentiable embedding (i.e. \( \phi \) is an immersion that is a homeomorphism onto its image), then \( N \) is called an embedded submanifold of \( M \) (or simply a submanifold).

\( \phi \) can also depend on one or multiple parameters. As long as the differential of \( \phi \) remains injective in these parameters, it represents a foliation. The dimension of the foliation is defined as the number of parameters.

INPUT:

- \( n \) – positive integer; dimension of the submanifold
- name – string; name (symbol) given to the submanifold
- field – field \( K \) on which the sub manifold is defined; allowed values are
  - 'real' or an object of type RealField (e.g., RR) for a manifold over \( \mathbb{R} \)
  - 'complex' or an object of type ComplexField (e.g., CC) for a manifold over \( \mathbb{C} \)
- structure – manifold structure (see TopologicalStructure or RealTopologicalStructure)
- ambient – (default: None) codomain \( M \) of the immersion \( \phi \); must be a differentiable manifold. If None, it is set to self
- base_manifold – (default: None) if not None, must be a differentiable manifold; the created object is then an open subset of base_manifold

2.14. Submanifolds of differentiable manifolds
• **diff_degree** – (default: infinity) degree of differentiability

• **latex_name** – (default: None) string; \LaTeX{} symbol to denote the submanifold; 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 submanifold, 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 DifferentiableManifold would return the previously constructed object corresponding to these arguments)

**EXAMPLES:**

Let \( N \) be a 2-dimensional submanifold of a 3-dimensional manifold \( M \):

\[
\text{sage: } M = \text{Manifold}(3, 'M') \\
\text{sage: } N = \text{Manifold}(2, 'N', ambient=M) \\
\text{sage: } N
\]

2-dimensional differentiable submanifold \( N \) immersed in the 3-dimensional differentiable manifold \( M \)

\[
\text{sage: } CM.\langle x, y, z \rangle = M.\text{chart}() \\
\text{sage: } CN.\langle u, v \rangle = N.\text{chart}()
\]

Let us define a 1-dimensional foliation indexed by \( t \):

\[
\text{sage: } t = \text{var('t')} \\
\text{sage: } \phi = N.\text{continuous_map}(M, \{(CN,CM): [u, v, t+u^2+v^2]\}) \\
\text{sage: } \phi.\text{display}() \\
N \rightarrow M \\
(u, v) \mapsto (x, y, z) = (u, v, u^2 + v^2 + t)
\]

The foliation inverse maps are needed for computing the adapted chart on the ambient manifold:

\[
\text{sage: } \phi_inv = M.\text{continuous_map}(N, \{(CM,CN): [x, y]\}) \\
\text{sage: } \phi_inv.\text{display}() \\
M \rightarrow N \\
(x, y, z) \mapsto (u, v) = (x, y) \\
\text{sage: } \phi_inv_t = M.\text{scalar_field}(\{CM: z-x^2-y^2}\}) \\
\text{sage: } \phi_inv_t.\text{display}() \\
M \rightarrow \mathbb{R} \\
(x, y, z) \mapsto -x^2 - y^2 + z
\]

\( \phi \) can then be declared as an embedding \( N \rightarrow M \):

\[
\text{sage: } N.\text{set_embedding}(\phi, \text{inverse}=\phi_inv, \text{var}=t, \\
\text{....: } \text{t_inverse}={t: \phi_inv_t})
\]

The foliation can also be used to find new charts on the ambient manifold that are adapted to the foliation, i.e., in which the expression of the immersion is trivial. At the same time, the appropriate coordinate changes are computed:

\[
\text{sage: } N.\text{adapted_chart}() \\
[\text{Chart } (M, (u_M, v_M, t_M))] \\
\text{sage: } M.\text{atlas}()
\]

(continues on next page)
Manifolds, Release 10.3

[Chart (M, (x, y, z)), Chart (M, (u_M, v_M, t_M))]

```sage
len(M.coord_changes())
```

2

See also:

`manifold` and `topological_submanifold`

`open_subset` (name, latex_name=None, coord_def={}, supersets=None)

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.

As `self` is a submanifold of its ambient manifold, the new open subset is also considered a submanifold of that. Hence the returned object is an instance of `DifferentiableSubmanifold`.

**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.
- supersets – (default: only self) list of sets that the new open subset is a subset of

**OUTPUT:**

- the open subset, as an instance of `DifferentiableSubmanifold`

**EXAMPLES:**

```sage
M = Manifold(3, 'M', structure="differentiable")
sage: N = Manifold(2, 'N', ambient=M, structure="differentiable")
2-dimensional differentiable submanifold N immersed in the 3-dimensional differentiable manifold M
sage: S = N.subset('S'); S
Subset S of the 2-dimensional differentiable submanifold N immersed in the 3-dimensional differentiable manifold M
sage: O = N.subset('O', is_open=True); O  # indirect doctest
Open subset O of the 2-dimensional differentiable submanifold N immersed in the 3-dimensional differentiable manifold M
sage: phi = N.diff_map(M)
sage: N.set_embedding(phi)
sage: N
2-dimensional differentiable submanifold N embedded in the 3-dimensional differentiable manifold M
sage: S = N.subset('S'); S
Subset S of the 2-dimensional differentiable submanifold N embedded in the 3-dimensional differentiable manifold M
sage: O = N.subset('O', is_open=True); O  # indirect doctest
Open subset O of the
```

(continues on next page)
2-dimensional differentiable submanifold \( N \) embedded in the 3-dimensional differentiable manifold \( M \)

## 2.15 Differentiable Vector Bundles

### 2.15.1 Differentiable Vector Bundles

Let \( K \) be a topological field. A \( C^k \)-differentiable vector bundle of rank \( n \) over the field \( K \) and over a \( C^k \)-differentiable manifold \( M \) (base space) is a \( C^k \)-differentiable manifold \( E \) (total space) together with a \( C^k \) differentiable and surjective map \( \pi : E \rightarrow M \) such that for every point \( x \in M \):

- the set \( E_x = \pi^{-1}(x) \) has the vector space structure of \( K^n \),
- there is a neighborhood \( U \subset M \) of \( x \) and a \( C^k \)-diffeomorphism \( \varphi : \pi^{-1}(x) \rightarrow U \times K^n \) such that \( v \mapsto \varphi^{-1}(y, v) \) is a linear isomorphism for any \( y \in U \).

An important case of a differentiable vector bundle over a differentiable manifold is the tensor bundle (see Tensor-Bundle)

**AUTHORS:**

- Michael Jung (2019): initial version

### class sage.manifolds.differentiable.vector_bundle.DifferentiableVectorBundle(rank, name, base_space, field='real', latex_name=None, category=None, unique_tag=None)

**Bases:** TopologicalVectorBundle

An instance of this class represents a differentiable vector bundle \( E \rightarrow M \)

**INPUT:**

- rank – positive integer; rank of the vector bundle
- name – string representation given to the total space
- base_space – the base space (differentiable manifold) \( M \) over which the vector bundle is defined
- field – field \( K \) which gives the fibers the structure of a vector space over \( K \); allowed values are
  - 'real' or an object of type RealField (e.g., RR) for a vector bundle over \( \mathbb{R} \)
  - 'complex' or an object of type ComplexField (e.g., CC) for a vector bundle over \( \mathbb{C} \)
  - an object in the category of topological fields (see Fields and TopologicalSpaces) for other types of topological fields
- latex_name – (default: None) \( \text{LaTeX} \) representation given to the total space
- category – (default: None) to specify the category; if None, VectorBundles(base_space, c_field).Differentiable() is assumed (see the category VectorBundles)
EXAMPLES:

A differentiable vector bundle of rank 2 over a 3-dimensional differentiable manifold:

```python
sage: M = Manifold(3, 'M')
sage: E = M.vector_bundle(2, 'E', field='complex'); E
Differentiable complex vector bundle E -> M of rank 2 over the base
space 3-dimensional differentiable manifold M
sage: E.category()
Category of smooth vector bundles over Complex Field with 53 bits of
precision with base space 3-dimensional differentiable manifold M
```

At this stage, the differentiable vector bundle has the same differentiability degree as the base manifold:

```python
sage: M.diff_degree() == E.diff_degree()
True
```

`bundle_connection(name, latex_name=None)`

Return a bundle connection on `self`.

**OUTPUT:**

• a bundle connection on `self` as an instance of `BundleConnection`

**EXAMPLES:**

```python
sage: M = Manifold(3, 'M', start_index=1)
sage: X.<x,y,z> = M.chart()
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e') # standard frame for E
sage: nab = E.bundle_connection('nabla', latex_name='\nabla'); nab
Bundle connection nabla on the Differentiable real vector bundle
E -> M of rank 2 over the base space 3-dimensional differentiable
manifold M
```

See also:

Further examples can be found in `BundleConnection`.

`characteristic_class(*args, **kwds)`

Deprecated: Use `characteristic_cohomology_class()` instead. See github issue #29581 for details.

`characteristic_cohomology_class(*args, **kwargs)`

Return a characteristic cohomology class associated with the input data.

**INPUT:**

• `val` – the input data associated with the characteristic class using the Chern-Weil homomorphism; this argument can be either a symbolic expression, a polynomial or one of the following predefined classes:
  - 'Chern' – total Chern class,
  - 'ChernChar' – Chern character,
  - 'Todd' – Todd class,
  - 'Pontryagin' – total Pontryagin class,
  - 'Hirzebruch' – Hirzebruch class,
  - 'AHat' – $\hat{A}$ class,

2.15. Differentiable Vector Bundles
- 'Euler' – Euler class.
- base_ring – (default: \(\mathbb{Q}\)) base ring over which the characteristic cohomology class ring shall be defined
- name – (default: None) string representation given to the characteristic cohomology class; if None the default algebra representation or predefined name is used
- latex_name – (default: None) LaTeX name given to the characteristic class; if None the value of name is used
- class_type – (default: None) class type of the characteristic cohomology class; the following options are possible:
  - 'multiplicative' – returns a class of multiplicative type
  - 'additive' – returns a class of additive type
  - 'Pfaffian' – returns a class of Pfaffian type
This argument must be stated if \(\text{val}\) is a polynomial or symbolic expression.

EXAMPLES:

Pontryagin class on the Minkowski space:

```
sage: M = Manifold(4, 'M', structure='Lorentzian', start_index=1)
sage: X.<t,x,y,z> = M.chart()
sage: g = M.metric()
sage: g[1,1] = -1
sage: g[2,2] = 1
sage: g[3,3] = 1
sage: g[4,4] = 1
sage: g.display()
g = -dt⊗dt + dx⊗dx + dy⊗dy + dz⊗dz
```

Let us introduce the corresponding Levi-Civita connection:

```
sage: nab = g.connection(); nab
Levi-Civita connection nabla_g associated with the Lorentzian metric g on the 4-dimensional Lorentzian manifold M
sage: nab.set_immutable()  # make nab immutable
```

Of course, \(\nabla_g\) is flat:

```
sage: nab.display()
```

Let us check the total Pontryagin class which must be the one element in the corresponding cohomology ring in this case:

```
sage: TM = M.tangent_bundle(); TM
Tangent bundle TM over the 4-dimensional Lorentzian manifold M
sage: p = TM.characteristic_cohomology_class('Pontryagin'); p
Characteristic cohomology class p(TM) of the Tangent bundle TM over the 4-dimensional Lorentzian manifold M
sage: p_form = p.get_form(nab); p_form.display_expansion()
p(TM, nabla_g) = 1
```

See also:

More examples can be found in `CharacteristicClass`. 
characteristic_cohomology_class_ring(base=Rational Field)

Return the characteristic cohomology class ring of self over a given base.

INPUT:

* base – (default: QQ) base over which the ring should be constructed; typically that would be \(\mathbb{Z}, \mathbb{Q}, \mathbb{R}\) or the symbolic ring

EXAMPLES:

```
sage: M = Manifold(4, 'M', start_index=1)
sage: R = M.tangent_bundle().characteristic_cohomology_class_ring()
sage: R
Algebra of characteristic cohomology classes of the Tangent bundle TM over the 4-dimensional differentiable manifold M
sage: p1 = R.gen(0); p1
Characteristic cohomology class \((p_1)(TM)\) of the Tangent bundle TM over the 4-dimensional differentiable manifold M
sage: 1 + p1
Characteristic cohomology class \((1 + p_1)(TM)\) of the Tangent bundle TM over the 4-dimensional differentiable manifold M
```

diff_degree()

Return the vector bundle’s degree of differentiability.

The degree of differentiability is the integer \(k\) (possibly \(k = \infty\)) such that the vector bundle is of class \(C^k\) over its base field. The degree always corresponds to the degree of differentiability of its base space.

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: E = M.vector_bundle(2, 'E')
sage: E.diff_degree()
+Infinity
sage: M = Manifold(2, 'M', structure='differentiable', ....: diff_degree=3)
sage: E = M.vector_bundle(2, 'E')
sage: E.diff_degree()
3
```

total_space()

Return the total space of self.

Note: At this stage, the total space does not come with induced charts.

OUTPUT:

* the total space of self as an instance of DifferentiableManifold

EXAMPLES:

```
sage: M = Manifold(3, 'M')
sage: E = M.vector_bundle(2, 'E')
sage: E.total_space()
6-dimensional differentiable manifold E
```

\textbf{class} \ \texttt{sage.manifolds.differentiable.vector_bundle.TensorBundle}(base_space, k, l, dest_map=None)
Bases: `DifferentiableVectorBundle`

Tensor bundle over a differentiable manifold along a differentiable map.

An instance of this class represents the pullback tensor bundle \( \Phi^* T^{(k,l)} N \) along a differentiable map (called destination map)

\[
\Phi : M \rightarrow N
\]

between two differentiable manifolds \( M \) and \( N \) over the topological field \( K \).

More precisely, \( \Phi^* T^{(k,l)} N \) consists of all pairs \((p,t) \in M \times T^{(k,l)} N \) such that \( t \in T_q^{(k,l)} N \) for \( q = \Phi(p) \), namely

\[
t : T_q^a N \times \cdots \times T_q^a N \times T_q N \times \cdots \times T_q N \rightarrow K
\]

\((k\) is called the contravariant and \( l\) the covariant rank of the tensor bundle\).

The trivializations are directly given by charts on the codomain (called ambient domain) of \( \Phi \). In particular, let \((V, \varphi)\) be a chart of \( N \) with components \((x^1, \ldots, x^n)\) such that \( q = \Phi(p) \in V \). Then, the matrix entries of \( t \in T_q^{(k,l)} N \) are given by

\[
t^{a_1 \cdots a_k}_{b_1 \cdots b_l} = t \left( \frac{\partial}{\partial x^{a_1}} \bigg|_q, \ldots, \frac{\partial}{\partial x^{a_k}} \bigg|_q, dx^{b_1} \bigg|_q, \ldots, dx^{b_l} \bigg|_q \right) \in K
\]

and a trivialization over \( U = \Phi^{-1}(V) \subset M \) is obtained via

\[
(p, t) \mapsto (p, t_1^{1}, \ldots, t_n^{n}, \ldots, t_1^{1}, \ldots, t_n^{n}) \in U \times K^{n(k+l)}.
\]

The standard case of a tensor bundle over a differentiable manifold corresponds to \( M = N \) 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 \( \mathbb{R} \)).

**INPUT:**

- `base_space` – the base space (differentiable manifold) \( M \) over which the tensor bundle is defined
- \( k \) – the contravariant rank of the corresponding tensor bundle
- \( l \) – the covariant rank of the corresponding tensor bundle
- `dest_map` – (default: `None`) destination map \( \Phi : M \rightarrow N \) (type: `DiffMap`); if `None`, it is assumed that \( M = M \) and \( \Phi \) is the identity map of \( M \) (case of the standard tensor bundle over \( M \))

**EXAMPLES:**

Pullback tangent bundle of \( \mathbb{R}^2 \) along a curve \( \Phi \):

```python
sage: M = Manifold(2, 'M')
sage: c_cart.<x,y> = M.chart()
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: PhiTM = R.tangent_bundle(dest_map=Phi); PhiTM
Tangent bundle Phi^*TM over the 1-dimensional differentiable manifold R along the Differentiable map Phi from the 1-dimensional differentiable manifold M to the 2-dimensional differentiable manifold M
```
The section module is the corresponding tensor field module:

```
sage: R_tensor_module = R.tensor_field_module((1,0), dest_map=Phi)
sage: R_tensor_module is PhiTM.section_module()
True
```

**ambient_domain()**

Return the codomain of the destination map.

**OUTPUT:**

- a `DifferentiableManifold` representing the codomain of the destination map

**EXAMPLES:**

```
sage: M = Manifold(2, 'M')
sage: c_cart.<x,y> = M.chart()
sage: e_cart = c_cart.frame() # standard basis
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: PhiT11 = R.tensor_bundle(1, 1, dest_map=Phi)
sage: PhiT11.ambient_domain()
2-dimensional differentiable manifold M
```

**atlas()**

Return the list of charts that have been defined on the codomain of the destination map.

**Note:** Since an atlas of charts gives rise to an atlas of trivializations, this method directly invokes `atlas()` of the class `TopologicalManifold`.

**EXAMPLES:**

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: Y.<u,v> = M.chart()
sage: TM = M.tangent_bundle()
sage: TM.atlas()
[Chart (M, (x, y)), Chart (M, (u, v))]
```

**change_of_frame(frame1, frame2)**

Return a change of vector frames defined on the base space of `self`.

**See also:**

For further details on frames on `self` see `local_frame()`.

**Note:** Since frames on `self` are directly induced by vector frames on the base space, this method directly invokes `change_of_frame()` of the class `DifferentiableManifold`.

**INPUT:**
• frame1 – local frame 1
• frame2 – local frame 2

OUTPUT:
• a FreeModuleAutomorphism representing, at each point, the vector space automorphism \( P \) that relates frame 1, \((e_i)\), to frame 2, \((f_i)\), say, according to \( f_i = P(e_i) \)

EXAMPLES:

```sage
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: TM = M.tangent_bundle()
sage: TM.change_of_frame(c_xy.frame(), c_uv.frame())
Field of tangent-space automorphisms on the 2-dimensional differentiable manifold M
sage: TM.change_of_frame(c_xy.frame(), c_uv.frame())[:,]
[1/2 1/2]
[1/2 -1/2]
sage: TM.change_of_frame(c_uv.frame(), c_xy.frame())
Field of tangent-space automorphisms on the 2-dimensional differentiable manifold M
sage: TM.change_of_frame(c_uv.frame(), c_xy.frame())[:,]
[1 1]
[1 -1]
sage: TM.change_of_frame(c_uv.frame(), c_xy.frame()) == \ 
....: M.change_of_frame(c_xy.frame(), c_uv.frame()).inverse()
True
```

changes_of_frame()

Return the changes of vector frames defined on the base space of self with respect to the destination map.

See also:
For further details on frames on self see `local_frame()`.

OUTPUT:
• dictionary of automorphisms on the tangent bundle 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
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: TM = M.tangent_bundle()
sage: e = X.frame(); e
Coordinate frame (M, (∂/∂x,∂/∂y))
```

At this stage, the dictionary of changes of frame is empty:

```sage
sage: TM.changes_of_frame()
()
```

We introduce a second frame on the manifold, relating it to frame \( e \) by a field of tangent space automorphisms:
Then we have:

```python
sage: TM.changes_of_frame() # random (dictionary output)
{(Coordinate frame (M, (∂/∂x,∂/∂y)), 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, (∂/∂x,∂/∂y))): Field of tangent-space automorphisms on the 2-dimensional differentiable manifold M)
```

Some checks:

```python
sage: TM.changes_of_frame()[(e,f)] == a
True
sage: TM.changes_of_frame()[(f,e)] == a^(-1)
True
```

coframes()

Return the list of coframes defined on the base manifold of self with respect to the destination map.

See also:

For further details on frames on self see local_frame().

OUTPUT:

- list of coframes defined on self

EXAMPLES:

Coframes on subsets of $\mathbb{R}^2$:

```python
sage: M = Manifold(2, 'R^2')
sage: c_cart.<x,y> = M.chart() # Cartesian coordinates on R^2
sage: TM = M.tangent_bundle()
sage: TM.coframes()
[Coordinate coframe (R^2, (dx,dy))]
sage: e = TM.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})
sage: TU = U.tangent_bundle()
sage: TU.coframes()
[Coordinate coframe (U, (dx,dy)), Coframe (U, (e^0,e^1))]
sage: e.restrict(U)
Vector frame (U, (e_0,e_1))
sage: TU.coframes()
[Coordinate coframe (U, (dx,dy)), Coframe (U, (e^0,e^1))]
sage: TM.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))]
```
**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 pulled back tangent space at \( p \).

If the destination map is the identity map, the default frame is the the first one defined on the manifold, usually the coordinate frame, unless it is changed via `set_default_frame()`.

If the destination map is non-trivial, the default frame usually must be set via `set_default_frame()`.

**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
sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart()
sage: TM = M.tangent_bundle()
sage: TM.default_frame()
Coordinate frame (M, (∂/∂x,∂/∂y))
```

**destination_map()**
Return the destination map.

**OUTPUT:**
- a `DifferentialMap` representing the destination map

**EXAMPLES:**

```sage
sage: M = Manifold(2, 'M')
sage: c_cart.<x,y> = M.chart()
sage: e_cart = c_cart.frame()  # standard basis
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: PhiT11 = R.tensor_bundle(1, 1, dest_map=Phi)
sage: PhiT11.destination_map()
Differentiable map Phi from the 1-dimensional differentiable manifold R to the 2-dimensional differentiable manifold M
```

**fiber(point)**
Return the tensor bundle fiber over a point.

**INPUT:**
- `point` – `ManifoldPoint`: point \( p \) of the base manifold of `self`

**OUTPUT:**
- an instance of `FiniteRankFreeModule` representing the tensor bundle fiber over \( p \)

**EXAMPLES:**
frames()  
Return the list of all vector frames defined on the base space of \texttt{self} with respect to the destination map.

See also:

For further details on frames on \texttt{self} see \texttt{local_frame}().

OUTPUT:

- list of local frames defined on \texttt{self}

EXAMPLES:

Vector frames on subsets of \( \mathbb{R}^2 \):

```python
sage: M = Manifold(2, 'M')
sage: c_cart.<x,y> = M.chart()  # Cartesian coordinates on \( \mathbb{R}^2 \)
sage: TM = M.tangent_bundle()
sage: TM.frames()
[Coordinate frame (\mathbb{R}^2, (\partial/\partial x, \partial/\partial y))]
sage: e = TM.vector_frame('e')
sage: TM.frames()
[Coordinate frame (\mathbb{R}^2, (\partial/\partial x, \partial/\partial y)),
 Vector frame (\mathbb{R}^2, (e_0, e_1))]
sage: U = M.open_subset('U', coord_def={c_cart: x^2+y^2<1})
sage: TU = U.tangent_bundle()
sage: TU.frames()
[Coordinate frame (U, (\partial/\partial x, \partial/\partial y))]
sage: TM.frames()
[Coordinate frame (\mathbb{R}^2, (\partial/\partial x, \partial/\partial y)),
 Vector frame (\mathbb{R}^2, (e_0, e_1)),
 Coordinate frame (U, (\partial/\partial x, \partial/\partial y))]
```

List of vector frames of a tensor bundle of type \((1, 1)\) along a curve:

```python
sage: M = Manifold(2, 'M')
sage: c_cart.<x,y> = M.chart()
sage: e_cart = c_cart.frame()  # standard basis
```

(continues on next page)
is_manifestly_trivial()

Return True if self is known to be a trivial and False otherwise.

If False is returned, either the tensor bundle is not trivial or no vector frame has been defined on it yet.

EXAMPLES:

A just created manifold has a priori no manifestly trivial tangent bundle:

\begin{verbatim}
sage: M = Manifold(2, 'M')
sage: TM = M.tangent_bundle()
sage: TM.is_manifestly_trivial()
False
\end{verbatim}

Defining a vector frame on it makes it trivial:

\begin{verbatim}
sage: e = TM.vector_frame('e')
sage: TM.is_manifestly_trivial()
True
\end{verbatim}

Defining a coordinate chart on the whole manifold also makes it trivial:

\begin{verbatim}
sage: N = Manifold(4, 'N')
sage: X.<t,x,y,z> = N.chart()
sage: TN = N.tangent_bundle()
sage: TN.is_manifestly_trivial()
True
\end{verbatim}

The situation is not so clear anymore when a destination map to a non-parallelizable manifold is stated:

\begin{verbatim}
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() # stereo coord from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv, <u,v> = V.chart() # stereo coord 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)),
\end{verbatim}
A priori, the pullback tangent bundle is not trivial:

\[ \text{sage: PhiTU.is_manifestly_trivial()} \]
False

But certainly, this bundle must be trivial since \( U \) is parallelizable. To ensure this, we need to define a local frame on \( U \) with values in \( \Phi^*T^2 \):

\[ \text{sage: PhiTU.local_frame('e', from_frame=c_xy.frame())} \]
Vector frame \( (U, (e_0, e_1)) \) with values on the 2-dimensional differentiable manifold \( S^2 \)
\[ \text{sage: PhiTU.is_manifestly_trivial()} \]
True

**local_frame** (*args, **kwargs)

Define a vector frame on domain, possibly with values in the tangent bundle of the ambient domain.

If the basis specified by the given symbol already exists, it is simply returned. If no argument is provided the vector field module’s default frame is returned.

Notice, that a vector frame automatically induces a local frame on the tensor bundle \( \text{self} \). More precisely, if \( e : U \to \Phi^*T^2 \) is a vector frame on \( U \subset M \) with values in \( \Phi^*T^2 \) along the destination map

\[ \Phi : M \longrightarrow N \]
then the map

\[ p \mapsto (e^*(p), \ldots, e^*(p), e(p), \ldots, e(p)) \in T^{(k,l)}_q N, \]

with \( q = \Phi(p) \), defines a basis at each point \( p \in U \) and therefore gives rise to a local frame on \( \Phi^*T^{(k,l)} N \) on the domain \( U \).

See also:

* **VectorFrame** for complete documentation.

**INPUT:**

- **symbol** – (default: None) either a string, to be used as a common base for the symbols of the vector fields constituting the vector frame, or a list/tuple of strings, representing the individual symbols of the vector fields; can be None only if **from_frame** is not None (see below)
• vector_fields — tuple or list of \( n \) linearly independent vector fields on domain \( (n \) being the dimension of domain) defining the vector frame; can be omitted if the vector frame is created from scratch or if from_frame is not None

• latex_symbol — (default: None) either a string, to be used as a common base for the \( \text{LaTeX} \) symbols of the vector fields constituting the vector frame, or a list/tuple of strings, representing the individual \( \text{LaTeX} \) symbols of the vector fields; if None, symbol is used in place of latex_symbol

• from_frame — (default: None) vector frame \( \tilde{e} \) on the codomain \( N \) of the destination map \( \Phi \); the returned frame \( e \) is then such that for all \( p \in U \), we have \( e(p) = \tilde{e}(\Phi(p)) \)

• indices — (default: None; used only if symbol is a single string) tuple of strings representing the indices labelling the vector fields of the frame; if None, the indices will be generated as integers within the range declared on self

• latex_indices — (default: None) tuple of strings representing the indices for the \( \text{LaTeX} \) symbols of the vector fields; if None, indices is used instead

• symbol_dual — (default: None) same as symbol but for the dual coframe; if None, symbol must be a string and is used for the common base of the symbols of the elements of the dual coframe

• latex_symbol_dual — (default: None) same as latex_symbol but for the dual coframe

• domain — (default: None) domain on which the local frame is defined; if None is provided, the base space of self is assumed

OUTPUT:

• the vector frame corresponding to the above specifications; this is an instance of VectorFrame.

EXAMPLES:

Defining a local frame for the tangent bundle of a 3-dimensional manifold:

\[
\text{sage: } M = \text{Manifold}(3, 'M') \\
\text{sage: } TM = M.tangent_bundle() \\
\text{sage: } e = TM.local_frame('e'); e \\
\text{Vector frame (M, (e_0,e_1,e_2))} \\
\text{sage: } e[0] \\
\text{Vector field e_0 on the 3-dimensional differentiable manifold M}
\]

Specifying the domain of the vector frame:

\[
\text{sage: } U = M.open_subset('U') \\
\text{sage: } f = TM.local_frame('f', \text{domain=U}); f \\
\text{Vector frame (U, (f_0,f_1,f_2))} \\
\text{sage: } f[0] \\
\text{Vector field f_0 on the Open subset U of the 3-dimensional differentiable manifold M}
\]

See also:

For more options, in particular for the choice of symbols and indices, see VectorFrame.

orientation() Get the preferred orientation of self if available.

See orientation() for details regarding orientations on vector bundles.

The tensor bundle \( \Phi^*T^{(k,l)}N \) of a manifold is orientable if the manifold \( \Phi(M) \) is orientable. The converse does not necessarily hold true. The usual case corresponds to \( \Phi \) being the identity map, where the tensor bundle \( T^{(k,l)}M \) is orientable if and only if the manifold \( M \) is orientable.
Note: Notice that the orientation of a general tensor bundle $\Phi^*T^{(k,l)}_N$ is canonically induced by the orientation of the tensor bundle $\Phi^*T^{(1,0)}_N$ as each local frame there induces the frames on $\Phi^*T^{(k,l)}_N$ in a canonical way.

If no preferred orientation has been set before, and if the ambient space already admits a preferred orientation, the corresponding orientation is returned and henceforth fixed for the tensor bundle.

EXAMPLES:

In the trivial case, i.e. if the destination map is the identity and the tangent bundle is covered by one frame, the orientation is easily obtained:

```
sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart()
sage: T11 = M.tensor_bundle(1, 1)
sage: T11.orientation()
[Coordinate frame (M, (∂/∂x, ∂/∂y))]
```

The same holds true if the ambient domain admits a trivial orientation:

```
sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart()
sage: R = Manifold(1, 'R')
sage: c_t.<t> = R.chart()
sage: Phi = R.diff_map(M, name='Phi')
sage: PhiT22 = R.tensor_bundle(2, 2, dest_map=Phi); PhiT22
Tensor bundle $\Phi^*T^{(2,2)}_M$ over the 1-dimensional differentiable manifold $\mathbb{R}$ along the Differentiable map $\Phi$ from the 1-dimensional differentiable manifold $\mathbb{R}$ to the 2-dimensional differentiable manifold $\mathbb{M}$
sage: PhiT22.local_frame()  # initialize frame
Vector frame $(\mathbb{R}, (\partial/\partial x, \partial/\partial y))$ with values on the 2-dimensional differentiable manifold $\mathbb{M}$
sage: PhiT22.orientation()
[Vector frame $(\mathbb{R}, (\partial/\partial x, \partial/\partial y))$ with values on the 2-dimensional differentiable manifold $\mathbb{M}$]
sage: PhiT22.local_frame() is PhiT22.orientation()[0]
True
```

In the non-trivial case, however, the orientation must be set manually by the user:

```
sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U'); V = M.open_subset('V')
sage: M.declare_union(U, V)
sage: c_xy.<x,y> = U.chart(); c_uv.<u,v> = V.chart()
sage: T11 = M.tensor_bundle(1, 1); T11
Tensor bundle $T^{(1,1)}_M$ over the 2-dimensional differentiable manifold $\mathbb{M}$
sage: T11.orientation()
[]
sage: T11.set_orientation([c_xy.frame(), c_uv.frame()])
sage: T11.orientation()
[Coordinate frame (U, (∂/∂x, ∂/∂y)), Coordinate frame (V, (∂/∂u, ∂/∂v))]
```

If the destination map is the identity, the orientation is automatically set for the manifold, too:
Conversely, if one sets an orientation on the manifold, the orientation on its tensor bundles is set accordingly:

\[
sage: c_{tz}.\langle t, z \rangle = U\text{.chart()}
\]
\[
sage: M.set_orientation([c_{tz}, c_{uv}])
\]
\[
sage: T11.orientation()
\]

```
[Coordinate frame (U, (∂/∂t,∂/∂z)), Coordinate frame (V, (∂/∂u,∂/∂v))]
```

**section(**\*args**, **\*kwargs**)**

Return a section of self on domain, namely a tensor field on the subset domain of the base space.

**Note:** This method directly invokes `tensor_field()` of the class `DifferentiableManifold`.

**INPUT:**

- `comp` – (optional) either the components of the tensor field with respect to the vector frame specified by the argument `frame` or a dictionary of components, the keys of which are vector frames or pairs `(f, c)` where `f` is a vector frame and `c` the chart in which the components are expressed
- `frame` – (default: None; unused if `comp` is not given or is a dictionary) vector frame in which the components are given; if None, the default vector frame of self is assumed
- `chart` – (default: None; unused if `comp` is not given or is a dictionary) coordinate chart in which the components are expressed; if None, the default chart on the domain of `frame` is assumed
- `domain` – (default: None) domain of the section; if None, self.base_space() is assumed
- `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`

**OUTPUT:**

- a `TensorField` (or if \( N \) is parallelizable, a `TensorFieldParallel`) representing the defined tensor field on the domain \( U \subset M \)

**EXAMPLES:**

```
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: T11M = M.tensor_bundle(1, 1); T11M
Tensor bundle $T^{(1,1)}M$ over the 2-dimensional differentiable manifold $M$
sage: t = T11M.section({eU: 
[1, x], 
[0, 2]}; name =t); t
Tensor field t of type (1,1) on the 2-dimensional differentiable manifold $M$
sage: t.display()
t = $\frac{\partial}{\partial x}\otimes dx + x \frac{\partial}{\partial x}\otimes dy + 2 \frac{\partial}{\partial y}\otimes dy$

An example of use with the arguments comp and domain:

sage: TM = M.tangent_bundle()
sage: w = TM.section([-y, x], domain=U); w
Vector field on the Open subset U of the 2-dimensional differentiable manifold M
sage: w.display()
$-y \frac{\partial}{\partial x} + x \frac{\partial}{\partial y}$

section_module(domain=None)

Return the section module on domain, namely the corresponding tensor field module, of self on domain.

Note: This method directly invokes tensor_field_module() of the class DifferentiableManifold.

INPUT:

* domain – (default: None) the domain of the corresponding section module; if None, the base space is assumed

OUTPUT:

* a TensorFieldModule (or if $N$ is parallelizable, a TensorFieldFreeModule) representing the module $T^{(k,l)}(U,\Phi)$ of type-$(k,l)$ tensor fields on the domain $U \subset M$ taking values on $\Phi(U) \subset N$

EXAMPLES:

sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: U = M.open_subset('U')
sage: TM = M.tangent_bundle()
sage: TUM = TM.section_module(domain=U); TUM
Module $X(U)$ of vector fields on the Open subset $U$ of the 2-dimensional differentiable manifold $M$
sage: TUM is U.tensor_field_module((1,0))
True

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 of the base space $M$.  

2.15. Differentiable Vector Bundles
See also:

For further details on frames on self see :meth:`local_frame`.

**Note:** Since frames on self are directly induced by vector frames on the base space, this method directly invokes :meth:`set_change_of_frame` of the class :class:`DifferentiableManifold`.

**INPUT:**

- frame1 – frame 1, denoted \((e_i)\) below
- frame2 – frame 2, denoted \((f_i)\) below
- change_of_frame – instance of class :class:`FreeModuleAutomorphism` 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:**

```
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: TM = M.tangent_bundle()
sage: TM.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: TM.change_of_frame(e,f)[e,:]  
[ 1  2]
[ 0  3]
```

.. note::

   If the destination map is the identity, the default frame of the base manifold gets changed here as well.

**INPUT:**

- frame – :class:`VectorFrame` a vector frame defined on the base manifold

**EXAMPLES:**

Changing the default frame on the tangent bundle of a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart()
sage: TM = M.tangent_bundle()
sage: e = TM.vector_frame('e')
```

(continues on next page)
**set_orientation** (*orientation*)

Set the preferred orientation of `self`.

**INPUT:**

- *orientation* – a vector frame or a list of vector frames, covering the base space of `self`

**Note:** If the destination map is the identity, the preferred orientation of the base manifold gets changed here as well.

**Warning:** It is the user's responsibility that the orientation set here is indeed an orientation. There is no check going on in the background. See `orientation()` for the definition of an orientation.

**EXAMPLES:**

Set an orientation on a tensor bundle:

```python
sage: M = Manifold(2, 'M')
sage: c_xy.<x,y> = M.chart()
sage: T11 = M.tensor_bundle(1, 1)
sage: e = T11.local_frame('e'); e
Vector frame (M, (e_0,e_1))
sage: T11.set_orientation(e)
sage: T11.orientation()
[Vector frame (M, (e_0,e_1))]
```

Set an orientation in the non-trivial case:

```python
sage: M = Manifold(2, 'M')
sage: U = M.open_subset('U'); V = M.open_subset('V')
sage: M.declare_union(U, V)
sage: c_xy.<x,y> = U.chart(); c_uv.<u,v> = V.chart()
sage: T12 = M.tensor_bundle(1, 2)
sage: e = T12.local_frame('e', domain=U)
sage: f = T12.local_frame('f', domain=V)
sage: T12.set_orientation([e, f])
sage: T12.orientation()
[Vector frame (U, (e_0,e_1)), Vector frame (V, (f_0,f_1))]
```

**transition** (*chart1, chart2*)

Return the change of trivializations in terms of a coordinate change between two differentiable charts defined on the codomain of the destination map.

The differentiable chart must have been defined previously, for instance by the method `transition_map()`.

**2.15. Differentiable Vector Bundles**
Note: Since a chart gives direct rise to a trivialization, this method is nothing but an invocation of `co-ord_change()` of the class `TopologicalManifold`.

**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')
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 coord. change
Change of coordinates from Chart (M, (x, y)) to Chart (M, (u, v))
sage: TM = M.tangent_bundle()
sage: TM.transition(c_xy, c_uv) # returns the coord. change above
Change of coordinates from Chart (M, (x, y)) to Chart (M, (u, v))
```

**transitions()**

Return the transition maps between trivialization maps in terms of coordinate changes defined via charts on the codomain of the destination map.

Note: Since a chart gives direct rise to a trivialization, this method is nothing but an invocation of `co-ord_changes()` of the class `TopologicalManifold`.

**EXAMPLES:**

Various changes of coordinates on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M')
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: TM = M.tangent_bundle()
sage: TM.transitions() # random (dictionary output)
{(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: TM.transitions() # 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: TM.transitions() # random (dictionary output)
```

(continues on next page)
{{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 \((M, (u, v))\))
}

```
sage: xy_to_rs = uv_to_rs * xy_to_uv
sage: TM.transitions()  # 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 \((M, (r, s))\))
```

### trivalization \((coordinates='', names=None, calc_method=None)\)

Return a trivialization of \(self\) in terms of a chart on the codomain of the destination map.

**Note:** Since a chart gives direct rise to a trivialization, this method is nothing but an invocation of \(chart()\) of the class \(TopologicalManifold\).

**INPUT:**

- **coordinates** – (default: ' ' (empty string)) string defining the coordinate symbols, ranges and possible periodicities, 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)
- **calc_method** – (default: None) string defining the calculus method to be used on this chart; must be one of
  - 'SR': Sage's default symbolic engine (Symbolic Ring)
  - 'sympy': SymPy
  - None: the current calculus method defined on the manifold is used (cf. \(set_calculus_method()\))

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

1. The coordinate symbol (a letter or a few letters).
2. (optional, only for manifolds over \(\mathbb{R}\)) The interval \(I\) defining the coordinate range: if not provided, the coordinate is assumed to span all \(\mathbb{R}\); 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 space character.
3. (optional) Indicator of the periodic character of the coordinate, either as period=T, where T is the period, or, for manifolds over \( \mathbb{R} \) only, as the keyword periodic (the value of the period is then deduced from the interval \( I \) declared in field 2; see the example below)

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

The order of fields 2 to 4 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, no period and no LaTeX spelling is to be set for any coordinate, the argument coordinates can be omitted when the shortcut operator <, > is used to declare the trivialization.

OUTPUT:

• the created chart, as an instance of Chart or one of its subclasses, like RealDiffChart for differentiable manifolds over \( \mathbb{R} \).

EXAMPLES:

Chart on a 2-dimensional manifold:

```python
sage: M = Manifold(2, 'M')
sage: TM = M.tangent_bundle()
sage: X = TM.trivialization('x y'); X
Chart (M, (x, y))
sage: X[0]
x
sage: X[1]
y
sage: X[:]
(x, y)
```

`vector_frame(*args, **kwargs)`

Define a vector frame on domain, possibly with values in the tangent bundle of the ambient domain.

If the basis specified by the given symbol already exists, it is simply returned. If no argument is provided the vector field module’s default frame is returned.

Notice, that a vector frame automatically induces a local frame on the tensor bundle self. More precisely, if \( e : U \to \Phi^*TN \) is a vector frame on \( U \subset M \) with values in \( \Phi^*TN \) along the destination map

\[
\Phi : M \to N
\]

then the map

\[
p \mapsto \left( e^*(p), \ldots, e^*(p), e(p), \ldots, e(p) \right) \in T_q^{(k,l)}N,
\]

with \( q = \Phi(p) \), defines a basis at each point \( p \in U \) and therefore gives rise to a local frame on \( \Phi^*T^{(k,l)}N \) on the domain \( U \).

See also:

VectorFrame for complete documentation.

INPUT:

• symbol – (default: None) either a string, to be used as a common base for the symbols of the vector fields constituting the vector frame, or a list/tuple of strings, representing the individual symbols of the vector fields; can be None only if from_frame is not None (see below)
• **vector_fields** – tuple or list of $n$ linearly independent vector fields on domain ($n$ being the dimension of domain) defining the vector frame; can be omitted if the vector frame is created from scratch or if from_frame is not None

• **latex_symbol** – (default: None) either a string, to be used as a common base for the LaTeX symbols of the vector fields constituting the vector frame, or a list/tuple of strings, representing the individual LaTeX symbols of the vector fields; if None, symbol is used in place of latex_symbol

• **from_frame** – (default: None) vector frame $\tilde{e}$ on the codomain $N$ of the destination map $\Phi$; the returned frame $e$ is then such that for all $p \in U$, we have $e(p) = \tilde{e}(\Phi(p))$

• **indices** – (default: None; used only if symbol is a single string) tuple of strings representing the indices labelling the vector fields of the frame; if None, the indices will be generated as integers within the range declared on self

• **latex_indices** – (default: None) tuple of strings representing the indices for the LaTeX symbols of the vector fields; if None, indices is used instead

• **symbol_dual** – (default: None) same as symbol but for the dual coframe; if None, symbol must be a string and is used for the common base of the symbols of the elements of the dual coframe

• **latex_symbol_dual** – (default: None) same as latex_symbol but for the dual coframe

• **domain** – (default: None) domain on which the local frame is defined; if None is provided, the base space of self is assumed

**OUTPUT:**

• the vector frame corresponding to the above specifications; this is an instance of `VectorFrame`

**EXAMPLES:**

Defining a local frame for the tangent bundle of a 3-dimensional manifold:

```
sage: M = Manifold(3, 'M')
sage: TM = M.tangent_bundle()
sage: e = TM.local_frame('e'); e
Vector frame (M, (e_0,e_1,e_2))
sage: e[0]
Vector field e_0 on the 3-dimensional differentiable manifold M
```

Specifying the domain of the vector frame:

```
sage: U = M.open_subset('U')
sage: f = TM.local_frame('f', domain=U); f
Vector frame (U, (F_0,F_1,F_2))
sage: f[0]
Vector field f_0 on the Open subset U of the 3-dimensional differentiable manifold M
```

See also:

For more options, in particular for the choice of symbols and indices, see `VectorFrame`. 

2.15. Differentiable Vector Bundles 813
2.15.2 Bundle Connections

Let $E \to M$ be a smooth vector bundle of rank $n$ over a smooth manifold $M$ and over a non-discrete topological field $K$ (typically $K = \mathbb{R}$ or $K = \mathbb{C}$). A bundle connection on this vector bundle is a $K$-linear map

$$\nabla : C^\infty(M; E) \to C^\infty(M; E \otimes T^* M)$$

such that the Leibniz rule applies for each scalar field $f \in C^\infty(M)$ and section $s \in C^\infty(M; E)$:

$$\nabla(fs) = f \cdot \nabla s + s \otimes df.$$ 

If $e$ is a local frame on $E$, we have

$$\nabla e_i = \sum_{j=1}^n e_j \otimes \omega^j_i,$$

and the corresponding $n \times n$-matrix $(\omega^j_i)_{i,j}$ consisting of one forms is called connection matrix of $\nabla$ with respect to $e$.

AUTHORS:

- Michael Jung (2019): initial version

```
class sage.manifolds.differentiable.bundle_connection.BundleConnection(vbundle, name, latex_name=None)
```

Bases: SageObject, Mutability

An instance of this class represents a bundle connection $\nabla$ on a smooth vector bundle $E \to M$.

INPUT:

- $\text{vbundle}$ – the vector bundle on which the connection is defined (must be an instance of class DifferentiableVectorBundle)
- $\text{name}$ – name given to the bundle connection
- $\text{latex_name}$ – (default: None) LaTeX symbol to denote the bundle connection; if None, it is set to name

EXAMPLES:

Define a bundle connection on a rank 2 vector bundle over some 3-dimensional smooth manifold:

```
sage: M = Manifold(3, 'M', start_index=1)
sage: X.<x,y,z> = M.chart()
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e') # standard frame for E
sage: nab = E.bundle_connection('nabla'); nab
Bundle connection nabla on the Differentiable real vector bundle E -> M
of rank 2 over the base space 3-dimensional differentiable manifold M
```

First, let us initialize all connection 1-forms w.r.t. the frame $e$ to zero:

```
sage: nab[e, :] = [[0, 0], [0, 0]]
```

This line can be shortened by the following:

```
sage: nab[e, :] = 0 # initialize to zero
```

The connection 1-forms are now initialized being differential 1-forms:
Now, we want to specify some non-zero entries:

```
sage: nab[e, 1, 2][:] = [x*z, y*z, z^2]
sage: nab[e, 2, 1][:] = [x, x^2, x^3]
sage: nab[e, 1, 1][:] = [x+z, y-z, x*y*z]
sage: nab.display()
```

connection (1,1) of bundle connection nabla w.r.t. Local frame
(\(E|_M, (e_1,e_2)\)) = (x + z) dx + (y - z) dy + x*y*z dz

connection (1,2) of bundle connection nabla w.r.t. Local frame
(\(E|_M, (e_1,e_2)\)) = x*z dx + y*z dy + z^2 dz

connection (2,1) of bundle connection nabla w.r.t. Local frame
(\(E|_M, (e_1,e_2)\)) = x dx + x^2 dy + x^3 dz

Notice, when we omit the frame, the default frame of the vector bundle is assumed (in this case \(e\)):

```
sage: nab[2, 2].display()
```

The same holds for the assignment:

```
sage: nab[1, 2] = 0

sage: nab[e, 1, 2].display()
```

connection (1,2) of bundle connection nabla w.r.t. Local frame
(\(E|_M, (e_1,e_2)\)) = 0

Keep noticed that item assignments for bundle connections only copy the right-hand-side and never create a binding to the original instance:

```
sage: omega = M.one_form('omega')
sage: omega[:] = [x*z, y*z, z^2]
sage: nab[1, 2] = omega
sage: nab[1, 2] == omega
True
sage: nab[1, 2] is omega
False
```

Hence, this is therefore equivalent to:

```
sage: nab[2, 2].copy_from(omega)
```

Preferably, we use `set_connection_form()` to specify the connection 1-forms:

```
sage: nab[:] = 0  # re-initialize to zero
sage: nab.set_connection_form(1, 2)[:] = [x*z, y*z, z^2]
sage: nab.set_connection_form(2, 1)[:] = [x, x^2, x^3]
sage: nab[1, 2].display()
```

connection (1,2) of bundle connection nabla w.r.t. Local frame
(\(E|_M, (e_1,e_2)\)) = x*z dx + y*z dy + z^2 dz

```
sage: nab[2, 1].display()
```

(continues on next page)
connection (2,1) of bundle connection nabla w.r.t. Local frame
(\mathcal{E}|_M, (e_1,e_2)) = x \, dx + x^2 \, dy + x^3 \, dz

Note: Notice that item assignments and \texttt{set_connection_form()} delete the connection 1-forms w.r.t. other frames for consistency reasons. To avoid this behavior, \texttt{add_connection_form()} must be used instead.

In conclusion, the connection 1-forms of a bundle connection are mutable until the connection itself is set immutable:

```
sage: nab.set_immutable()
sage: nab[1, 2] = omega
Traceback (most recent call last): ...
ValueError: object is immutable; please change a copy instead
```

By definition, a bundle connection acts on vector fields and sections:

```
sage: v = M.vector_field((x^2,y^2,z^2), name = 'v'); v.display()
v = x^2 \partial/\partial x + y^2 \partial/\partial y + z^2 \partial/\partial z
sage: s = E.section((x-y^2, -z), name = 's'); s.display()
s = (-y^2 + x) \, e_1 - z \, e_2
sage: nab_vs = nab(v, s); nab_vs.display()
connection nabla_v(s) on the 3-dimensional differentiable manifold M with values in the real vector bundle E of rank 2
nabla_v(s) = (-x^3*z^3 - 2*y^3 + x^2 - (x^2*y^2 + x^3)*z) \, e_1 +
(-y^2 - x)*z^4 - (x^3*y^2 + y^5 - x^4 - x*y^3)*z - z^2) \, e_2
```

The bundle connection action certainly obeys the defining formula for the connection 1-forms:

```
sage: vframe = X.frame()
sage: all(nab(vframe[k], e[i]) == sum(nab[e, i, j](vframe[k]) * e[j] ....: for j in E.irange()) ....: for i in E.irange() for k in M.irange())
True
```

The connection 1-forms are computed automatically for different frames:

```
sage: f = E.local_frame('f', ((1+x^2)*e[1], e[1]-e[2]))
sage: nab.display(frame=f)
connection (1,1) of bundle connection nabla w.r.t. Local frame
(\mathcal{E}|_M, (f_1, f_2)) = ((x^3 + x)*z + 2*x)/(x^2 + 1) \, dx + y*z \, dy + z^2 \, dz
connection (1,2) of bundle connection nabla w.r.t. Local frame
(\mathcal{E}|_M, (f_1, f_2)) = -(x^3 + x)*z \, dx - (x^2 + 1)*y*z \, dy -
(x^2 + 1)*z^2 \, dz
connection (2,1) of bundle connection nabla w.r.t. Local frame
(\mathcal{E}|_M, (f_1, f_2)) = (x*z - x)/(x^2 + 1) \, dx -
(x^2 - y*z)/(x^2 + 1) \, dy - (x^3 - z^2)/(x^2 + 1) \, dz
connection (2,2) of bundle connection nabla w.r.t. Local frame
(\mathcal{E}|_M, (f_1, f_2)) = -x*z \, dx - y*z \, dy - z^2 \, dz
```

The new connection 1-forms obey the defining formula, too:

```
sage: all(nab(vframe[k], f[i]) == sum(nab[f, i, j](vframe[k]) * f[j] ....: for j in E.irange())
```

(continues on next page)
... for i in E.irange() for k in M.irange())
True

After the connection has been specified, the curvature 2-forms can be derived:

```python
sage: Omega = nab.curvature_form
sage: for i in E.irange():
....: for j in E.irange():
....: print(Omega(i,j,e).display())
curvature (1,1) of bundle connection nabla w.r.t. Local frame
(E|_M, (e_1,e_2)) = -(x^3 - x*y)*z dx∧dy + (-x^4*z + x*z^2) dx∧dz +
(-x^3*y^2*z + x^2*z^2) dy∧dz
curvature (1,2) of bundle connection nabla w.r.t. Local frame
(E|_M, (e_1,e_2)) = -x dx∧dz - y dy∧dz
curvature (2,1) of bundle connection nabla w.r.t. Local frame
(E|_M, (e_1,e_2)) = 2*x dx∧dy + 3*x^2 dx∧dz
curvature (2,2) of bundle connection nabla w.r.t. Local frame
(E|_M, (e_1,e_2)) = (x^3 - x*y)*z dx∧dy + (x^4*z - x*z^2) dx∧dz +
(x^3*y^2*z - x^2*z^2) dy∧dz
```

The derived forms certainly obey the structure equations, see `curvature_form()` for details:

```python
sage: omega = nab.connection_form
sage: check = []
sage: for i in E.irange():
....: for j in E.irange():
....: check.append(Omega(i,j,e) == omega(i,j,e).exterior_derivative() +
....: sum(omega(k,j,e).wedge(omega(i,k,e))
....: for k in E.irange()))
sage: check # long time
[True, True, True, True]
```

```python
add_connection_form(i, j, frame=None)
```

Return the connection form $\omega^j_i$ in a given frame for assignment.

See method `connection_forms()` for details about the definition of the connection forms.

To delete the connection forms in other frames, use the method `set_connection_form()` instead.

**INPUT:**

- $i, j$ – indices identifying the 1-form $\omega^j_i$
- `frame` – (default: None) local frame in which the connection 1-form is defined; if None, the default frame of the vector bundle is assumed.

**Warning:** If the connection has already forms in other frames, it is the user’s responsibility to make sure that the 1-forms to be added are consistent with them.

**OUTPUT:**

- connection 1-form $\omega^j_i$ in the given frame, as an instance of the class `DiffForm`; if such connection 1-form did not exist previously, it is created. See method `connection_forms()` for the storage convention of the connection 1-forms.

**EXAMPLES:**
```python
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e')  # standard frame for E
sage: nab = E.bundle_connection('nabla', latex_name=r'\nabla')
sage: nab.add_connection_form(0, 1, frame=e)[:] = [x^2, x]
sage: nab[e, 0, 1].display()
connection (0,1) of bundle connection nabla w.r.t. Local frame
(E|_M, (e_0,e_1)) = x^2 dx + x dy
```

Since \( e \) is the vector bundle's default local frame, its mention may be omitted:

```python
sage: nab.add_connection_form(1, 0)[:] = [y^2, y]
sage: nab[1, 0].display()
connection (1,0) of bundle connection nabla w.r.t. Local frame
(E|_M, (e_0,e_1)) = y^2 dx + y dy
```

Adding connection 1-forms w.r.t. to another local frame:

```python
sage: f = E.local_frame('f')
sage: nab.add_connection_form(1, 1, frame=f)[:] = [x, y]
sage: nab[f, 1, 1].display()
connection (1,1) of bundle connection nabla w.r.t. Local frame
(E|_M, (f_0,f_1)) = x dx + y dy
```

The forms w.r.t. the frame \( e \) have been kept:

```python
sage: nab[e, 0, 1].display()
connection (0,1) of bundle connection nabla w.r.t. Local frame
(E|_M, (e_0,e_1)) = x^2 dx + x dy
```

To delete them, use the method `set_connection_form()` instead.

`connection_form(i, j, frame=None)`

Return the connection 1-form corresponding to the given index and local frame.

See also:
Consult `connection_forms()` for detailed information.

INPUT:

- \( i, j \) – indices identifying the 1-form \( \omega^j_i \)
- \( frame \) – (default: None) local frame relative to which the connection 1-forms are defined; if None, the default frame of the vector bundle’s corresponding section module is assumed.

OUTPUT:

- the 1-form \( \omega^j_i \), as an instance of `DiffForm`

EXAMPLES:

```python
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e')  # standard frame for E
sage: nab = E.bundle_connection('nabla', latex_name=r'\nabla')
sage: nab.set_connection_form(0, 1)[:] = [x^2, x]
sage: nab.set_connection_form(1, 0)[:] = [y^2, y]
```

(continues on next page)
connection_forms (frame=None)

Return the connection forms relative to the given frame.

If $e$ is a local frame on $E$, we have

$$\nabla e_i = \sum_{j=1}^{n} e_j \otimes \omega^j_i,$$

and the corresponding $n \times n$-matrix $(\omega^j_i)_{i,j}$ consisting of one forms is called connection matrix of $\nabla$ with respect to $e$.

If the connection coefficients are not known already, they are computed from the above formula.

INPUT:

• frame – (default: None) local frame relative to which the connection forms are required; if none is provided, the vector bundle’s default frame is assumed

OUTPUT:

• connection forms relative to the frame frame, as a dictionary with tuples $(i,j)$ as key and one forms as instances of diff_form as value representing the matrix entries.

EXAMPLES:

Connection forms of a bundle connection on a rank 2 vector bundle over a 3-dimensional manifold:

```
sage: M = Manifold(3, 'M', start_index=1)
sage: c_xyz.<x,y,z> = M.chart()
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e')
sage: nab = E.bundle_connection('nabla', r"nabla")
sage: nab[:] = 0 # initialize curvature forms
sage: forms = nab.connection_forms()
sage: [forms[k] for k in sorted(forms)]
```

```
[1-form connection (1,1) of bundle connection nabla w.r.t. Local frame (E|_M, (e_1,e_2)) on the 3-dimensional differentiable manifold M,
1-form connection (1,2) of bundle connection nabla w.r.t. Local frame (E|_M, (e_1,e_2)) on the 3-dimensional differentiable manifold M,
1-form connection (2,1) of bundle connection nabla w.r.t. Local frame (E|_M, (e_1,e_2)) on the 3-dimensional differentiable manifold M,
1-form connection (2,2) of bundle connection nabla w.r.t. Local frame (E|_M, (e_1,e_2)) on the 3-dimensional differentiable manifold M]
```
- name – name given to the copy
- latex_name – (default: None) LaTeX symbol to denote the copy; if none is provided, the LaTeX symbol is set to name

**Note:** The name and the derived quantities are not copied.

**EXAMPLES:**

```python
sage: M = Manifold(3, 'M', start_index=1)
sage: X.<x,y,z> = M.chart()
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e')
sage: nab = E.bundle_connection('nabla')
sage: nab.set_connection_form(1, 1)[:] = [x^2, x-z, y^3]
sage: nab.set_connection_form(1, 2)[:] = [1, x, z*y^3]
sage: nab.set_connection_form(2, 1)[:] = [1, 2, 3]
sage: nab.set_connection_form(2, 2)[:] = [0, 0, 0]
sage: nab.display()
connection (1,1) of bundle connection nabla w.r.t. Local frame (E|_M, (e_1,e_2)) = x^2 dx + (x - z) dy + y^3 dz
connection (1,2) of bundle connection nabla w.r.t. Local frame (E|_M, (e_1,e_2)) = dx + x dy + y^3*z dz
connection (2,1) of bundle connection nabla w.r.t. Local frame (E|_M, (e_1,e_2)) = dx + 2 dy + 3 dz
sage: nab_copy = nab.copy('nablo'); nab_copy
Bundle connection nablo on the Differentiable real vector bundle E -> M of rank 2 over the base space 3-dimensional differentiable manifold M
sage: nab is nab_copy
False
sage: nab == nab_copy
True
sage: nab_copy.display()
connection (1,1) of bundle connection nablo w.r.t. Local frame (E|_M, (e_1,e_2)) = x^2 dx + (x - z) dy + y^3 dz
connection (1,2) of bundle connection nablo w.r.t. Local frame (E|_M, (e_1,e_2)) = dx + x dy + y^3*z dz
connection (2,1) of bundle connection nablo w.r.t. Local frame (E|_M, (e_1,e_2)) = dx + 2 dy + 3 dz
```

**curvature_form(i, j, frame=None)**

Return the curvature 2-form corresponding to the given index and local frame.

The **curvature 2-forms** with respect to the frame $e$ are the 2-forms $\Omega^j_i$ given by the formula

$$\Omega^j_i = d\omega^j_i + \sum_{k=1}^n \omega^j_k \wedge \omega^k_i$$

**INPUT:**

- $i, j$ – indices identifying the 2-form $\Omega^j_i$
- $frame$ – (default: None) local frame relative to which the curvature 2-forms are defined; if None, the default frame of the vector bundle is assumed.

**OUTPUT:**

- the 2-form $\Omega^j_i$, as an instance of `DiffForm`
EXAMPLES:

```
sage: M = Manifold(2, 'M', start_index=1)
sage: X.<x,y> = M.chart()
sage: E = M.vector_bundle(1, 'E')
sage: nab = E.bundle_connection('nabla', latex_name=r'\nabla')
sage: e = E.local_frame('e')
sage: nab.set_connection_form(1, 1)[:] = [x^2, x]
sage: curv = nab.curvature_form(1, 1); curv
2-form curvature (1,1) of bundle connection nabla w.r.t. Local frame (E|_M, (e_1)) on the 2-dimensional differentiable manifold M
sage: curv.display()
curvature (1,1) of bundle connection nabla w.r.t. Local frame (E|_M, (e_1)) = dx∧dy
```

**del_other_forms** *(frame=None)*

Delete all the connection forms but those corresponding to frame.

**INPUT:**

- **frame** — (default: None) local frame, the connection forms w.r.t. which are to be kept; if None, the default frame of the vector bundle is assumed.

**EXAMPLES:**

We first create two sets of connection forms:

```
sage: M = Manifold(2, 'M', start_index=1)
sage: X.<x,y> = M.chart()
sage: E = M.vector_bundle(2, 'E')
sage: nab = E.bundle_connection('nabla', latex_name=r'\nabla')
sage: e = E.local_frame('e')
sage: nab.set_connection_form(1, 1, frame=e)[:]= [x^2, x]
sage: f = E.local_frame('f')
sage: nab.add_connection_form(1, 1, frame=f)[:]= [y^2, y]
sage: nab[e, 1, 1].display()
connection (1,1) of bundle connection nabla w.r.t. Local frame (E|_M, (e_1,e_2)) = x^2 dx + x dy
sage: nab[f, 1, 1].display()
connection (1,1) of bundle connection nabla w.r.t. Local frame (E|_M, (f_1,f_2)) = y^2 dx + y dy
```

Let us delete the connection forms w.r.t. all frames except for frame e:

```
sage: nab.del_other_forms(e)
sage: nab[e, 1, 1].display()
connection (1,1) of bundle connection nabla w.r.t. Local frame (E|_M, (e_1,e_2)) = x^2 dx + x dy
```

The connection forms w.r.t. frame e have indeed been deleted:

```
sage: nab[f, :]
Traceback (most recent call last):
...
ValueError: no basis could be found for computing the components in the Local frame (E|_M, (e_1,e_2))
```

**display** *(frame=None, vector_frame=None, chart=None, only_nonzero=True)*

Display all the connection 1-forms w.r.t. to a given local frame, one per line.
The output is either text-formatted (console mode) or LaTeX-formatted (notebook mode).

INPUT:

- `frame` (default: None) local frame of the vector bundle relative to which the connection 1-forms are defined; if `None`, the default frame of the bundle is used

- `vector_frame` (default: None) vector frame of the manifold relative to which the connection 1-forms should be displayed; if `None`, the default frame of the local frame’s domain is used

- `chart` (default: None) chart specifying the coordinate expression of the connection 1-forms; if `None`, the default chart of the domain of `frame` is used

- `only_nonzero` (default: True) boolean; if True, only nonzero connection coefficients are displayed

EXAMPLES:

Set connection 1-forms:

```
sage: M = Manifold(3, 'M', start_index =1)
sage: X.<x,y,z> = M.chart()
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e') # standard frame for E
sage: nab = E.bundle_connection('nabla', latex_name=r'
abla'); nab
Bundle connection nabla on the Differentiable real vector bundle E -> M of rank 2 over the base space 3-dimensional differentiable manifold M
sage: nab[:] = 0
sage: nab[1, 1][:] = [x, y, z]
sage: nab[2, 2][:] = [x^2, y^2, z^2]
```

By default, only the nonzero connection coefficients are displayed:

```
sage: nab.display()
connection (1,1) of bundle connection nabla w.r.t. Local frame (E|_M, (e_1,e_2)) = x dx + y dy + z dz
connection (2,2) of bundle connection nabla w.r.t. Local frame (E|_M, (e_1,e_2)) = x^2 dx + y^2 dy + z^2 dz
```

By default, the displayed connection 1-forms are those w.r.t. the default frame of the vector bundle. The aforementioned is therefore equivalent to:

```
sage: nab.display(frame=E.default_frame())
connection (1,1) of bundle connection nabla w.r.t. Local frame (E|_M, (e_1,e_2)) = x dx + y dy + z dz
connection (2,2) of bundle connection nabla w.r.t. Local frame (E|_M, (e_1,e_2)) = x^2 dx + y^2 dy + z^2 dz
```

Moreover, the connection 1-forms are displayed w.r.t. the default vector frame on the local frame’s domain, i.e.:

```
sage: domain = e.domain()
sage: nab.display(vector_frame=domain.default_frame())
connection (1,1) of bundle connection nabla w.r.t. Local frame (E|_M, (e_1,e_2)) = x dx + y dy + z dz
```

(continues on next page)
connection (2,2) of bundle connection nabla w.r.t. Local frame 
\( (E|_M, (e_1,e_2)) = x^2 dx + y^2 dy + z^2 dz \)

By default, the parameter only_nonzero is set to True. Otherwise, the connection 1-forms being zero are shown as well:

```
sage: nab.display(only_nonzero=False)
```

connection (1,1) of bundle connection nabla w.r.t. Local frame 
\( (E|_M, (e_1,e_2)) = x dx + y dy + z dz \)

connection (1,2) of bundle connection nabla w.r.t. Local frame 
\( (E|_M, (e_1,e_2)) = 0 \)

connection (2,1) of bundle connection nabla w.r.t. Local frame 
\( (E|_M, (e_1,e_2)) = 0 \)

connection (2,2) of bundle connection nabla w.r.t. Local frame 
\( (E|_M, (e_1,e_2)) = x^2 dx + y^2 dy + z^2 dz \)

**set_connection_form** \((i, j, \text{frame}=\text{None})\)

Return the connection form \(\omega^j_i\) in a given frame for assignment.

See method **connection_forms()** for details about the definition of the connection forms.

The connection forms with respect to other frames are deleted, in order to avoid any inconsistency. To keep them, use the method **add_connection_form()** instead.

**INPUT:**

- \(i, j\) – indices identifying the 1-form \(\omega^j_i\)
- \(\text{frame}\) – (default: None) local frame in which the connection 1-form is defined; if None, the default frame of the vector bundle is assumed.

**OUTPUT:**

- connection 1-form \(\omega^j_i\) in the given frame, as an instance of the class **DiffForm**; if such connection 1-form did not exist previously, it is created. See method **connection_forms()** for the storage convention of the connection 1-forms.

**EXAMPLES:**

Setting the connection forms of a bundle connection w.r.t. some local frame:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e') # standard frame for E
sage: nab = E.bundle_connection('nabla', latex_name=r'\nabla')
sage: nab.set_connection_form(0, 1)[:]= [x^2, x]
sage: nab[0, 1].display()
connection (0,1) of bundle connection nabla w.r.t. Local frame 
\( (E|_M, (e_0,e_1)) = x^2 dx + x dy \)
```

Since \(e\) is the vector bundle’s default local frame, its mention may be omitted:

```
sage: nab.set_connection_form(1, 0)[:]= [y^2, y]
sage: nab[1, 0].display()
connection (1,0) of bundle connection nabla w.r.t. Local frame 
\( (E|_M, (e_0,e_1)) = y^2 dx + y dy \)
```

Setting connection 1-forms w.r.t. to another local frame:
The forms w.r.t. the frame \( e \) have been deleted:

```python
sage: nab\[e, 0, 1\].display()
Traceback (most recent call last):
  ...
ValueError: no basis could be found for computing the components in
the Local frame (E|_M, (f_0,f_1))
```

To keep them, use the method `add_connection_form()` instead.

```python
set_immutable()
```

Set `self` and all restrictions of `self` immutable.

**EXAMPLES:**

An affine connection can be set immutable:

```python
sage: M = Manifold(3, 'M', start_index=1)
sage: X.<x,y,z> = M.chart()
sage: E = M.vector_bundle(2, 'E')
sage: e = E.local_frame('e')
sage: nab = E.bundle_connection('nabla')
sage: nab.set_connection_form(1, 1)[:] = [x^2, x -z, y^3]
sage: nab.set_connection_form(1, 2)[:] = [1, x, z*y^3]
sage: nab.set_connection_form(2, 1)[:] = [1, 2, 3]
sage: nab.set_connection_form(2, 2)[:] = [0, 0, 0]
sage: nab.is_immutable()
False
sage: nab.set_immutable()
```

The coefficients of immutable elements cannot be changed:

```python
sage: f = E.local_frame('f')
sage: nab.add_connection_form(1, 1, frame=f)[:] = [x, y, z]
Traceback (most recent call last):
  ...
ValueError: object is immutable; please change a copy instead
```

```python
vector_bundle()
```

Return the vector bundle on which the bundle connection is defined.

**OUTPUT:**

- instance of class `DifferentiableVectorBundle` representing the vector bundle on which `self` is defined.

**EXAMPLES:**

```python
sage: M = Manifold(3, 'M', start_index=1)
sage: c_xyz.<x,y,z> = M.chart()
sage: E = M.vector_bundle(2, 'E')
```
sage: nab = E.bundle_connection('nabla', r'nabla')
sage: nab.vector_bundle()
Differentiable real vector bundle $E \to M$ of rank 2 over the base space 3-dimensional differentiable manifold $M$

2.15.3 Characteristic cohomology classes

A characteristic class $\kappa$ is a natural transformation that associates with each vector bundle $E \to M$ a cohomology class $\kappa(E) \in H^*(M; R)$ such that for any continuous map $f : N \to M$ from another topological manifold $N$, the naturality condition is satisfied:

$$f^* \kappa(E) = \kappa(f^* E) \in H^*(N; R)$$

The cohomology class $\kappa(E)$ is called characteristic cohomology class. Roughly speaking, characteristic cohomology classes measure the non-triviality of vector bundles.

One way to obtain and compute characteristic classes in the de Rham cohomology with coefficients in the ring $\mathbb{C}$ is via the so-called Chern-Weil theory using the curvature of a differentiable vector bundle.

For that let $\nabla$ be a connection on $E$, $e$ a local frame on $E$ and $\Omega$ be the corresponding curvature matrix (see: curvature_form()).

Namely, if $P : \text{Mat}_{n\times n}(\mathbb{C}) \to \mathbb{C}$ is an invariant polynomial, the object

$$[P(\Omega)] \in H_{\text{dir}}^2(M, \mathbb{C})$$

is well-defined, independent of the choice of $\nabla$ (the proof can be found in [Roe1988] pp. 31) and fulfills the naturality condition. This is the foundation of the Chern-Weil theory and the following definitions.

---

Note: This documentation is rich of examples, but sparse in explanations. Please consult the references for more details.

AUTHORS:
- Michael Jung (2019): initial version
- Michael Jung (2021): new algorithm; complete refactoring

REFERENCES:
- [Mil1974]
- [Roe1988]

Contents

We consider the following three types of classes:

- **Additive Classes**
- **Multiplicative Classes**
- **Pfaffian Classes**
Additive Classes

In the complex case, let $f$ be a holomorphic function around zero. Then we call

$$\left[ \operatorname{tr} \left( f \left( \frac{\Omega}{2\pi i} \right) \right) \right] \in H^2_{\text{dr}}(M, \mathbb{C})$$

the additive characteristic class associated to $f$ of the complex vector bundle $E$.

Important and predefined additive classes are:

- **Chern Character** with $f(x) = \exp(x)$

In the real case, let $g$ be a holomorphic function around zero with $g(0) = 0$. Then we call

$$\left[ \operatorname{tr} \left( \frac{1}{2} g \left( - \frac{\Omega^2}{4\pi^2} \right) \right) \right] \in H^4_{\text{dr}}(M, \mathbb{C})$$

the additive characteristic class associated to $g$ of the real vector bundle $E$.

**EXAMPLES:**

Consider the Chern character on some 2-dimensional spacetime:

```python
sage: M = Manifold(2, 'M', structure='Lorentzian')
sage: X.<t,x> = M.chart()
sage: E = M.vector_bundle(1, 'E', field='complex'); E
Differentiable complex vector bundle $E \to M$ of rank 1 over the base space
2-dimensional Lorentzian manifold $M$
sage: e = E.local_frame('e')

Let us define the connection $\nabla^E$ in terms of an electro-magnetic potential $A(t)$:

```python
sage: nab = E.bundle_connection('nabla^E', latex_name=r'\nabla^E')
sage: omega = M.one_form(name='omega')
sage: A = function('A')
sage: nab.set_connection_form(0, 0)[1] = I*A(t)
sage: nab[0, 0].display()
connection (0,0) of bundle connection nabla^E w.r.t. Local frame
(E|_M, (e_0)) = I*A(t) dx
sage: nab.set_immutable()
```

The Chern character is then given by:

```python
sage: ch = E.characteristic_cohomology_class('ChernChar'); ch
Characteristic cohomology class $c(E)$ of the Differentiable complex vector
bundle $E \to M$ of rank 1 over the base space 2-dimensional Lorentzian
manifold $M$
```

The corresponding characteristic form w.r.t. the bundle connection can be obtained via `get_form()`:

```python
sage: ch_form = ch.get_form(nab); ch_form.display_expansion()
$c(E, \nabla^E) = 1 + 1/2*d(A)/dt/pi dt∧dx$
```
**Multiplicative Classes**

In the **complex** case, let $f$ be a holomorphic function around zero. Then we call

$$\left[ \det \left( f \left( \frac{\Omega}{2\pi i} \right) \right) \right] \in H^{2*}_{\text{dR}}(M, \mathbb{C})$$

the **multiplicative characteristic class** associated to $f$ of the complex vector bundle $E$.

Important and predefined multiplicative classes on complex vector bundles are:

- **Chern class** with $f(x) = 1 + x$
- **Todd class** with $f(x) = \frac{x}{1 - \exp(-x)}$

In the **real** case, let $g$ be a holomorphic function around zero with $g(0) = 1$. Then we call

$$\left[ \det \left( \sqrt{g \left( -\frac{\Omega^2}{4\pi^2} \right)} \right) \right] \in H^{4*}_{\text{dR}}(M, \mathbb{C})$$

the **multiplicative characteristic class** associated to $g$ on the **real** vector bundle $E$.

Important and predefined multiplicative classes on real vector bundles are:

- **Pontryagin class** with $g(x) = 1 + x$
- **$\hat{A}$ class** with $g(x) = \frac{\sqrt{x/2}}{\sinh(\sqrt{x}/2)}$
- **Hirzebruch class** with $g(x) = \sqrt{x} \tanh(\sqrt{x})$

**EXAMPLES:**

We consider the **Chern class** of the tautological line bundle $\gamma^1$ over $\mathbb{CP}^1$:

```python
sage: M = Manifold(2, 'CP^1', start_index=1)
sage: U = M.open_subset('U')
sage: c_cart.<x,y> = U.chart()  # homogeneous coordinates in real terms
sage: c_comp.<z, zbar> = U.chart(rz:z zbar: \bar{z})  # complexification
sage: cart_to_comp = c_cart.transition_map(c_comp, (x +I*y, x -I*y))
```

Now, the Chern class can be constructed:

```python
sage: nab = E.bundle_connection('nabla')
sage: omega = U.one_form(name='omega')
```

To apply the Chern-Weil approach, we need a bundle connection in terms of a connection one form. To achieve this, we take the connection induced from the hermitian metric on the trivial bundle $\mathbb{C}^2 \times \mathbb{CP}^1 \supset \gamma^1$. In this the frame $e$ corresponds to the section $[z : 1] \mapsto (z, 1)$ and its magnitude-squared is given by $1 + |z|^2$:

```python
sage: nab[e, i, 1] = omega
sage: nab.set_immutable()
```

Now, the Chern class can be constructed:

```python
sage: c = E.characteristic_cohomology_class('Chern'); c
```

Characteristic cohomology class $c(\gamma^1)$ of the Differentiable complex vector bundle $\gamma^1 \to \mathbb{CP}^1$ of rank 1 over the base space 2-dimensional differentiable manifold $\mathbb{CP}^1$

```python
sage: c_form = c.get_form(nab)
```
Since $U$ and $\mathbb{CP}^1$ differ only by a point and therefore a null set, it is enough to integrate the top form over the domain $U$:

\[ c(\gamma^1, \nabla) = 1 + 1/2*I/(\pi + \pi*z^2*zbar^2 + 2*\pi*z*zbar) \; dz\wedge dzbar \]

The result shows that $c_1(\gamma^1)$ generates the second integer cohomology of $\mathbb{CP}^1$.

**Pfaffian Classes**

Usually, there is no such thing as “Pfaffian classes” in literature. However, using the matrix’ Pfaffian and inspired by the aforementioned definitions, such classes can be defined as follows.

Let $E$ be a real vector bundle of rank $2n$ and $f$ an odd real function being analytic at zero. Furthermore, let $\Omega$ be skew-symmetric, which certainly will be true if $\nabla$ is metric and $e$ is orthonormal. Then we call

\[ \left[ \text{Pf} \left( f \left( \frac{\Omega}{2\pi} \right) \right) \right] \in H^{2n*}(M, \mathbb{R}) \]

the Pfaffian class associated to $f$.

The most important Pfaffian class is the Euler class which is simply given by $f(x) = x$.

**EXAMPLES:**

We consider the Euler class of $S^2$:

```
sage: M.<x,y> = manifolds.Sphere(2, coordinates='stereographic')
sage: TM = M.tangent_bundle()
sage: e_class = TM.characteristic_cohomology_class('Euler'); e_class
Characteristic cohomology class e(TS^2) of the Tangent bundle TS^2 over the 2-sphere S^2 of radius 1 smoothly embedded in the Euclidean space E^3
```

To compute a particular representative of the Euler class, we need to determine a connection, which is in this case given by the standard metric:

```
sage: g = M.metric('g') # standard metric on S2, long time
sage: nab = g.connection() # long time
sage: nab.set_immutable() # long time
```

Now the representative of the Euler class with respect to the connection $\nabla_g$ induced by the standard metric can be computed:

```
sage: e_class_form = e_class.get_form(nab) # long time
sage: e_class_form.display_expansion() # long time
e(TS^2, nabla_g) = 2/(\pi + \pi*x^4 + \pi*y^4 + 2*\pi*x^2 + 2*(\pi + \pi*x^2)*y^2) \; dx\wedge dy
```

Let us check whether this form represents the Euler class correctly:

```
sage: # long time
sage: expr = e_class_form[2][[1,2]].expr()
sage: expr = integrate(expr, x, -infinity, infinity)
```

(continues on next page)
As we can see, the integral coincides with the Euler characteristic of $S^2$ so that our form actually represents the Euler class appropriately.

```python
sage: sage.manifolds.differentiable.characteristic_cohomology_class.Algorithm_generic
Bases: SageObject

Abstract algorithm class to compute the characteristic forms of the generators.

EXAMPLES:
```

```python
sage: from sage.manifolds.differentiable.characteristic_cohomology_class import...
Algorithm_generic
sage: algorithm = Algorithm_generic()
sage: algorithm.get
Cached version of <function Algorithm_generic.get at 0x...>
sage: algorithm.get_local
Traceback (most recent call last):
...
NotImplementedError: <abstract method get_local at 0x...>
sage: algorithm.get_gen_pow
Cached version of <function Algorithm_generic.get_gen_pow at 0x...>
```

```python
get (nab)
Return the global characteristic forms of the generators w.r.t a given connection.

The result is cached.

OUTPUT:
• a list containing the generator's global characteristic forms as instances of `sage.manifolds.differentiable.diff_form.DiffForm`

EXAMPLES:
```
```python
sage: M = manifolds.EuclideanSpace(4)
sage: TM = M.tangent_bundle()
sage: g = M.metric()
sage: nab = g.connection()
sage: nab.set_immutable()
sage: p = TM.characteristic_cohomology_class('Pontryagin')
sage: p_form = p.get_form(nab); p_form # long time
Mixed differential form p(TE^4, nabla_g) on the 4-dimensional Euclidean space E^4
sage: p_form.display_expansion() # long time
p(TE^4, nabla_g) = 1
```

```python
get_gen_pow (nab, i, n)
Return the n-th power of the i-th generator's characteristic form w.r.t nab.

The result is cached.

EXAMPLES:
```
```python
sage: M = manifolds.EuclideanSpace(8)
sage: TM = M.tangent_bundle()
sage: g = M.metric()
sage: nab = g.connection()
sage: nab.set_immutable()

sage: A = TM.characteristic_cohomology_class('AHat')
sage: A_form = A.get_form(nab); A_form  # long time
Mixed differential form A^8(TE^8, nabla_g) on the 8-dimensional Euclidean space E^8
sage: A_form.display_expansion()  # long time
A^8(TE^8, nabla_g) = 1
```

### get_local (cmat)

Abstract method to get the local forms of the generators w.r.t. a given curvature form matrix `cmat`.

**OUTPUT:**

- a list containing the generator’s local characteristic forms

**ALGORITHM:**

The inherited class determines the algorithm.

**EXAMPLES:**

4-dimensional Euclidean space:

```python
sage: M = manifolds.EuclideanSpace(4)
sage: TM = M.tangent_bundle()
sage: g = M.metric()
sage: nab = g.connection()
sage: e = M.frames()[0]  # select standard frame
sage: cmat = [ [nab.curvature_form(i, j, e)  # long time
...: for j in TM.irange()]
...: for i in TM.irange()]
```

Import the algorithm:

```python
sage: from sage.manifolds.differentiable.characteristic_cohomology_class import PontryaginAlgorithm
sage: algorithm = PontryaginAlgorithm()
sage: [p1] = algorithm.get_local(cmat)  # long time
sage: p1.display()  # long time
0
```

A concrete implementation is given by a `sage.misc.fast_methods.Singleton`:

```python
sage: algorithm is PontryaginAlgorithm()
True
```

### class `sage.manifolds.differentiable.characteristic_cohomology_class.CharacteristicCohomologyClass

**Bases:** `FiniteGCAbstract

Characteristic cohomology class ring.

Let $E \rightarrow M$ be a real or complex vector bundle of rank $k$ and $R$ be a torsion-free subring of $C$. 

830 Chapter 2. Differentiable Manifolds
Let $BG$ be the classifying space of the group $G$. As for vector bundles, we consider

- $G = O(k)$ if $E$ is real,
- $G = SO(k)$ if $E$ is real and oriented,
- $G = U(k)$ if $E$ is complex.

The cohomology ring $H^*(BG; R)$ can be explicitly expressed for the aforementioned cases:

$$H^*(BG; R) \cong \begin{cases} R[c_1, \ldots, c_k] & \text{if } G = U(k), \\ R[p_1, \ldots, p_{\lfloor k/2 \rfloor}] & \text{if } G = O(k), \\ R[p_1, \ldots, p_k, e] / (p_k - e^2) & \text{if } G = SO(2k), \\ R[p_1, \ldots, p_k, e] & \text{if } G = SO(2k + 1). \end{cases}$$

The Chern-Weil homomorphism relates the generators in the de Rham cohomology as follows. If $\Omega$ is a curvature form matrix on $E$, for the Chern classes we get

$$\det \left( 1 + \frac{t\Omega}{2\pi i} \right) = 1 + \sum_{n=1}^{k} c_n(E)t^n,$$

for the Pontryagin classes we have

$$\det \left( 1 - \frac{t\Omega}{2\pi} \right) = 1 + \sum_{n=1}^{\lfloor k/2 \rfloor} p_n(E)t^n,$$

and for the Euler class we obtain

$$\text{Pf} \left( \frac{\Omega}{2\pi} \right) = e(E).$$

Consequently, the cohomology ring $H^*(BG; R)$ is mapped (not necessarily injectively) to a subring of $H^*_{dR}(M, C)$ via the Chern-Weil homomorphism. This implementation attempts to represent this subring.

**Note:** Some generators might have torsion in $H^*(BG; R)$ giving a zero element in the de Rham cohomology. Those generators are still considered in the implementation. Generators whose degree exceed the dimension of the base space, however, are ignored.

**INPUT:**

- `base` – base ring
- `vbundle` – vector bundle

**EXAMPLES:**

Characteristic cohomology class ring over the tangent bundle of an 8-dimensional manifold:

```sage
M = Manifold(8, 'M')
sage: TM = M.tangent_bundle()
sage: CR = TM.characteristic_cohomology_class_ring(); CR
Algebra of characteristic cohomology classes of the Tangent bundle TM over the 8-dimensional differentiable manifold M
sage: CR.gens()
(Characteristic cohomology class (p_1)(TM) of the Tangent bundle TM over the 8-dimensional differentiable manifold M, Characteristic cohomology class (p_2)(TM) of the Tangent bundle TM over the 8-dimensional differentiable manifold M)
```
The default base ring is $\mathbb{Q}$:

```python
sage: CR.base_ring()
Rational Field
```

Characteristic cohomology class ring over a complex vector bundle:

```python
sage: M = Manifold(4, 'M')
sage: E = M.vector_bundle(2, 'E', field='complex')
sage: CR_E = E.characteristic_cohomology_class_ring(); CR_E
Algebra of characteristic cohomology classes of the Differentiable complex vector bundle $E \to M$ of rank 2 over the base space 4-dimensional differentiable manifold $M$
sage: CR_E.gens()
(Characteristic cohomology class $(c_1)(E)$ of the Differentiable complex vector bundle $E \to M$ of rank 2 over the base space 4-dimensional differentiable manifold $M$,
Characteristic cohomology class $(c_2)(E)$ of the Differentiable complex vector bundle $E \to M$ of rank 2 over the base space 4-dimensional differentiable manifold $M$)
```

Characteristic cohomology class ring over an oriented manifold:

```python
sage: S2 = manifolds.Sphere(2, coordinates='stereographic')
sage: TS2 = S2.tangent_bundle()
sage: S2.has_orientation()
True
sage: CR = TS2.characteristic_cohomology_class_ring()
sage: CR.gens()
(Characteristic cohomology class $(e)(TS^2)$ of the Tangent bundle $TS^2$ over the 2-sphere $S^2$ of radius 1 smoothly embedded in the Euclidean space $E^3$)
```

**Element**

alias of `CharacteristicCohomologyClassRingElement`

class `sage.manifolds.differentiable.characteristic_cohomology_class.CharacteristicCohomologyClassRingElement`

Bases: `IndexedFreeModuleElement`

Characteristic cohomology class.

Let $E \to M$ be a real/complex vector bundle of rank $k$. A characteristic cohomology class of $E$ is generated by either

- Chern classes if $E$ is complex,
- Pontryagin classes if $E$ is real,
- Pontryagin classes and the Euler class if $E$ is real and orientable,

via the Chern-Weil homomorphism.

For details about the ring structure, see `CharacteristicCohomologyClassRing`.

To construct a characteristic cohomology class, please use `CharacteristicCohomologyClass()`.
EXAMPLES:

```python
sage: M = Manifold(12, 'M')
sage: TM = M.tangent_bundle()
sage: CR = TM.characteristic_cohomology_class_ring()
sage: p1, p2, p3 = CR.gens()
sage: p1*p2+p3
Characteristic cohomology class (p_1⌣p_2 + p_3)(TM) of the Tangent
bundle TM over the 12-dimensional differentiable manifold M
```

```python
sage: A = TM.characteristic_cohomology_class(AHat); A
Characteristic cohomology class A^(TM) of the Tangent bundle TM over
the 12-dimensional differentiable manifold M
```

```python
sage: A == 1 - p1/24 + (7*p1^2-4*p2)/5760 + (44*p1*p2-31*p1^3-16*p3)/967680
True
```

```python
get_form(nab)
```

Return the characteristic form of self.

INPUT:

- nab – get the characteristic form w.r.t. to the connection nab

OUTPUT:

- an instance of `sage.manifolds.differentiable.mixed_form.MixedForm`

EXAMPLES:

Trivial characteristic form on Euclidean space:

```python
sage: M = manifolds.EuclideanSpace(4)
sage: TM = M.tangent_bundle()
sage: one = TM.characteristic_cohomology_class_ring().one()
sage: g = M.metric()
sage: nab = g.connection()
sage: one.get_form(nab)
Mixed differential form one on the 4-dimensional Euclidean space E^4
```

Pontryagin form on the 4-sphere:

```python
sage: M = manifolds.Sphere(4)
sage: TM = M.tangent_bundle()
sage: p = TM.characteristic_cohomology_class('Pontryagin'); p
Characteristic cohomology class p(TS^4) of the Tangent bundle TS^4
over the 4-sphere S^4 of radius 1 smoothly embedded in the
5-dimensional Euclidean space E^5
```

```python
sage: g = M.metric() # long time
sage: nab = g.connection() # long time
sage: nab.set Immutable() # long time
sage: p_form = p.get_form(nab); p_form # long time
Mixed differential form p(TS^4, nabla_g) on the 4-sphere S^4 of
radius 1 smoothly embedded in the 5-dimensional Euclidean space E^5
```

```python
representative(nab=None)
```

Return any representative of self.

INPUT:
• `nab` – (default: None) if stated, return the representative w.r.t. the connection `nab`; otherwise an arbitrary already computed representative will be chosen.

OUTPUT:
• an instance of `sage.manifolds.differentiable.mixed_form.MixedForm`

EXAMPLES:

Define the 4-dimensional Euclidean space:

```python
sage: M = manifolds.EuclideanSpace(4)
sage: TM = M.tangent_bundle()
sage: one = TM.characteristic_cohomology_class_ring().one()
```

No characteristic form has been computed so far, thus we get an error:

```python
sage: one.representative()
Traceback (most recent call last):
  ...: AttributeError: cannot pick a representative
```

Get a characteristic form:

```python
sage: g = M.metric()
sage: nab = g.connection()
sage: nab.set_immutable()
sage: one.get_form(nab)
Mixed differential form one on the 4-dimensional Euclidean space E^4
```

Now, the result is cached and `representative` returns a form:

```python
sage: one.representative()
Mixed differential form one on the 4-dimensional Euclidean space E^4
```

Alternatively, the option `nab` can be used to return the characteristic form w.r.t. a fixed connection:

```python
sage: one.representative(nab)
Mixed differential form one on the 4-dimensional Euclidean space E^4
```

See also:

`CharacteristicCohomologyClassRingElement.get_form()`

`sage.manifolds.differentiable.mixed_form.MixedForm.set_name(name=None, latex_name=None)`

Set (or change) the text name and LaTeX name of the characteristic class.

INPUT:

• `name` – (string; default: None) name given to the characteristic cohomology class

• `latex_name` – (string; default: None) LaTeX symbol to denote the characteristic cohomology class; if None while name is provided, the LaTeX symbol is set to name

EXAMPLES:

```python
sage: M = Manifold(8, 'M')
sage: TM = M.tangent_bundle()
sage: x = var('x')
sage: k = TM.characteristic_cohomology_class(1+x^2, class_type='multiplicative'); k
```

(continues on next page)
Characteristic cohomology class \((1 + p_1^2 - 2p_2)(TM)\) of the Tangent bundle TM over the 8-dimensional differentiable manifold M

\[
sage: \kappa.set\_name(name='k', latex\_name=r'\kappa')
\]

Characteristic cohomology class \(k(TM)\) of the Tangent bundle TM over the 8-dimensional differentiable manifold M

\[
sage: \kappa
\]

Characteristic cohomology class \(k(TM)\) of the Tangent bundle TM over the 8-dimensional differentiable manifold M

\[
sage: latex(\kappa)
\]

\[
\kappa\left(TM\right)
\]

\[\text{class} \]

\[\text{sage.manifolds.differentiable.characteristic_cohomology_class.ChernAlgorithm} \]

\[\text{Bases: Singleton, Algorithm\_generic} \]

Algorithm class to generate Chern forms.

\[\text{EXAMPLES:} \]

Define a complex line bundle over a 2-dimensional manifold:

\[
sage: M = \text{Manifold}(2, 'M', structure='Lorentzian')
\]

\[
sage: X.<t,x> = M.chart()
\]

\[
sage: E = M.vector\_bundle(1, 'E', field='complex'); E
\]

Differentiable complex vector bundle \(E \to M\) of rank 1 over the base space

\[2\text{-dimensional Lorentzian manifold M} \]

\[
sage: e = E.local\_frame('e')
\]

\[
sage: \omega = M.one\_form(name=\omega)
\]

\[
sage: A = \text{function}(A)
\]

\[
sage: \nab.set\_connection\_form(0, 0)[1] = I*\omega(t)
\]

\[
sage: \nab[0, 0].display()
\]

connection \((0,0)\) of bundle connection \(\nabla^E\) w.r.t. Local frame

\[
(E|_M, (e_0)) = I*\omega(t) \, dx
\]

\[
sage: \nab.set\_immutable()
\]

Import the algorithm and apply \(\nab\) to it:

\[
sage: from \text{sage.manifolds.differentiable.characteristic_cohomology_class import...}
\]

\[
\longrightarrow \text{ChernAlgorithm}
\]

\[
sage: \text{algorithm = ChernAlgorithm()}
\]

\[
sage: \text{algorithm.get}(\nab)
\]

\[2\text{-form on the 2-dimensional Lorentzian manifold M} \]

\[
sage: \text{algorithm.get}(\nab)[0].display()
\]

\[1/2*d(A)/dt/pi \, dt\wedge dx\]

Check some equalities:

\[
sage: cmat = [[\nab.nabla\_connection_form(0, 0, e)][1]]
\]

\[
sage: \text{algorithm.get}(\nab)[0] == \text{algorithm.get\_local}(\text{cmat})[0] \quad \# \text{bundle trivial}
\]

\[\text{True}\]

\[
sage: \text{algorithm.get\_gen\_pow}(\nab, 0, 1) == \text{algorithm.get}(\nab)[0]
\]

\[\text{True}\]

\[\text{get\_local}(\text{cmat}) \]

Return the local Chern forms w.r.t. a given curvature form matrix.

\[\text{OUTPUT:}\]
• a list containing the local characteristic Chern forms as instances of \texttt{sage.manifolds.differentiable.diff_form.DiffForm}

\textbf{ALGORITHM:}

The algorithm is based on the Faddeev-LeVerrier algorithm for the characteristic polynomial.

\textbf{EXAMPLES:}

Define a complex line bundle over a 2-dimensional manifold:

\begin{verbatim}
sage: M = Manifold(2, 'M', structure='Lorentzian')
sage: X.<t,x> = M.chart()
sage: E = M.vector_bundle(1, 'E', field='complex'); E
Differentiable complex vector bundle E -> M of rank 1 over the base
  space 2-dimensional Lorentzian manifold M
sage: e = E.local_frame('e')
sage: nab = E.bundle_connection('nabla^E', latex_name=r'\nabla^E')
sage: omega = M.one_form(name='omega')
sage: A = function('A')
sage: nab.set_connection_form(0, 0)[1] = I*A(t)
sage: nab[0, 0].display()
connection (0,0) of bundle connection nabla^E w.r.t. Local frame
  (E|_M, (e_0)) = I*A(t) dx
sage: cmat = [[nab.curvature_form(i, j, e) for j in E.irange()]
  for i in E.irange()]
\end{verbatim}

Import the algorithm and apply \texttt{cmat} to it:

\begin{verbatim}
sage: from sage.manifolds.differentiable.characteristic_cohomology_class import ChernAlgorithm
sage: algorithm = ChernAlgorithm()
sage: algorithm.get_local(cmat)
[2-form on the 2-dimensional Lorentzian manifold M]
\end{verbatim}

\textbf{class}

\texttt{sage.manifolds.differentiable.characteristic_cohomology_class.EulerAlgorithm}

Bases: \texttt{Singleton, Algorithm.generic}

Algorithm class to generate Euler forms.

\textbf{EXAMPLES:}

Consider the 2-dimensional Euclidean space:

\begin{verbatim}
sage: M = manifolds.EuclideanSpace(2)
sage: g = M.metric()
sage: nab = g.connection()
sage: nab.set_immutable()
\end{verbatim}

Import the algorithm and apply \texttt{nab} to it:

\begin{verbatim}
sage: from sage.manifolds.differentiable.characteristic_cohomology_class import EulerAlgorithm
sage: algorithm = EulerAlgorithm()
sage: algorithm.get(nab)
[2-form on the Euclidean plane E^2]
sage: algorithm.get(nab)[0].display()
0
\end{verbatim}
\textbf{get} (\textit{nab})

Return the global characteristic forms of the generators w.r.t. a given connection.

INPUT:
- a metric connection $\nabla$

OUTPUT:
- a list containing the global characteristic Euler form

ALGORITHM:
Assume that $\nabla$ is compatible with the metric $g$, and let $(s_1, \ldots, s_n)$ be any oriented frame. Denote by $G_s = (g(s_i, s_j))_{ij}$ the metric tensor and let $\Omega_s$ be the curvature form matrix of $\nabla$ w.r.t. $s$. Then, we get:

$$(G_s \cdot \Omega_s)_{ij} = g(R(\ldots) s_i, s_j),$$

where $R$ is the Riemannian curvature tensor w.r.t. $\nabla$.

The characteristic Euler form is now obtained by the expression

$$\frac{1}{\sqrt{|\det(G_s)|}} \text{Pf}\left(\frac{G_s \cdot \Omega_s}{2\pi}\right).$$

\textbf{EXAMPLES:}
Consider the 2-sphere:

```
sage: M.<x,y> = manifolds.Sphere(2, coordinates='stereographic')
sage: g = M.metric()  # long time
sage: nab = g.connection()  # long time
sage: nab.set_immutable()  # long time
```

Import the algorithm and apply \textit{nab} to it:

```
sage: from sage.manifolds.differentiable.characteristic_cohomology_class import EulerAlgorithm
```

```
sage: algorithm = EulerAlgorithm()
sage: algorithm.get(nab)  # long time
[2-form on the 2-sphere $S^2$ of radius 1 smoothly embedded in the Euclidean space $E^3$

$2/(\pi + \pi x^4 + \pi y^4 + 2\pi x^2 + 2(\pi + \pi x^2)*y^2) \ dx\wedge dy$
```

\textbf{REFERENCES:}
- [Che1944]
- [Baer2020]

\textbf{get\_local} (\textit{cmat})

Return the normalized Pfaffian w.r.t. a given curvature form matrix.

The normalization is given by the factor $\left(\frac{1}{2\pi}\right)^{\frac{k}{2}}$, where $k$ is the dimension of the curvature matrix.

OUTPUT:
- a list containing the normalized Pfaffian of a given curvature form
Note: In general, the output does not represent the local characteristic Euler form. The result is only guaranteed to be the local Euler form if \( \text{cmat} \) is given w.r.t. an orthonormal oriented frame. See \text{get()} for details.

**ALGORITHM:**

The algorithm is based on the Bär-Faddeev-LeVerrier algorithm for the Pfaffian.

**EXAMPLES:**

Consider the 2-sphere:

```sage
M.<th,phi> = manifolds.Sphere(2) # use spherical coordinates
t M = M.tangent_bundle()
 g = M.metric()
nab = g.connection()
 e = M.frames()[0] # select frame (opposite orientation!)
cmat = [[nab.curvature_form(i, j, e) for j in TM.irange()] for i in TM.irange()]
```

We need some preprocessing because the frame is not orthonormal:

```sage
gcmat = [[sum(g[[e, i, j]] * nab.curvature_form(j, k, e) for j in TM.irange()) for k in TM.irange()] for i in TM.irange()]
```

Now, \( gcmat \) is guaranteed to be skew-symmetric and can be applied to \text{get_local()}. Then, the result must be normalized:

```sage
from sage.manifolds.differentiable.characteristic_cohomology_class import EulerAlgorithm
euler = -algorithm.get_local(gcmat)[0] / sqrt(g.det(frame=e))
euler.display()
1/2*sin(th)/pi dth∧dphi
```

**PontryaginAlgorithm**

**EXAMPLES:**

5-dimensional Euclidean space:

```sage
M = manifolds.EuclideanSpace(5)
g = M.metric()
nab = g.connection()
nab.set_immutable()
from sage.manifolds.differentiable.characteristic_cohomology_class import PontryaginAlgorithm
algorithm = PontryaginAlgorithm()
euler = -algorithm.get_local(g) / sqrt(g.det(frame=e))
euler.display()
1/2*sin(th)/pi dth∧dphi
```

(continues on next page)
get_local \((cmat)\)

Return the local Pontryagin forms w.r.t. a given curvature form matrix.

OUTPUT:

- a list containing the local characteristic Pontryagin forms

ALGORITHM:

The algorithm is based on the Faddeev-LeVerrier algorithm for the characteristic polynomial.

EXAMPLES:

5-dimensional Euclidean space:

```python
sage: M = manifolds.EuclideanSpace(5)
sage: TM = M.tangent_bundle()
sage: g = M.metric()
sage: nab = g.connection()
sage: e = M.frames()[0]  # select standard frame
sage: cmat = [[nab.curvature_form(i, j, e) for j in TM.irange()] for i in TM.irange()]
```

Import the algorithm:

```python
sage: from sage.manifolds.differentiable.characteristic_cohomology_class import PontryaginAlgorithm
sage: algorithm = PontryaginAlgorithm()
sage: [p1] = algorithm.get_local(cmat)  # long time
sage: p1.display()  # long time
0
```

2.15. Differentiable Vector Bundles
sage: e_form, p1_form = algorithm.get(nab)  # long time
sage: e_form.display()  # long time
0
sage: p1_form.display()  # long time
0

get(nab)

Return the global characteristic forms of the generators w.r.t. a given connection.

OUTPUT:

- a list containing the global Euler form in the first entry, and the global Pontryagin forms in the remaining entries.

EXAMPLES:

4-dimensional Euclidean space:

sage: M = manifolds.EuclideanSpace(4)
sage: g = M.metric()
sage: nab = g.connection()
sage: nab.set_immutable()

Import the algorithm:

sage: from sage.manifolds.differentiable.characteristic_cohomology_class.
              → import PontryaginEulerAlgorithm
sage: algorithm = PontryaginEulerAlgorithm()
sage: algorithm.get(nab)  # long time
[4-form on the 4-dimensional Euclidean space E^4,
  4-form on the 4-dimensional Euclidean space E^4]

get_gen_pow(nab, i, n)

Return the $n$-th power of the $i$-th generator w.r.t nab.

The result is cached.

EXAMPLES:

4-dimensional Euclidean space:

sage: M = manifolds.EuclideanSpace(4)
sage: TM = M.tangent_bundle()
sage: g = M.metric()
sage: nab = g.connection()
sage: nab.set_immutable()

Import the algorithm:

sage: from sage.manifolds.differentiable.characteristic_cohomology_class.
              → import PontryaginEulerAlgorithm
sage: algorithm = PontryaginEulerAlgorithm()
sage: e = algorithm.get_gen_pow(nab, 0, 1)  # Euler, long time
sage: e.display()  # long time
0
sage: p1_pow2 = algorithm.get_gen_pow(nab, 1, 2)  # 1st Pontryagin squared,
              → long time
sage: p1_pow2
8-form zero on the 4-dimensional Euclidean space E^4
**get_local** \((\text{cmat})\)

Return the local Euler and Pontryagin forms w.r.t. a given curvature form matrix.

**Note:** Similar as for \texttt{EulerAlgorithm}, the first entry only represents the Euler form if the curvature form matrix is chosen w.r.t. an orthonormal oriented frame. See \texttt{EulerAlgorithm.get_local()} for details.

**OUTPUT:**

- a list containing the local Euler form in the first entry, and the local Pontryagin forms in the remaining entries.

**EXAMPLES:**

6-dimensional Euclidean space:

```python
sage: M = manifolds.EuclideanSpace(6)
sage: TM = M.tangent_bundle()
sage: g = M.metric()
sage: nab = g.connection()
sage: e = M.frames()[0]  # select the standard frame
sage: cmat = [ [nab.curvature_form(i, j, e)  # long time
... for j in TM.irange()]
... for i in TM.irange() ]
```

Import the algorithm:

```python
sage: from sage.manifolds.differentiable.characteristic_cohomology_class import PontryaginEulerAlgorithm
sage: algorithm = PontryaginEulerAlgorithm()
sage: e, p1 = algorithm.get_local(cmat)  # long time
sage: e.display()  # long time
```

```python
sage: p1.display()  # long time
```

sage.manifolds.differentiable.characteristic_cohomology_class.additive_sequence \((q, k, n=\text{None})\)

Turn the polynomial \(q\) into its additive sequence.

Let \(q\) be a polynomial and \(x_1, \ldots, x_n\) indeterminates. The **additive sequence of** \(q\) is then given by the polynomials \(Q_j\)

\[
\sum_{j=0}^{n} Q_j(\sigma_1, \ldots, \sigma_j)z^j = \sum_{i=1}^{k} q(zx_i),
\]

where \(\sigma_i\) is the \(i\)-th elementary symmetric polynomial in the indeterminates \(x_i\).

**INPUT:**

- \(q\) – polynomial to turn into its additive sequence.
- \(k\) – maximal index \(k\) of the sum
- \(n\) – (default: \texttt{None}) the highest order of the sequence \(n\); if \texttt{None}, the order of \(q\) is assumed.

**OUTPUT:**
• A symmetric polynomial representing the additive sequence.

EXAMPLES:

```python
sage: P.<x> = PolynomialRing(QQ)
sage: from sage.manifolds.differentiable.characteristic_cohomology_class import...  
                        additive_sequence
sage: f = 1 + x - x^2
sage: sym = additive_sequence(f, 2); sym
```

The maximal order of the result can be stated with \( n \):

```python
sage: sym_1 = additive_sequence(f, 2, 1); sym_1
2*e[] + e[1]
```

`sage.manifolds.differentiable.characteristic_cohomology_class.fast_wedge_power(form, n)`

Return the wedge product power of \( form \) using a square-and-wedge algorithm.

INPUT:

• \( form \) -- a differential form
• \( n \) -- a non-negative integer

EXAMPLES:

```python
sage: M = Manifold(4, 'M')
sage: X.<x,y,z,t> = M.chart()
sage: omega = M.diff_form(2, name='omega')
sage: omega[0,1] = t*y^2 + 2*x
sage: omega[0,3] = z - 2*y
sage: from sage.manifolds.differentiable.characteristic_cohomology_class import...  
                        fast_wedge_power
sage: fast_wedge_power(omega, 0)
Scalar field 1 on the 4-dimensional differentiable manifold M
sage: fast_wedge_power(omega, 1)
2-form omega on the 4-dimensional differentiable manifold M
sage: fast_wedge_power(omega, 2)
4-form omega∧omega on the 4-dimensional differentiable manifold M
```

`sage.manifolds.differentiable.characteristic_cohomology_class.multiplicative_sequence(q, n=None)`

Turn the polynomial \( q \) into its multiplicative sequence.

Let \( q \) be a polynomial and \( x_1, \ldots, x_n \) indeterminates. The multiplicative sequence of \( q \) is then given by the polynomials \( K_j \)

\[
\sum_{j=0}^{n} K_j(\sigma_1, \ldots, \sigma_j)z^j = \prod_{i=1}^{n} q(z x_i),
\]

where \( \sigma_i \) is the \( i \)-th elementary symmetric polynomial in the indeterminates \( x_i \).

INPUT:

• \( q \) -- polynomial to turn into its multiplicative sequence.
• \( n \) -- (default: None) the highest order \( n \) of the sequence; if None, the order of \( q \) is assumed.

OUTPUT:
• A symmetric polynomial representing the multiplicative sequence.

EXAMPLES:

```python
sage: P.<x> = PolynomialRing(QQ)
sage: from sage.manifolds.differentiable.characteristic_cohomology_class import multiplicative_sequence
sage: f = 1 + x - x^2
sage: sym = multiplicative_sequence(f); sym
```

The maximal order of the result can be stated with `n`:

```python
sage: sym_5 = multiplicative_sequence(f, n=5); sym_5
```
3.1 Pseudo-Riemannian Manifolds

A pseudo-Riemannian manifold is a pair \((M, g)\) where \(M\) is a real differentiable manifold \(M\) (see Differentiable-Manifold) and \(g\) is a field of non-degenerate symmetric bilinear forms on \(M\), which is called the metric tensor, or simply the metric (see PseudoRiemannianMetric).

Two important subcases are

- Riemannian manifold: the metric \(g\) is positive definite, i.e. its signature is \(n = \dim M\);
- Lorentzian manifold: the metric \(g\) has signature \(n - 2\) (positive convention) or \(2 - n\) (negative convention).

On a pseudo-Riemannian manifold, one may use various standard operators acting on scalar and tensor fields, like \(\text{grad()}\) or \(\text{div()}\).

All pseudo-Riemannian manifolds, whatever the metric signature, are implemented via the class PseudoRiemannianManifold.

Example 1: the sphere as a Riemannian manifold of dimension 2

We start by declaring \(S^2\) as a 2-dimensional Riemannian manifold:

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

We then cover \(S^2\) by two stereographic charts, from the North pole and from the South pole respectively:

```python
sage: U = M.open_subset('U')
sage: stereoN.<x,y> = U.chart()
sage: V = M.open_subset('V')
sage: stereoS.<u,v> = V.chart()
sage: M.declare_union(U,V)
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: W = U.intersection(V)
sage: stereoN_to_S.display()
```

Change of coordinates from Chart \((W, (x, y))\) to Chart \((W, (u, v))\):

```python
sage: stereoN_to_S.display()
u = x/(x^2 + y^2)
v = y/(x^2 + y^2)
sage: stereoN_to_S.inverse().display()
```

(continues on next page)
We get the metric defining the Riemannian structure by:

```python
sage: g = M.metric()
sage: g
Riemannian metric g on the 2-dimensional Riemannian manifold S^2
```

At this stage, the metric $g$ is defined as a Python object but there remains to initialize it by setting its components with respect to the vector frames associated with the stereographic coordinates. Let us begin with the frame of chart `stereoN`:

```python
sage: eU = stereoN.frame()
sage: g[eU, 0, 0] = 4/(1 + x^2 + y^2)^2
sage: g[eU, 1, 1] = 4/(1 + x^2 + y^2)^2
```

The metric components in the frame of chart `stereoS` are obtained by continuation of the expressions found in $W = U \cap V$ from the known change-of-coordinate formulas:

```python
sage: eV = stereoS.frame()
sage: g.add_comp_by_continuation(eV, W)
```

At this stage, the metric $g$ is well defined in all $S^2$:

```python
sage: g.display(eU)
g = 4/(x^2 + y^2 + 1)^2 dx⊗dx + 4/(x^2 + y^2 + 1)^2 dy⊗dy
sage: g.display(eV)
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
```

The expression in frame `eV` can be given a shape similar to that in frame `eU`, by factorizing the components:

```python
sage: g[eV, 0, 0].factor()
4/(u^2 + v^2 + 1)^2
sage: g[eV, 1, 1].factor()
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
```

Let us consider a scalar field $f$ on $S^2$:

```python
sage: f = M.scalar_field({stereoN: 1/(1+x^2+y^2)}, name='f')
sage: f.add_expr_by_continuation(stereoS, W)
sage: f.display()
f: S^2 → R
  on U: (x, y) ↦ 1/(x^2 + y^2 + 1)
  on V: (u, v) ↦ (u^2 + v^2)/(u^2 + v^2 + 1)
```

The gradient of $f$ (with respect to the metric $g$) is:

```python
sage: gradf = f.gradient()
sage: gradf
Vector field grad(f) on the 2-dimensional Riemannian manifold S^2
sage: gradf.display(eU)
grad(f) = -1/2*x ∂/∂x - 1/2*y ∂/∂y
sage: gradf.display(eV)
grad(f) = 1/2*u ∂/∂u + 1/2*v ∂/∂v
```
It is possible to write \( \text{grad}(f) \) instead of \( f\.\text{gradient}() \), by importing the standard differential operators of vector calculus:

\[
\text{sage: from sage.manifolds.operators import } * \\
\text{sage: grad}(f) == \text{gradf}
\]

The Laplacian of \( f \) (with respect to the metric \( g \)) is obtained either as \( f\.\text{laplacian}() \) or, thanks to the above import, as \( \text{laplacian}(f) \):

\[
\text{sage: } Df = \text{laplacian}(f) \\
\text{sage: } Df \\
\text{Scalar field Delta}(f) \text{ on the 2-dimensional Riemannian manifold } S^2 \\
\text{sage: } Df\.\text{display}() \\
\Delta(f): S^2 \to \mathbb{R} \\
on U: (x, y) \mapsto (x^2 + y^2 - 1)/(x^2 + y^2 + 1) \\
on V: (u, v) \mapsto -(u^2 + v^2 - 1)/(u^2 + v^2 + 1)
\]

Let us check the standard formula \( \Delta f = \text{div}(\text{grad } f) \):

\[
\text{sage: } Df == \text{div}(\text{gradf})
\]

Since each open subset of \( S^2 \) inherits the structure of a Riemannian manifold, we can get the metric on it via the method \( \text{metric}() \):

\[
\text{sage: } gU = U\.\text{metric}() \\
\text{sage: } gU \\
\text{Riemannian metric } g \text{ on the Open subset } U \text{ of the 2-dimensional Riemannian manifold } S^2 \\
\text{sage: } gU\.\text{display}() \\
g = 4/(x^2 + y^2 + 1)^2 \, dx\otimes dx + 4/(x^2 + y^2 + 1)^2 \, dy\otimes dy
\]

Of course, \( gU \) is nothing but the restriction of \( g \) to \( U \):

\[
\text{sage: } gU \text{ is } g\.\text{restrict}(U)
\]

**Example 2: Minkowski spacetime as a Lorentzian manifold of dimension 4**

We start by declaring a 4-dimensional Lorentzian manifold \( M \):

\[
\text{sage: } M = \text{Manifold}(4, 'M', \text{structure}='\text{Lorentzian}') \\
\text{sage: } M \\
4\text{-dimensional Lorentzian manifold } M
\]

We define Minkowskian coordinates on \( M \):

\[
\text{sage: } X.\langle t,x,y,z \rangle = M\.\text{chart}()
\]

We construct the metric tensor by:

\[
\text{sage: } g = M\.\text{metric}() \\
\text{sage: } g \\
\text{Lorentzian metric } g \text{ on the 4-dimensional Lorentzian manifold } M
\]

and initialize it to the Minkowskian value:
We may check that the metric is flat, i.e. has a vanishing Riemann curvature tensor:

```
sage: g.riemann().display()
Riem(g) = 0
```

A vector field on $M$:

```
sage: u = M.vector_field(name='u')
sage: u[0] = cosh(t)
sage: u[1] = sinh(t)
sage: u.display()
u = \cosh(t) \partial/\partial t + \sinh(t) \partial/\partial x
```

The scalar square of $u$ is:

```
sage: s = u.dot(u); s
Scalar field u.u on the 4-dimensional Lorentzian manifold M
```

Scalar products are taken with respect to the metric tensor:

```
sage: u.dot(u) == g(u,u)
True
```

$u$ is a unit timelike vector, i.e. its scalar square is identically $-1$:

```
sage: s.display()
u.u: M \to \mathbb{R}
  (t, x, y, z) \mapsto -1
sage: s.expr()
-1
```

Let us consider a unit spacelike vector:

```
sage: v = M.vector_field(name='v')
sage: v[0] = sinh(t)
sage: v[1] = cosh(t)
sage: v.display()
v = \sinh(t) \partial/\partial t + \cosh(t) \partial/\partial x
sage: v.dot(v).display()
v.v: M \to \mathbb{R}
  (t, x, y, z) \mapsto 1
sage: v.dot(v).expr()
1
```

$u$ and $v$ are orthogonal vectors with respect to Minkowski metric:

```
sage: u.dot(v).display()
u.v: M \to \mathbb{R}
```

(continues on next page)
The divergence of \( u \) is:

\[
\text{sage: } s = u.\text{div}(); s
\]

Scalar field \( \text{div}(u) \) on the 4-dimensional Lorentzian manifold \( M \)

\[
\text{sage: } s.\text{display()}
\]

\( \text{div}(u) : M \to \mathbb{R} \)

\[
(t, x, y, z) \mapsto \sinh(t)
\]

while its d’Alembertian is:

\[
\text{sage: } Du = u.\text{dalembertian}(); Du
\]

Vector field \( \Box(u) \) on the 4-dimensional Lorentzian manifold \( M \)

\[
\text{sage: } Du.\text{display()}
\]

\( \Box(u) = -\cosh(t) \frac{\partial}{\partial t} - \sinh(t) \frac{\partial}{\partial x} \)

AUTHORS:

• Eric Gourgoulhon (2018): initial version

REFERENCES:

• B. O’Neill: *Semi-Riemannian Geometry* [ONe1983]
• J. M. Lee: *Riemannian Manifolds* [Lee1997]
A pseudo-Riemannian manifold is a pair $(M, g)$ where $M$ is a real differentiable manifold $M$ (see DifferentiableManifold) and $g$ is a field of non-degenerate symmetric bilinear forms on $M$, which is called the metric tensor, or simply the metric (see PseudoRiemannianMetric).

Two important subcases are

- **Riemannian manifold**: the metric $g$ is positive definite, i.e. its signature is $n = \dim M$;
- **Lorentzian manifold**: the metric $g$ has signature $n - 2$ (positive convention) or $2 - n$ (negative convention).

**INPUT:**

- $n$ – positive integer; dimension of the manifold
- **name** – string; name (symbol) given to the manifold
- **metric_name** – (default: None) string; name (symbol) given to the metric; if None, 'g' is used
- **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)
- **base_manifold** – (default: None) if not None, must be a differentiable manifold; the created object is then an open subset of base_manifold
- **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
- **metric_latex_name** – (default: None) string; LaTeX symbol to denote the metric; if none is provided, it is set to metric_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(RR).Differentiable() (or Manifolds(RR).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 DifferentiableManifold and TopologicalManifold, would return the previously constructed object corresponding to these arguments).

**EXAMPLES:**

Pseudo-Riemannian manifolds are constructed via the generic function `Manifold()`, using the keyword `structure`:

```
sage: M = Manifold(4, 'M', structure='pseudo-Riemannian', signature=0)
sage: M
4-dimensional pseudo-Riemannian manifold M
sage: M.category()
Category of smooth manifolds over Real Field with 53 bits of precision
```

The metric associated with $M$ is:

```
sage: M.metric()
Pseudo-Riemannian metric g on the 4-dimensional pseudo-Riemannian manifold M
sage: M.metric().signature()
0
```
Its value has to be initialized either by setting its components in various vector frames (see the above examples regarding the 2-sphere and Minkowski spacetime) or by making it equal to a given field of symmetric bilinear forms (see the method `set` of the metric class). Both methods are also covered in the documentation of method `metric()` below.

The metric object belongs to the class `PseudoRiemannianMetric`:

```python
sage: isinstance(M_metric(), sage.manifolds.differentiable.metric....: PseudoRiemannianMetric)
True
```

See the documentation of this class for all operations available on metrics.

The default name of the metric is \(g\); it can be customized:

```python
sage: M = Manifold(4, 'M', structure='pseudo-Riemannian',....: metric_name='gam', metric_latex_name=r'\gamma')
sage: M_metric()
Riemannian metric \(\text{gam}\) on the 4-dimensional Riemannian manifold \(M\)
sage: latex(M_metric())
\(\gamma\)
```

A Riemannian manifold is constructed by the proper setting of the keyword `structure`:

```python
sage: M = Manifold(4, 'M', structure='Riemannian'); M
4-dimensional Riemannian manifold \(M\)
sage: M_metric()
Riemannian metric \(g\) on the 4-dimensional Riemannian manifold \(M\)
sage: M_metric().signature()
4
```

Similarly, a Lorentzian manifold is obtained by:

```python
sage: M = Manifold(4, 'M', structure='Lorentzian'); M
4-dimensional Lorentzian manifold \(M\)
sage: M_metric()
Lorentzian metric \(g\) on the 4-dimensional Lorentzian manifold \(M\)
```

The default Lorentzian signature is taken to be positive:

```python
sage: M_metric().signature()
2
```

but one can opt for the negative convention via the keyword `signature`:

```python
sage: M = Manifold(4, 'M', structure='Lorentzian', signature='negative')
sage: M_metric()
Lorentzian metric \(g\) on the 4-dimensional Lorentzian manifold \(M\)
sage: M_metric().signature()
-2
```

Return the metric giving the pseudo-Riemannian structure to the manifold, or define a new metric tensor on the manifold.
INPUT:

- **name** – (default: None) name given to the metric; if name is None or matches the name of the metric defining the pseudo-Riemannian structure of self, the latter metric is returned

- **signature** – (default: None; ignored if name is 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; ignored if name is None) LaTeX symbol to denote the metric; if None, it is formed from name

- **dest_map** – (default: None; ignored if name is None) instance of class DiffMap representing the destination map $\Phi : U \to M$, where $U$ is the current manifold; if None, the identity map is assumed (case of a metric tensor field on $U$)

OUTPUT:

- instance of `PseudoRiemannianMetric`

EXAMPLES:

Metric of a 3-dimensional Riemannian manifold:

```python
sage: M = Manifold(3, 'M', structure='Riemannian', start_index=1)
sage: X.<x,y,z> = M.chart()
sage: g = M.metric(); g
Riemannian metric g on the 3-dimensional Riemannian manifold M
```

The metric remains to be initialized, for instance by setting its components in the coordinate frame associated to the chart $X$:

```python
sage: g[1,1], g[2,2], g[3,3] = 1, 1, 1
sage: g.display()
g = dx⊗dx + dy⊗dy + dz⊗dz
```

Alternatively, the metric can be initialized from a given field of nondegenerate symmetric bilinear forms; we may create the former object by:

```python
sage: X.coframe()
Coordinate coframe (M, (dx,dy,dz))
sage: dx, dy, dz = X.coframe()[1], X.coframe()[2], X.coframe()[3]
sage: b = dx*dx + dy*dy + dz*dz
sage: b
Field of symmetric bilinear forms dx⊗dx+dy⊗dy+dz⊗dz on the
3-dimensional Riemannian manifold M
```

We then use the metric method `set()` to make $g$ being equal to $b$ as a symmetric tensor field of type $(0,2)$:

```python
sage: g.set(b)
sage: g.display()
g = dx⊗dx + dy⊗dy + dz⊗dz
```

Another metric can be defined on $M$ by specifying a metric name distinct from that chosen at the creation of the manifold (which is $g$ by default, but can be changed thanks to the keyword `metric_name` in `Manifold()`):

```python
sage: h = M.metric('h'); h
Riemannian metric h on the 3-dimensional Riemannian manifold M
```
The metric tensor \( h \) is distinct from the metric entering in the definition of the Riemannian manifold \( M \):

\[
\text{sage: } h \text{ \textit{is} } M.\text{metric()}
\]

\[
\text{False}
\]

while we have of course:

\[
\text{sage: } g \text{ \textit{is} } M.\text{metric()}
\]

\[
\text{True}
\]

Providing the same name as the manifold’s default metric returns the latter:

\[
\text{sage: } M.\text{metric('g')} \text{ \textit{is} } M.\text{metric()}
\]

\[
\text{True}
\]

In the present case (\( M \) is diffeomorphic to \( \mathbb{R}^3 \)), we can even create a Lorentzian metric on \( M \):

\[
\text{sage: } h = M.\text{metric('h', signature=1)}; h
\]

Lorentzian metric \( h \) on the 3-dimensional Riemannian manifold \( M \)

---

3.1. Pseudo-Riemannian Manifolds
We initialize the metric of $M$:

```sage
sage: g = M.metric()
sage: g[0,0], g[1,1] = 1, 1
```

Then the metric on $U$ is determined as the restriction of $g$ to $U$:

```sage
sage: gU = U.metric(); gU
Riemannian metric $g$ on the Open subset $U$ of the 2-dimensional Riemannian manifold $M$
sage: gU.display()
g = dx⊗dx + dy⊗dy
sage: gU is g.restrict(U)
True
```

**volume_form** *(contra=0)*

Volume form (Levi-Civita tensor) $\epsilon$ associated with self.

This assumes that self is an orientable manifold, with a preferred orientation; see orientation() for details.

The volume form $\epsilon$ is an $n$-form ($n$ being the manifold's dimension) such that, for any vector frame $(e_i)$ that is orthonormal with respect to the metric of the pseudo-Riemannian manifold self,

$$\epsilon(e_1, \ldots, e_n) = \pm 1$$

There are only two such $n$-forms, which are opposite of each other. The volume form $\epsilon$ is selected as the one that returns +1 for any right-handed vector frame with respect to the chosen orientation of self.

**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, or of the subclass MultivectorField if $k = n$

**EXAMPLES:**

Volume form of the Euclidean 3-space:

```sage
sage: M = Manifold(3, 'M', structure='Riemannian', start_index=1)
sage: X.<x,y,z> = M.chart()
sage: g = M.metric()
sage: g[1,1], g[2,2], g[3,3] = 1, 1, 1
sage: eps = M.volume_form(); eps
3-form eps_g on the 3-dimensional Riemannian manifold M
sage: eps.display()
eps_g = dx∧dy∧dz
sage: eps1 = M.volume_form(1); eps1
Tensor field of type (1,2) on the 3-dimensional Riemannian manifold M
sage: eps1.display()
```

Raising the first index:

```sage
```

(continues on next page)
\[
\frac{\partial}{\partial x} \otimes dy \otimes dz - \frac{\partial}{\partial x} \otimes dz \otimes dy - \frac{\partial}{\partial y} \otimes dx \otimes dz + \frac{\partial}{\partial y} \otimes dz \otimes dx + \frac{\partial}{\partial z} \otimes dx \otimes dy - \frac{\partial}{\partial z} \otimes dy \otimes dx \\
\text{sage: } \text{eps1.symmetries()}
\]

no symmetry; antisymmetry: (1, 2)

Raising the first and second indices:

\[
\text{sage: eps2 = M.volume_form(2); eps2}
\]

Tensor field of type (2,1) on the 3-dimensional Riemannian manifold M

\[
\text{sage: eps2.display()}
\]

\[
\frac{\partial}{\partial x} \otimes \frac{\partial}{\partial y} \otimes dz - \frac{\partial}{\partial x} \otimes \frac{\partial}{\partial z} \otimes dy - \frac{\partial}{\partial y} \otimes \frac{\partial}{\partial x} \otimes dz + \frac{\partial}{\partial y} \otimes \frac{\partial}{\partial z} \otimes dx + \frac{\partial}{\partial z} \otimes \frac{\partial}{\partial x} \otimes dy - \frac{\partial}{\partial z} \otimes \frac{\partial}{\partial y} \otimes dx
\]

\[
\text{sage: eps2.symmetries()}
\]

no symmetry; antisymmetry: (0, 1)

Fully contravariant version:

\[
\text{sage: eps3 = M.volume_form(3); eps3}
\]

3-vector field on the 3-dimensional Riemannian manifold M

\[
\text{sage: eps3.display()}
\]

\[
\frac{\partial}{\partial x} \wedge \frac{\partial}{\partial y} \wedge \frac{\partial}{\partial z}
\]

3.2 Euclidean Spaces and Vector Calculus

3.2.1 Euclidean Spaces

An **Euclidean space of dimension** \(n\) is an affine space \(E\), whose associated vector space is a \(n\)-dimensional vector space over \(\mathbb{R}\) and is equipped with a positive definite symmetric bilinear form, called the **scalar product** or **dot product** [Ber1987]. An Euclidean space of dimension \(n\) can also be viewed as a Riemannian manifold that is diffeomorphic to \(\mathbb{R}^n\) and that has a flat metric \(g\). The Euclidean scalar product is then that defined by the Riemannian metric \(g\).

The current implementation of Euclidean spaces is based on the second point of view. This allows for the introduction of various coordinate systems in addition to the usual Cartesian systems. Standard curvilinear systems (planar, spherical and cylindrical coordinates) are predefined for 2-dimensional and 3-dimensional Euclidean spaces, along with the corresponding transition maps between them. Another benefit of such an implementation is the direct use of methods for vector calculus already implemented at the level of Riemannian manifolds (see, e.g., the methods `cross_product()` and `curl()`, as well as the module `operators`).

Euclidean spaces are implemented via the following classes:

- `EuclideanSpace` for generic values \(n\),
- `EuclideanPlane` for \(n = 2\),
- `Euclidean3dimSpace` for \(n = 3\).

The user interface is provided by `EuclideanSpace`.
Example 1: the Euclidean plane

We start by declaring the Euclidean plane E, with \((x, \ y)\) as Cartesian coordinates:

```sage
sage: E.<x,y> = EuclideanSpace()
sage: E
Euclidean plane E^2
sage: dim(E)
2
```

E is automatically endowed with the chart of Cartesian coordinates:

```sage
sage: E.atlas()
[Chart (E^2, (x, y))]
sage: cartesian = E.default_chart(); cartesian
Chart (E^2, (x, y))
```

Thanks to the use of \(<x, y>\) when declaring E, the coordinates \((x, y)\) have been injected in the global namespace, i.e. the Python variables \(x\) and \(y\) have been created and are available to form symbolic expressions:

```sage
sage: y
y
sage: type(y)
<class sage.symbolic.expression.Expression>
sage: assumptions()
[x is real, y is real]
```

The metric tensor of E is predefined:

```sage
sage: g = E.metric(); g
Riemannian metric g on the Euclidean plane E^2
sage: g.display()
g = dx⊗dx + dy⊗dy
sage: g[:]
[1 0]
[0 1]
```

It is a flat metric, i.e. it has a vanishing Riemann tensor:

```sage
sage: g.riemann()
Tensor field Riem(g) of type (1,3) on the Euclidean plane E^2
sage: g.riemann().display()
Riem(g) = 0
```

Polar coordinates \((r, \phi)\) are introduced by:

```sage
sage: polar.<r,ph> = E.polar_coordinates()
sage: polar
Chart (E^2, (r, ph))
```

E is now endowed with two coordinate charts:

```sage
sage: E.atlas()
[Chart (E^2, (x, y)), Chart (E^2, (r, ph))]
```

The ranges of the coordinates introduced so far are:
The transition map from polar coordinates to Cartesian ones is:

```
sage: E.coord_change(polar, cartesian).display()
x = r*cos(ph)
y = r*sin(ph)
```

while the reverse one is:

```
sage: E.coord_change(cartesian, polar).display()
r = sqrt(x^2 + y^2)
ph = arctan2(y, x)
```

A point of $E$ is constructed from its coordinates (by default in the Cartesian chart):

```
sage: p = E((-1,1), name='p'); p
Point p on the Euclidean plane $E^2$
sage: p.parent()
Euclidean plane $E^2$
```

The coordinates of a point are obtained by letting the corresponding chart act on it:

```
sage: cartesian(p)
(-1, 1)
sage: polar(p)
(sqrt(2), 3/4*pi)
```

At this stage, $E$ is endowed with three vector frames:

```
sage: E.frames()
[Coordinate frame $(E^2, (e_x,e_y))$, Coordinate frame $(E^2, (∂/∂r,∂/∂ph))$, Vector frame $(E^2, (e_r,e_ph))$]
```

The third one is the standard orthonormal frame associated with polar coordinates, as we can check from the metric components in it:

```
sage: polar_frame = E.polar_frame(); polar_frame
Vector frame $(E^2, (e_r,e_ph))$
sage: g[polar_frame,:]
[1 0]
[0 1]
```

The expression of the metric tensor in terms of polar coordinates is:

```
sage: g.display(polar)
g = dr⊗dr + r^2 dph⊗dph
```

A vector field on $E$:

```
sage: v = E.vector_field(-y, x, name='v'); v
Vector field v on the Euclidean plane $E^2$
sage: v.display()
```

(continues on next page)
\[ v = -y \mathbf{e}_x + x \mathbf{e}_y \]

By default, the components of \( v \), as returned by display or the bracket operator, refer to the Cartesian frame on \( E \); to get the components with respect to the orthonormal polar frame, one has to specify it explicitly, generally along with the polar chart for the coordinate expression of the components:

\[
\text{sage: } v[\text{polar}\
\text{sage: } v[\text{polar}\
\text{[0, r]}
\]

Note that the default frame for the display of vector fields can be changed thanks to the method \textit{set\_default\_frame}(); in the same vein, the default coordinates can be changed via the method \textit{set\_default\_chart}():

\[
\text{sage: } E.\text{set\_default\_frame}(\text{polar}\_\text{frame})
\text{sage: } E.\text{set\_default\_chart}(\text{polar})
\text{sage: } v.\text{display}()
\text{sage: } v[:]
\text{[0, r]}
\text{sage: } E.\text{set\_default\_frame}(E.\text{cartesian}\_\text{frame}()) \# \text{revert to Cartesian frame}
\text{sage: } E.\text{set\_default\_chart}(\text{cartesian}) \# \text{and chart}
\]

When defining a vector field from components relative to a vector frame different from the default one, the vector frame has to be specified explicitly:

\[
\text{sage: } v = E.\text{vector\_field}(1, 0, \text{frame}=\text{polar}\_\text{frame})
\text{sage: } v.\text{display}(\text{polar}\_\text{frame})
\text{e\_r}
\text{sage: } v.\text{display}()
\text{x/sqrt(x^2 + y^2) e\_x + y/sqrt(x^2 + y^2) e\_y}
\]

The argument \textit{chart} must be used to specify in which coordinate chart the components are expressed:

\[
\text{sage: } v = E.\text{vector\_field}(0, r, \text{frame}=\text{polar}\_\text{frame}, \text{chart}=\text{polar})
\text{sage: } v.\text{display}(\text{polar}\_\text{frame}, \text{polar})
\text{r e\_ph}
\text{sage: } v.\text{display}()
\text{-y e\_x + x e\_y}
\]

It is also possible to pass the components as a dictionary, with a pair (vector frame, chart) as a key:

\[
\text{sage: } v = E.\text{vector\_field}(((\text{polar}\_\text{frame}, \text{polar})\text{): } (0, r)))
\text{sage: } v.\text{display}(\text{polar}\_\text{frame}, \text{polar})
\text{r e\_ph}
\]

The key can be reduced to the vector frame if the chart is the default one:

\[
\text{sage: } v = E.\text{vector\_field}((\text{polar}\_\text{frame}: \{0, 1\}))
\text{sage: } v.\text{display}(\text{polar}\_\text{frame})
\text{e\_ph}
\]

Finally, it is possible to construct the vector field without initializing any component:
The components can then be set in a second stage, via the square bracket operator, the unset components being assumed to be zero:

```
sage: v[1] = -y
sage: v.display()  # v[2] is zero
-y e_x
sage: v[2] = x
sage: v.display()
-y e_x + x e_y
```

The above is equivalent to:

```
sage: v[:] = -y, x
sage: v.display()
-y e_x + x e_y
```

The square bracket operator can also be used to set components in a vector frame that is not the default one:

```
sage: v = E.vector_field(name='v')
sage: v[polar_frame, 2, polar] = r
sage: v.display(polar_frame, polar)
```

The value of the vector field \( v \) at point \( p \):

```
sage: vp = v.at(p); vp
Vector v at Point p on the Euclidean plane E^2
sage: vp.display()
v = -e_x - e_y
sage: vp.display(polar_frame.at(p))
v = sqrt(2) e_ph
```

A scalar field on \( E \):

```
sage: f = E.scalar_field(x*y, name='f'); f
Scalar field f on the Euclidean plane E^2
sage: f.display()
f: E^2 → R
(x, y) ↦ x*y
(r, ph) ↦ r^2*cos(ph)*sin(ph)
```

The value of \( f \) at point \( p \):

```
sage: f(p)
-1
```

The gradient of \( f \):

```
sage: from sage.manifolds.operators import *  # to get grad, div, etc.
sage: w = grad(f); w
Vector field grad(f) on the Euclidean plane E^2
sage: w.display()
```

(continues on next page)
\[
\text{grad}(f) = y \ e_x + x \ e_y
\]

\[\text{sage: } w\text{.display(polar\_frame, polar)}\]

\[
\text{grad}(f) = 2r\cos(ph)\sin(ph) \ e_r + (2\cos(ph)^2 - 1)\cos(ph)\ e_ph
\]

The dot product of two vector fields:

\[\text{sage: } s = v\text{.dot}(w); s\]

Scalar field \(v \cdot \text{grad}(f)\) on the Euclidean plane \(E^2\)

\[\text{sage: } s\text{.display()}; s\]

\[
\begin{align*}
    \text{grad}(f): & E^2 \to \mathbb{R} \\
    (x, y) & \mapsto x^2 - y^2 \\
    (r, ph) & \mapsto (2\cos(ph)^2 - 1)\cos(ph)
\end{align*}
\]

\[\text{sage: } s\text{.expr()}\]

\[
    x^2 - y^2
\]

The norm is related to the dot product by the standard formula:

\[\text{sage: } \text{norm}(v)^2 == v\text{.dot}(v)\]

True

The divergence of the vector field \(v\):

\[\text{sage: } s = \text{div}(v); s\]

Scalar field \(\text{div}(v)\) on the Euclidean plane \(E^2\)

\[\text{sage: } s\text{.display()}; s\]

\[
\begin{align*}
    \text{div}(v): & E^2 \to \mathbb{R} \\
    (x, y) & \mapsto 0 \\
    (r, ph) & \mapsto 0
\end{align*}
\]

Example 2: Vector calculus in the Euclidean 3-space

We start by declaring the 3-dimensional Euclidean space \(E\), with \((x, y, z)\) as Cartesian coordinates:

\[\text{sage: } E.<x,y,z> = \text{EuclideanSpace()}\]

Euclidean space \(E^3\)

A simple vector field on \(E\):

\[\text{sage: } v = E\text{.vector\_field}(-y, x, 0, \text{name}='v')\]

\[\text{sage: } v\text{.display()}; v\]

\[
[-y, x, 0]
\]

The Euclidean norm of \(v\):

\[\text{sage: } s = \text{norm}(v); s\]

Scalar field \(|v|\) on the Euclidean space \(E^3\)

\[\text{sage: } s\text{.display()}; s\]

\[
\begin{align*}
    |v|: & E^3 \to \mathbb{R} \\
    (x, y, z) & \mapsto \sqrt{x^2 + y^2}
\end{align*}
\]

The divergence of \(v\) is zero:
Sage: from sage.manifolds.operators import *
Sage: div(v)
Scalar field div(v) on the Euclidean space E^3
Sage: div(v).display()
div(v): E^3 → ℝ
  (x, y, z) ↦ 0

while its curl is a constant vector field along $\mathbf{e}_z$:

Sage: w = curl(v); w
Vector field curl(v) on the Euclidean space E^3
Sage: w.display()
curl(v) = 2 \mathbf{e}_z

The gradient of a scalar field:

Sage: f = E.scalar_field(sin(x*y*z), name='f')
Sage: u = grad(f); u
Vector field grad(f) on the Euclidean space E^3
Sage: u.display()
grad(f) = y*z*cos(x*y*z) \mathbf{e}_x + x*z*cos(x*y*z) \mathbf{e}_y + x*y*cos(x*y*z) \mathbf{e}_z

The curl of a gradient is zero:

Sage: curl(u).display()
curl(grad(f)) = 0

The dot product of two vector fields:

Sage: s = u.dot(v); s
Scalar field grad(f).v on the Euclidean space E^3
Sage: s.expr()
(x^2 - y^2)*z*cos(x*y*z)

The cross product of two vector fields:

Sage: a = u.cross(v); a
Vector field grad(f) x v on the Euclidean space E^3
Sage: a.display()
grad(f) x v = -x^2*y*z*cos(x*y*z) \mathbf{e}_x - x*y^2*z*cos(x*y*z) \mathbf{e}_y + 2*x*y*z*cos(x*y*z) \mathbf{e}_z

The scalar triple product of three vector fields:

Sage: triple_product = E.scalar_triple_product()
Sage: s = triple_product(u, v, w); s
Scalar field epsilon(grad(f),v,curl(v)) on the Euclidean space E^3
Sage: s.expr()
4*x*y*z*cos(x*y*z)

Let us check that the scalar triple product of $u$, $v$ and $w$ is $u \cdot (v \times w)$:

Sage: s == u.dot(v.cross(w))
True

AUTHORS:

- Eric Gourgoulhon (2018): initial version
REFERENCES:

- M. Berger: Geometry I [Ber1987]

```python
class sage.manifolds.differentiable.examples.euclidean.Euclidean3dimSpace(name=None,
latex_name=None,
coordinates='Cartesian',
symbols=None,
metric_name='g',
metric_latex_name=None,
start_index=1,
base_manifold=None,
category=None,
unique_tag=None)
```

Bases: EuclideanSpace

3-dimensional Euclidean space.

A 3-dimensional Euclidean space is an affine space $E$, whose associated vector space is a 3-dimensional vector space over $\mathbb{R}$ and is equipped with a positive definite symmetric bilinear form, called the scalar product or dot product.

The class `Euclidean3dimSpace` inherits from `PseudoRiemannianManifold` (via `EuclideanSpace`) since a 3-dimensional Euclidean space can be viewed as a Riemannian manifold that is diffeomorphic to $\mathbb{R}^3$ and that has a flat metric $g$. The Euclidean scalar product is the one defined by the Riemannian metric $g$.

INPUT:

- `name` – (default: None) string; name (symbol) given to the Euclidean 3-space; if None, the name will be set to 'E^3'
- `latex_name` – (default: None) string; LaTeX symbol to denote the Euclidean 3-space; if None, it is set to '\mathbb{E}^3' if name is None and to name otherwise
- `coordinates` – (default: 'Cartesian') string describing the type of coordinates to be initialized at the Euclidean 3-space creation; allowed values are 'Cartesian' (see `cartesian_coordinates()`), 'spherical' (see `spherical_coordinates()`) and 'cylindrical' (see `cylindrical_coordinates()`)
- `symbols` – (default: None) string defining the coordinate text symbols and LaTeX symbols, with the same conventions as the argument `coordinates` in `RealDiffChart`, namely `symbols` is a string of coordinate fields separated by a blank space, where each field contains the coordinate’s text symbol and possibly the coordinate’s LaTeX symbol (when the latter is different from the text symbol), both symbols being separated by a colon (:); if None, the symbols will be automatically generated according to the value of `coordinates`
• **metric_name** – (default: 'g') string; name (symbol) given to the Euclidean metric tensor

• **metric_latex_name** – (default: None) string; LaTeX symbol to denote the Euclidean metric tensor; if none is provided, it is set to metric_name

• **start_index** – (default: 1) integer; lower value of the range of indices used for “indexed objects” in the Euclidean 3-space, e.g. coordinates of a chart

• **base_manifold** – (default: None) if not None, must be an Euclidean 3-space; the created object is then an open subset of base_manifold

• **category** – (default: None) to specify the category; if None, Manifolds(RR).Smooth() & MetricSpaces().Complete() is assumed

• **names** – (default: None) unused argument, except if symbols is not provided; it must then be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator <,> is used)

• **init_coord_methods** – (default: None) dictionary of methods to initialize the various type of coordinates, with each key being a string describing the type of coordinates; to be used by derived classes only

• **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 PseudoRiemannianManifold would return the previously constructed object corresponding to these arguments)

**EXAMPLES:**

A 3-dimensional Euclidean space:

```python
sage: E = EuclideanSpace(3); E
Euclidean space E^3
sage: latex(E)
\mathbb{E}^{3}
```

E belongs to the class `Euclidean3dimSpace` (actually to a dynamically generated subclass of it via SageMath’s category framework):

```python
sage: type(E)
<class 'sage.manifolds.differentiable.examples.euclidean.Euclidean3dimSpace_with_category'>
```

E is both a real smooth manifold of dimension 3 and a complete metric space:

```python
sage: E.category()
Join of Category of smooth manifolds over Real Field with 53 bits of precision and Category of connected manifolds over Real Field with 53 bits of precision and Category of complete metric spaces
sage: dim(E)
3
```

It is endowed with a default coordinate chart, which is that of Cartesian coordinates \((x, y, z)\):

```python
sage: E.atlas()
[Chart (E^3, (x, y, z))]
sage: E.default_chart()
Chart (E^3, (x, y, z))
sage: cartesian = E.cartesian_coordinates()
sage: cartesian == E.default_chart()
True
```

A point of E:
Manifolds, Release 10.3

```python
sage: p = E((3,-2,1)); p
Point on the Euclidean space E^3
sage: cartesian(p)
(3, -2, 1)
sage: p in E
True
sage: p.parent() is E
True
```

E is endowed with a default metric tensor, which defines the Euclidean scalar product:

```python
sage: g = E.metric(); g
Riemannian metric g on the Euclidean space E^3
sage: g.display()
g = dx⊗dx + dy⊗dy + dz⊗dz
```

Curvilinear coordinates can be introduced on \( E \): see `spherical_coordinates()` and `cylindrical_coordinates()`.

See also:

*Example 2: Vector calculus in the Euclidean 3-space*

`cartesian_coordinates` *(symbols=None, names=None)*

Return the chart of Cartesian coordinates, possibly creating it if it does not already exist.

**INPUT:**

- `symbols` – (default: None) string defining the coordinate text symbols and LaTeX symbols, with the same conventions as the argument `coordinates` in `RealDiffChart`; this is used only if the Cartesian chart has not been already defined; if None the symbols are generated as \((x, y, z)\).

- `names` – (default: None) unused argument, except if `symbols` is not provided; it must be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator `<,>` is used)

**OUTPUT:**

- the chart of Cartesian coordinates, as an instance of `RealDiffChart`

**EXAMPLES:**

```python
sage: E = EuclideanSpace(3)
sage: E.cartesian_coordinates()
Chart (E^3, (x, y, z))

sage: E.cartesian_coordinates().coord_range()
x: (-oo, +oo); y: (-oo, +oo); z: (-oo, +oo)
```

An example where the Cartesian coordinates have not been previously created:

```python
sage: E = EuclideanSpace(3, coordinates='spherical')
sage: E.atlas()  # only spherical coordinates have been initialized
[Chart (E^3, (r, th, ph))]
sage: E.cartesian_coordinates(symbols='X Y Z')
Chart (E^3, (X, Y, Z))
sage: E.atlas()  # the Cartesian chart has been added to the atlas
[Chart (E^3, (r, th, ph)), Chart (E^3, (X, Y, Z))]
```

The coordinate variables are returned by the square bracket operator:
Manifolds, Release 10.3

It is also possible to use the operator \(<,>\) to set symbolic variable containing the coordinates:

```
sage: E = EuclideanSpace(3, coordinates='spherical')
sage: cartesian.<u,v,w> = E.cartesian_coordinates()
sage: cartesian
Chart (E^3, (u, v, w))
sage: u, v, w
(u, v, w)
```

The command `cartesian.<u,v,w> = E.cartesian_coordinates()` is actually a shortcut for:

```
sage: cartesian = E.cartesian_coordinates(symbols='u v w')
sage: u, v, w = cartesian[:]
```

### cylindrical_coordinates (symbols=None, names=None)

Return the chart of cylindrical coordinates, possibly creating it if it does not already exist.

**INPUT:**

- **symbols** – (default: None) string defining the coordinate text symbols and \(\text{LaTeX}\) symbols, with the same conventions as the argument `coordinates` in `RealDiffChart`; this is used only if the cylindrical chart has not been already defined; if None the symbols are generated as \((\rho, \phi, z)\).

- **names** – (default: None) unused argument, except if `symbols` is not provided; it must be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator \(<,>\) is used)

**OUTPUT:**

- the chart of cylindrical coordinates, as an instance of `RealDiffChart`

**EXAMPLES:**

```
sage: E = EuclideanSpace(3)
sage: E.cylindrical_coordinates()
Chart (E^3, (rh, ph, z))
sage: latex(_)
\left(\mathbb{E}^{3},(\rho, \phi, z)\right)
sage: E.cylindrical_coordinates().coord_range()
rh: (0, +\infty); ph: [0, 2*pi] (periodic); z: (-\infty, +\infty)
```

The relation to Cartesian coordinates is:

```
sage: E.coord_change(E.cylindrical_coordinates(),
.....: E.cartesian_coordinates()).display()
\begin{align*}
x &= \rho \cos(\phi) \\
y &= \rho \sin(\phi) \\
z &= z
\end{align*}
sage: E.coord_change(E.cartesian_coordinates(),
.....: E.cylindrical_coordinates()).display()
\begin{align*}
\rho &= \sqrt{x^2 + y^2} \\
\phi &= \arctan2(y, x) \\
z &= z
\end{align*}
```
The coordinate variables are returned by the square bracket operator:

```python
sage: E.cylindrical_coordinates()[1]
rh
sage: E.cylindrical_coordinates()[3]
z
sage: E.cylindrical_coordinates()[:]
(rh, ph, z)
```

They can also be obtained via the operator `<,>`:

```python
sage: cylindrical.<rh,ph,z> = E.cylindrical_coordinates()
sage: cylindrical
Chart (E^3, (rh, ph, z))
sage: rh, ph, z
(rh, ph, z)
```

Actually, `cylindrical.<rh,ph,z> = E.cylindrical_coordinates()` is a shortcut for:

```python
sage: cylindrical = E.cylindrical_coordinates()
sage: rh, ph, z = cylindrical[:]
```

The coordinate symbols can be customized:

```python
sage: E = EuclideanSpace(3)
sage: E.cylindrical_coordinates(symbols=r"R Phi: \Phi Z")
```

```
Chart (E^3, (R, Phi, Z))
sage: latex(E.cylindrical_coordinates())
\left(\mathbb{E}^{3},(R, {\Phi}, Z)\right)
```

Note that if the cylindrical coordinates have been already initialized, the argument `symbols` has no effect:

```python
sage: E.cylindrical_coordinates(symbols=r"rh:\rho ph: \phi z")
```

`cylindrical_frame()`

Return the orthonormal vector frame associated with cylindrical coordinates.

**OUTPUT:**

- VectorFrame

**EXAMPLES:**

```python
sage: E = EuclideanSpace(3)
sage: E.cylindrical_frame()
Vector frame (E^3, (e_rh,e_ph,e_z))
sage: E.cylindrical_frame()[1]
(Vector field e_rh on the Euclidean space E^3, e_ph on the Euclidean space E^3, e_z on the Euclidean space E^3)
```

The cylindrical frame expressed in terms of the Cartesian one:

```python
sage: for e in E.cylindrical_frame():
    ....:     e.display(E.cartesian_frame(), E.cylindrical_coordinates())
```

(continues on next page)
\[ e_r = \cos(\phi) \ e_x + \sin(\phi) \ e_y \]
\[ e_\phi = -\sin(\phi) \ e_x + \cos(\phi) \ e_y \]
\[ e_z = e_z \]

The orthonormal frame \((e_r, e_\phi, e_z)\) expressed in terms of the coordinate frame \(\left( \frac{\partial}{\partial r}, \frac{\partial}{\partial \phi}, \frac{\partial}{\partial z} \right)\):

```
sage: for e in E.cylindrical_frame():
    ....:    e.display(E.cylindrical_coordinates())
    e_rh = \partial/\partial r
    e_ph = 1/rh \partial/\partial \phi
    e_z = \partial/\partial z
```

\textbf{scalar\_triple\_product} \(\text{name=None, latex}\_\text{name}=\text{None}\)

Return the scalar triple product operator, as a 3-form.

The \textit{scalar triple product} (also called \textit{mixed product}) of three vector fields \(u, v\) and \(w\) defined on an Euclidean space \(E\) is the scalar field

\[ \epsilon(u, v, w) = u \cdot (v \times w). \]

The scalar triple product operator \(\epsilon\) is a \textit{3-form}, i.e. a field of fully antisymmetric trilinear forms; it is also called the \textit{volume form} of \(E\) or the \textit{Levi-Civita tensor} of \(E\).

\textbf{INPUT}:

- \texttt{name} – (default: None) string; name given to the scalar triple product operator; if None, 'epsilon' is used
- \texttt{latex\_name} – (default: None) string; \LaTeX\ symbol to denote the scalar triple product; if None, it is set to \texttt{r'\epsilon'} if \texttt{name} is None and to \texttt{name} otherwise.

\textbf{OUTPUT}:

- the scalar triple product operator \(\epsilon\), as an instance of \texttt{DiffFormParal}

\textbf{EXAMPLES}:

```
sage: E.<x,y,z> = EuclideanSpace()
sage: triple_product = E.scalar_triple_product()
sage: triple_product
3-form epsilon on the Euclidean space E^3
sage: latex(triple_product)
\epsilon
sage: u = E.vector_field(x, y, z, name='u')
sage: v = E.vector_field(-y, x, 0, name='v')
sage: w = E.vector_field(y*z, x*z, x*y, name='w')
sage: s = triple_product(u, v, w); s
Scalar field epsilon(u,v,w) on the Euclidean space E^3
sage: s.display()
epsilon(u,v,w): E^3 → ℝ
\(x, y, z\) ↦ \(x^3*y + x*y^3 - 2*x*y*z^2\)
sage: s.expr()
x^3*y + x*y^3 - 2*x*y*z^2
sage: latex(s)
\epsilon\left(u,v,w\right)
sage: s == - triple_product(w, v, u)
True
```

Check of the identity \(\epsilon(u, v, w) = u \cdot (v \times w)\):
Customizing the name:

```
sage: E.scalar_triple_product(name='S')
3-form S on the Euclidean space E^3
sage: latex(_)
S
sage: E.scalar_triple_product(name='Omega', latex_name=r'\Omega')
3-form Omega on the Euclidean space E^3
sage: latex(_)
\Omega
```

`spherical_coordinates (symbols=\text{None}, names=\text{None})`

Return the chart of spherical coordinates, possibly creating it if it does not already exist.

**INPUT:**

- **symbols** — (default: \text{None}) string defining the coordinate text symbols and \LaTeX\ symbols, with the same conventions as the argument \text{coordinates} in \text{RealDiffChart}; this is used only if the spherical chart has not been already defined; if \text{None} the symbols are generated as \((r, \theta, \phi)\).
- **names** — (default: \text{None}) unused argument, except if \text{symbols} is not provided; it must be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator \(\langle, \rangle\) is used)

**OUTPUT:**

- the chart of spherical coordinates, as an instance of \text{RealDiffChart}

**EXAMPLES:**

```
sage: E = EuclideanSpace(3)
sage: E.spherical_coordinates()
Chart \((E^3, (r, \theta, \phi))\)
sage: latex(_)
\left(\mathbb{E}^{3},(r, \ {\theta}, \ {\phi})\right)
sage: E.spherical_coordinates().coord_range()
r: (0, +\infty); th: (0, \pi); ph: [0, 2\pi] (periodic)
```

The relation to Cartesian coordinates is:

```
sage: E.coord_change(E.spherical_coordinates(),
.....: E.cartesian_coordinates()).display()
x = r*cos(ph)*sin(th)
y = r*sin(ph)*sin(th)
z = r*cos(th)
sage: E.coord_change(E.cartesian_coordinates(),
.....: E.spherical_coordinates()).display()
r = sqrt(x^2 + y^2 + z^2)
th = asin2(sqrt(x^2 + y^2), z)
ph = asin2(y, x)
```

The coordinate variables are returned by the square bracket operator:

```
sage: E.spherical_coordinates()[1]
r
sage: E.spherical_coordinates()[3]
ph
```
They can also be obtained via the operator \(<,>\):

\[
\text{sage: spherical.<r, th, ph> = E.spherical_coordinates()}
\]
\[
\text{sage: spherical}
\]
\[
\text{Chart (E^3, (r, th, ph))}
\]
\[
\text{sage: r, th, ph}
\]
\[\text{(r, th, ph)}\]

Actually, \(\text{spherical.<r, th, ph> = E.spherical_coordinates()}\) is a shortcut for:

\[
\text{sage: spherical = E.spherical_coordinates()}
\]
\[
\text{sage: r, th, ph = spherical[:]}\]

The coordinate symbols can be customized:

\[
\text{sage: E = EuclideanSpace(3)}
\]
\[
\text{sage: E.spherical_coordinates(symbols=r"R T: \Theta F: \Phi")}
\]
\[
\text{Chart (E^3, (R, T, F))}
\]
\[
\text{sage: latex(E.spherical_coordinates())}
\]
\[
\left(\mathbb{E}^{3},(R, {\Theta}, {\Phi})\right)
\]

Note that if the spherical coordinates have been already initialized, the argument \text{symbols} has no effect:

\[
\text{sage: E.spherical_coordinates(symbols=r"r th: \theta ph: \phi")}
\]
\[
\text{Chart (E^3, (R, T, F))}
\]

\text{spherical_frame()}  
  Return the orthonormal vector frame associated with spherical coordinates.  

\text{OUTPUT:}  
  • VectorFrame

\text{EXAMPLES:}  

\[
\text{sage: E = EuclideanSpace(3)}
\]
\[
\text{sage: E.spherical_frame()}
\]
\[
\text{Vector frame (E^3, (e_r, e_th, e_ph))}
\]
\[
\text{sage: E.spherical_frame()[1]}
\]
\[
\text{Vector field e_r on the Euclidean space E^3}
\]
\[
\text{sage: E.spherical_frame()[:]}\]
\[
\text{(Vector field e_r on the Euclidean space E^3, Vector field e_th on the Euclidean space E^3, Vector field e_ph on the Euclidean space E^3)}\]

The spherical frame expressed in terms of the Cartesian one:

\[
\text{sage: for e in E.spherical_frame():}
\]
\[\ldots:
\]
\[
\text{e.display(E.cartesian_frame(), E.spherical_coordinates())}
\]
\[
\text{e_r = cos(ph)*sin(th) e_x + sin(ph)*sin(th) e_y + cos(th) e_z}
\]
\[
\text{e_th = cos(ph)*cos(th) e_x + cos(th)*sin(ph) e_y - sin(th) e_z}
\]
\[
\text{e_ph = -sin(ph) e_x + cos(ph) e_y}
\]

The orthonormal frame \((e_r, e_\theta, e_\phi)\) expressed in terms of the coordinate frame \(\left(\frac{\partial}{\partial r}, \frac{\partial}{\partial \theta}, \frac{\partial}{\partial \phi}\right)\):
sage: for e in E.spherical_frame():
....:     e.display(E.spherical_coordinates())

\(e_r = \frac{\partial}{\partial r}\)
\(e_{\theta} = \frac{1}{r} \frac{\partial}{\partial \theta}\)
\(e_{\phi} = \frac{1}{(r\sin(\theta))} \frac{\partial}{\partial \phi}\)

class sage.manifolds.differentiable.examples.euclidean.EuclideanPlane(name=None, latex_name=None, coordinates='Cartesian', symbols=None, metric_name='g', metric_latex_name=None, start_index=1, base_manifold=None, category=None, unique_tag=None)

Bases: EuclideanSpace

Euclidean plane.

An Euclidean plane is an affine space \(E\), whose associated vector space is a 2-dimensional vector space over \(\mathbb{R}\) and is equipped with a positive definite symmetric bilinear form, called the scalar product or dot product.

The class EuclideanPlane inherits from PseudoRiemannianManifold (via EuclideanSpace) since an Euclidean plane can be viewed as a Riemannian manifold that is diffeomorphic to \(\mathbb{R}^2\) and that has a flat metric \(g\). The Euclidean scalar product is the one defined by the Riemannian metric \(g\).

INPUT:

- name – (default: None) string; name (symbol) given to the Euclidean plane; if None, the name will be set to 'E'^2'
- latex_name – (default: None) string; \LaTeX{} symbol to denote the Euclidean plane; if None, it is set to '\mathbb{E}'^2' if name is None and to name otherwise
- coordinates – (default: 'Cartesian') string describing the type of coordinates to be initialized at the Euclidean plane creation; allowed values are 'Cartesian' (see cartesian_coordinates()) and 'polar' (see polar_coordinates())
- symbols – (default: None) string defining the coordinate text symbols and \LaTeX{} symbols, with the same conventions as the argument coordinates in RealDiffChart, namely symbols is a string of coordinate fields separated by a blank space, where each field contains the coordinate's text symbol and possibly the coordinate's \LaTeX{} symbol (when the latter is different from the text symbol), both symbols being separated by a colon (:); if None, the symbols will be automatically generated according to the value of coordinates
- metric_name – (default: 'g') string; name (symbol) given to the Euclidean metric tensor
• `metric_latex_name` – (default: None) string; LaTeX symbol to denote the Euclidean metric tensor; if none is provided, it is set to `metric_name`

• `start_index` – (default: 1) integer; lower value of the range of indices used for “indexed objects” in the Euclidean plane, e.g. coordinates of a chart

• `base_manifold` – (default: None) if not None, must be an Euclidean plane; the created object is then an open subset of `base_manifold`

• `category` – (default: None) to specify the category; if None, `Manifolds(RR).Smooth() & MetricSpaces().Complete()` is assumed

• `names` – (default: None) unused argument, except if `symbols` is not provided; it must then be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator `<,>` is used)

• `init_coord_methods` – (default: None) dictionary of methods to initialize the various type of coordinates, with each key being a string describing the type of coordinates; to be used by derived classes only

• `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 `PseudoRiemannianManifold` would return the previously constructed object corresponding to these arguments)

EXAMPLES:

One creates an Euclidean plane `E` with:

```sage```
```e
E.<x,y> = EuclideanSpace(); E
Euclidean plane E^2
```
```
esage: E.category()
Join of Category of smooth manifolds over Real Field with 53 bits of precision and Category of connected manifolds over Real Field with 53 bits of precision and Category of complete metric spaces
sage: dim(E)
2
```

It is endowed with a default coordinate chart, which is that of Cartesian coordinates `(x, y)`:  

```sage```
```e
sage: E.atlas()
[Chart (E^2, (x, y))]
sage: E.default_chart()
Chart (E^2, (x, y))
sage: cartesian = E.cartesian_coordinates()
sage: cartesian
is E.default_chart()
True
```
```
A point of `E`:

```sage```
```e
sage: p = E((3,-2)); p
Point on the Euclidean plane E^2
sage: cartesian(p)
(3, -2)
sage: p in E
True
sage: p.parent() is E
True
```
```
E is endowed with a default metric tensor, which defines the Euclidean scalar product:
Curvilinear coordinates can be introduced on $E$: see `polar_coordinates()`.

See also:

**Example 1: the Euclidean plane**

`cartesian_coordinates(symbols=None, names=None)`

Return the chart of Cartesian coordinates, possibly creating it if it does not already exist.

**INPUT:**

- `symbols` – (default: `None`) string defining the coordinate text symbols and LaTeX symbols, with the same conventions as the argument `coordinates` in `RealDiffChart`; this is used only if the Cartesian chart has not been already defined; if `None` the symbols are generated as $(x, y)$.
- `names` – (default: `None`) unused argument, except if `symbols` is not provided; it must be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator `<,>` is used)

**OUTPUT:**

- the chart of Cartesian coordinates, as an instance of `RealDiffChart`

**EXAMPLES:**

```python
sage: E = EuclideanSpace(2)
sage: E.cartesian_coordinates()
Chart (E^2, (x, y))
sage: E.cartesian_coordinates().coord_range()
x: (-oo, +oo); y: (-oo, +oo)
```

An example where the Cartesian coordinates have not been previously created:

```python
sage: E = EuclideanSpace(2, coordinates='polar')
sage: E.atlas()  # only polar coordinates have been initialized
[Chart (E^2, (r, ph))]
sage: E.cartesian_coordinates(symbols='X Y')
Chart (E^2, (X, Y))
sage: E.atlas()  # the Cartesian chart has been added to the atlas
[Chart (E^2, (r, ph)), Chart (E^2, (X, Y))]
```

Note that if the Cartesian coordinates have been already initialized, the argument `symbols` has no effect:

```python
sage: E.cartesian_coordinates(symbols='x y')
Chart (E^2, (X, Y))
```

The coordinate variables are returned by the square bracket operator:

```python
sage: E.cartesian_coordinates()[:]
(X, Y)
```

It is also possible to use the operator `<,>` to set symbolic variable containing the coordinates:
sage: E = EuclideanSpace(2, coordinates='polar')
sage: cartesian.<u,v> = E.cartesian_coordinates()
sage: cartesian
Chart (E^2, (u, v))
sage: u,v
(u, v)

The command `cartesian.<u,v> = E.cartesian_coordinates()` is actually a shortcut for:

sage: cartesian = E.cartesian_coordinates(symbols='u v')
sage: u, v = cartesian[:]

**polar_coordinates** *(symbols=None, names=None)*

Return the chart of polar coordinates, possibly creating it if it does not already exist.

**INPUT:**

- **symbols** – (default: None) string defining the coordinate text symbols and LaTeX symbols, with the same conventions as the argument `coordinates` in `RealDiffChart`; this is used only if the polar chart has not been already defined; if `None` the symbols are generated as `(r, \phi)`.
- **names** – (default: None) unused argument, except if `symbols` is not provided; it must be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator `<,>` is used)

**OUTPUT:**

- the chart of polar coordinates, as an instance of `RealDiffChart`

**EXAMPLES:**

sage: E = EuclideanSpace(2)
sage: E.polar_coordinates()
Chart (E^2, (r, ph))

sage: latex(_)
\left(\mathbb{E}^{2},(r, \phi)\right)

sage: E.polar_coordinates().coord_range()
r: (0, +oo); ph: [0, 2*pi] (periodic)

The relation to Cartesian coordinates is:

sage: E.coord_change(E.polar_coordinates(),
.....: E.cartesian_coordinates()).display()
x = r*cos(ph)
y = r*sin(ph)

sage: E.coord_change(E.cartesian_coordinates(),
.....: E.polar_coordinates()).display()
r = sqrt(x^2 + y^2)
ph = arctan2(y, x)

The coordinate variables are returned by the square bracket operator:

sage: E.polar_coordinates()[1]
r
sage: E.polar_coordinates()[2]
ph
sage: E.polar_coordinates()[:]
(r, ph)

They can also be obtained via the operator `<,>`:
Manifolds, Release 10.3

\begin{verbatim}
sage: polar.<r,ph> = E.polar_coordinates(); polar
Chart (E^2, (r, ph))
sage: r, ph
(r, ph)

Actually, polar.<r,ph> = E.polar_coordinates() is a shortcut for:

sage: polar = E.polar_coordinates()
sage: r, ph = polar[:]

The coordinate symbols can be customized:

sage: E = EuclideanSpace(2)
sage: E.polar_coordinates(symbols=r"r th: \theta")
Chart (E^2, (r, th))
sage: latex(E.polar_coordinates())
\left(\mathbb{E}^{2},(r, \{\theta})\right)

Note that if the polar coordinates have been already initialized, the argument symbols has no effect:

sage: E.polar_coordinates(symbols=r"R Th: \Theta")
Chart (E^2, (r, th))

\textbf{polar_frame()}

Return the orthonormal vector frame associated with polar coordinates.

\textbf{OUTPUT}:

\begin{itemize}
  \item instance of \texttt{VectorFrame}
\end{itemize}

\textbf{EXAMPLES}:

\begin{verbatim}
sage: E = EuclideanSpace(2)
sage: E.polar_frame()
Vector frame (E^2, (e_r,e_ph))
sage: E.polar_frame()[:]
(Vector field e_r on the Euclidean plane E^2, Vector field e_ph on the Euclidean plane E^2)
\end{verbatim}

The orthonormal polar frame expressed in terms of the Cartesian one:

\begin{verbatim}
sage: for e in E.polar_frame():
    ....:     e.display(E.cartesian_frame(), E.polar_coordinates())
e_r = cos(ph) e_x + sin(ph) e_y
e_ph = -sin(ph) e_x + cos(ph) e_y
\end{verbatim}

The orthonormal frame \((e_r, e_\phi)\) expressed in terms of the coordinate frame \(\left(\frac{\partial}{\partial r}, \frac{\partial}{\partial \phi}\right)\):

\begin{verbatim}
sage: for e in E.polar_frame():
    ....:     e.display(E.polar_coordinates())
e_r = \partial/\partial r
e_ph = 1/r \partial/\partial \phi
\end{verbatim}
\end{verbatim}
Euclidean space.

An Euclidean space of dimension $n$ is an affine space $E$, whose associated vector space is a $n$-dimensional vector space over $\mathbb{R}$ and is equipped with a positive definite symmetric bilinear form, called the scalar product or dot product.

Euclidean space of dimension $n$ can be viewed as a Riemannian manifold that is diffeomorphic to $\mathbb{R}^n$ and that has a flat metric $g$. The Euclidean scalar product is the one defined by the Riemannian metric $g$.

INPUT:

- $n$ – positive integer; dimension of the space over the real field
- name – (default: None) string; name (symbol) given to the Euclidean space; if None, the name will be set to `$E^n$`
- latex_name – (default: None) string; LaTeX symbol to denote the space; if None, it is set to `\mathbb{E}^{n}` if name is None and to name otherwise
- coordinates – (default: 'Cartesian') string describing the type of coordinates to be initialized at the Euclidean space creation; allowed values are
  - 'Cartesian' (canonical coordinates on $\mathbb{R}^n$)
  - 'polar' for $n=2$ only (see `polar_coordinates()`)
  - 'spherical' for $n=3$ only (see `spherical_coordinates()`)
  - 'cylindrical' for $n=3$ only (see `cylindrical_coordinates()`)
- symbols – (default: None) string defining the coordinate text symbols and LaTeX symbols, with the same conventions as the argument coordinates in `RealDiffChart`, namely symbols is a string of coordinate fields separated by a blank space, where each field contains the coordinate's text symbol and possibly the coordinate's LaTeX symbol (when the latter is different from the text symbol), both symbols being separated
by a colon (:); if None, the symbols will be automatically generated according to the value of coordinates

- **metric_name** – (default: 'g') string; name (symbol) given to the Euclidean metric tensor
- **metric_latex_name** – (default: None) string; LaTeX symbol to denote the Euclidean metric tensor; if none is provided, it is set to **metric_name**
- **start_index** – (default: 1) integer; lower value of the range of indices used for “indexed objects” in the Euclidean space, e.g. coordinates of a chart
- **names** – (default: None) unused argument, except if **symbols** is not provided; it must then be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator <,> is used)

If **names** is specified, then **n** does not have to be specified.

**EXAMPLES:**

Constructing a 2-dimensional Euclidean space:

```python
sage: E = EuclideanSpace(2); E
Euclidean plane E^2
```

Each call to **EuclideanSpace** creates a different object:

```python
sage: E1 = EuclideanSpace(2)
sage: E1 is E
False
sage: E1 == E
False
```

The LaTeX symbol of the Euclidean space is by default E^n, where n is the dimension:

```python
sage: latex(E)
\mathbb{E}^{2}
```

But both the name and LaTeX names of the Euclidean space can be customized:

```python
sage: F = EuclideanSpace(2, name='F', latex_name=r'\mathcal{F}'); F
Euclidean plane F
sage: latex(F)
\mathcal{F}
```

By default, an Euclidean space is created with a single coordinate chart: that of Cartesian coordinates:

```python
sage: E.atlas()
[Chart (E^2, (x, y))]
sage: E.cartesian_coordinates()
Chart (E^2, (x, y))
sage: E.default_chart() is E.cartesian_coordinates()
True
```

The coordinate variables can be initialized, as the Python variables x and y, by:

```python
sage: x, y = E.cartesian_coordinates()[:]
```

However, it is possible to both construct the Euclidean space and initialize the coordinate variables in a single stage, thanks to SageMath operator <,>:
Note that providing the dimension as an argument of `EuclideanSpace` is not necessary in that case, since it can be deduced from the number of coordinates within `<,>`. Besides, the coordinate symbols can be customized:

```
sage: E.<x,y> = EuclideanSpace()
sage: E.cartesian_coordinates()
Chart (E^2, (X, Y))
```

By default, the LaTeX symbols of the coordinates coincide with the text ones:

```
sage: latex(X+Y)
X + Y
```

However, it is possible to customize them, via the argument `symbols`, which must be a string, usually prefixed by `r` (for `raw` string, in order to allow for the backslash character of LaTeX expressions). This string contains the coordinate fields separated by a blank space; each field contains the coordinate’s text symbol and possibly the coordinate’s LaTeX symbol (when the latter is different from the text symbol), both symbols being separated by a colon (`:`):

```
sage: E.<xi,ze> = EuclideanSpace(symbols=r"xi:\xi ze: \zeta")
sage: E.cartesian_coordinates()
Chart (E^2, (xi, ze))
sage: latex(xi+ze)
{\xi} + {\zeta}
```

Thanks to the argument `coordinates`, an Euclidean space can be constructed with curvilinear coordinates initialized instead of the Cartesian ones:

```
sage: E.<r,ph> = EuclideanSpace(coordinates='polar')
sage: E.atlas()  # no Cartesian coordinates have been constructed
[Chart (E^2, (r, ph))]
sage: polar = E.polar_coordinates(); polar
Chart (E^2, (r, ph))
sage: E.default_chart() is polar
True
sage: latex(r+ph)
{\phi} + r
```

The Cartesian coordinates, along with the transition maps to and from the curvilinear coordinates, can be constructed at any time by:

```
sage: cartesian.<x,y> = E.cartesian_coordinates()
sage: E.atlas()  # both polar and Cartesian coordinates now exist
[Chart (E^2, (r, ph)), Chart (E^2, (x, y))]
```

The transition maps have been initialized by the command `E.cartesian_coordinates()`:

```
sage: E.coord_change(polar, cartesian).display()
x = r*cos(ph)
y = r*sin(ph)
sage: E.coord_change(cartesian, polar).display()
r = sqrt(x^2 + y^2)
ph = arctan2(y, x)
```

The default name of the Euclidean metric tensor is $g$: 

```
3.2. Euclidean Spaces and Vector Calculus 877
```
sage: E.metric()
Riemannian metric g on the Euclidean plane E^2
sage: latex(_)
g
But this can be customized:

sage: E = EuclideanSpace(2, metric_name='h')
sage: E.metric()
Riemannian metric h on the Euclidean plane E^2
sage: latex(_)
h
sage: E = EuclideanSpace(2, metric_latex_name=r'\mathbf{g}')
sage: E.metric()
Riemannian metric g on the Euclidean plane E^2
sage: latex(_)
\mathbf{g}

A 4-dimensional Euclidean space:

sage: E = EuclideanSpace(4); E
4-dimensional Euclidean space E^4
sage: latex(E)
\mathbb{E}^4

E is both a real smooth manifold of dimension 4 and a complete metric space:

sage: E.category()
Join of Category of smooth manifolds over Real Field with 53 bits of precision and Category of connected manifolds over Real Field with 53 bits of precision and Category of complete metric spaces
sage: dim(E)
4

It is endowed with a default coordinate chart, which is that of Cartesian coordinates \((x_1, x_2, x_3, x_4)\):

sage: E.atlas()
[Chart (E^4, (x1, x2, x3, x4))]
sage: E.default_chart()
Chart (E^4, (x1, x2, x3, x4))
sage: E.default_chart() is E.cartesian_coordinates()
True

E is also endowed with a default metric tensor, which defines the Euclidean scalar product:

sage: g = E.metric(); g
Riemannian metric g on the 4-dimensional Euclidean space E^4
sage: g.display()
g = dx1⊗dx1 + dx2⊗dx2 + dx3⊗dx3 + dx4⊗dx4

cartesian_coordinates (symbols=None, names=None)

Return the chart of Cartesian coordinates, possibly creating it if it does not already exist.

INPUT:

- symbols – (default: None) string defining the coordinate text symbols and LaTeX symbols, with the same conventions as the argument coordinates in RealDiffChart; this is used only if the Cartesian chart has not been already defined; if None the symbols are generated as \((x_1, \ldots, x_n)\).
• names – (default: None) unused argument, except if symbols is not provided; it must be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator <, > is used)

OUTPUT:
• the chart of Cartesian coordinates, as an instance of RealDiffChart

EXAMPLES:
```
sage: E = EuclideanSpace(4)
sage: X = E.cartesian_coordinates(); X
Chart (E^4, (x1, x2, x3, x4))
sage: X[2]
x2
sage: X[:]
(x1, x2, x3, x4)
sage: latex(X[:])
\left((x_{1}), (x_{2}), (x_{3}), (x_{4})\right)
```

cartesian_frame()

Return the orthonormal vector frame associated with Cartesian coordinates.

OUTPUT:
• CoordFrame

EXAMPLES:
```
sage: E = EuclideanSpace(2)
sage: E.cartesian_frame()
Coordinate frame (E^2, (e_x,e_y))
sage: E.cartesian_frame()[1]
Vector field e_x on the Euclidean plane E^2
sage: E.cartesian_frame()[:]
(Vector field e_x on the Euclidean plane E^2,
 Vector field e_y on the Euclidean plane E^2)
```

For Cartesian coordinates, the orthonormal frame coincides with the coordinate frame:
```
sage: E.cartesian_frame() is E.cartesian_coordinates().frame()
True
```

dist (p, q)

Euclidean distance between two points.

INPUT:
• p – an element of self
• q – an element of self

OUTPUT:
• the Euclidean distance \(d(p, q)\)

EXAMPLES:
```
sage: E.<x,y> = EuclideanSpace()
sage: p = E((1,0))
```

(continues on next page)
sphere \( (\text{radius}=1, \text{center}=\text{None}, \text{name}=\text{None}, \text{latex\_name}=\text{None}, \text{coordinates}=\text{\textquote{\textit{spherical}}}, \text{names}=\text{None}) \)

Return an \((n-1)\)-sphere smoothly embedded in \self.

**INPUT:**

- `radius` – (default: 1) the radius greater than 1 of the sphere
- `center` – (default: \text{None}) point on \self\ representing the barycenter of the sphere
- `name` – (default: \text{None}) string; name (symbol) given to the sphere; if \text{None}, the name will be generated according to the input
- `latex\_name` – (default: \text{None}) string; \LaTeX\ symbol to denote the sphere; if \text{None}, the symbol will be generated according to the input
- `coordinates` – (default: \text{'\textit{spherical}'}) string describing the type of coordinates to be initialized at the sphere’s creation; allowed values are
  - 'spherical' spherical coordinates (see \text{spherical\_coordinates()}))
  - 'stereographic' stereographic coordinates given by the stereographic projection (see \text{stereographic\_coordinates()})
- `names` – (default: \text{None}) must be a tuple containing the coordinate symbols (this guarantees the shortcut operator \text{<,>} to function); if \text{None}, the usual conventions are used (see examples in \text{Sphere} for details)

**EXAMPLES:**

Define a 2-sphere with radius 2 centered at \((1, 2, 3)\) in Cartesian coordinates:

```
sage: E3 = EuclideanSpace(3)
sage: c = E3.point((1,2,3), name='c'); c
Point c on the Euclidean space E^3
sage: S2_2 = E3.sphere(radius=2, center=c); S2_2
2-sphere S^2_{\text{\textit{2}}}(c) of radius 2 smoothly embedded in the Euclidean space E^3 centered at the Point c
```

The ambient space is precisely our previously defined Euclidean space:

```
sage: S2_2.ambient() is E3
True
```

The embedding into Euclidean space:

```
sage: S2_2.embedding().display()
iota: S^2_{\text{\textit{2}}}(c) \rightarrow E^3
on A: (\theta, \phi) \mapsto (x, y, z) = (2\cos(\phi)\sin(\theta) + 1,
  2\sin(\phi)\sin(\theta) + 2,
  2\cos(\theta) + 3)
```

See \text{Sphere} for more examples.
### 3.2.2 Spheres smoothly embedded in Euclidean Space

Let $E^{n+1}$ be a Euclidean space of dimension $n + 1$ and $c \in E^{n+1}$. An $n$-sphere with radius $r$ and centered at $c$, usually denoted by $S^n_r(c)$, smoothly embedded in the Euclidean space $E^{n+1}$ is an $n$-dimensional smooth manifold together with a smooth embedding 

$$
iota: S^n_r \rightarrow E^{n+1}$$

whose image consists of all points having the same Euclidean distance to the fixed point $c$. If we choose Cartesian coordinates $(x_1, \ldots, x_{n+1})$ on $E^{n+1}$ with $x(c) = 0$ then the above translates to

$$
iota(S^n_r(c)) = \{ p \in E^{n+1} : \|x(p)\| = r \}.$$ 

This corresponds to the standard $n$-sphere of radius $r$ centered at $c$.

**AUTHORS:**

- Michael Jung (2020): initial version

**REFERENCES:**

- M. Berger: *Geometry I&II* [Ber1987], [Ber1987a]
- J. Lee: *Introduction to Smooth Manifolds* [Lee2013]

**EXAMPLES:**

We start by defining a 2-sphere of unspecified radius $r$:

```sage
r = var('r')
sage: S2_r = manifolds.Sphere(2, radius=r); S2_r
2-sphere $S^2_r$ of radius $r$ smoothly embedded in the Euclidean space $E^3$
```

The embedding $\iota$ is constructed from scratch and can be returned by the following command:

```sage
i = S2_r.embedding(); i
iota: 2-sphere $S^2_r$ of radius $r$ smoothly embedded in the Euclidean space $E^3$
```

As a submanifold of a Riemannian manifold, namely the Euclidean space, the 2-sphere admits an induced metric:

```sage
g = S2_r.induced_metric()
sage: g.display()
g = r^2 d\theta \otimes d\theta + r^2 \sin(\theta)^2 d\phi \otimes d\phi
```

The induced metric is also known as the *first fundamental form* (see `first_fundamental_form()`):

```sage
sage: g.is(S2_r.first_fundamental_form())
True
```

The *second fundamental form* encodes the extrinsic curvature of the 2-sphere as hypersurface of Euclidean space (see `second_fundamental_form()`).
One quantity that can be derived from the second fundamental form is the Gaussian curvature:

\[
K = S^2_r \text{gauss_curvature()}
\]

As we have seen, spherical coordinates are initialized by default. To initialize stereographic coordinates retrospectively, we can use the following command:

\[
\text{sage: } S^2_r \text{stereographic_coordinates()}
\]

To get all charts corresponding to stereographic coordinates, we can use the `coordinate_charts()`:

\[
\text{sage: } \text{stereoN, stereoS} = S^2_r \text{coordinate_charts('stereographic')}\]

See also:

See `stereographic_coordinates()` and `spherical_coordinates()` for details.

Note: Notice that the derived quantities such as the embedding as well as the first and second fundamental forms must be computed from scratch again when new coordinates have been initialized. That makes the usage of previously declared objects obsolete.

Consider now a 1-sphere with barycenter \((1, 0)\) in Cartesian coordinates:

\[
\text{sage: } E2 = \text{EuclideanSpace(2)}
\]

\[
\text{sage: } c = E2 \text{point((1,0), name='c')}
\]

Get stereographic coordinates:

\[
\text{sage: } \text{stereoN, stereoS = S1c-coordinate_charts('stereographic')}\]

The embedding takes now the following form in all coordinates:

\[
\text{sage: } S1c \text{embedding().display()}
\]

\[
iota: S^1(c) \to E^2
\]

882 Chapter 3. Pseudo-Riemannian Manifolds
Since the sphere is a hypersurface, we can get a normal vector field by using `normal`:

```sage
n = S1c.normal(); n
```
Vector field `n` along the 1-sphere $S^1(c)$ of radius 1 smoothly embedded in the Euclidean plane $E^2$ centered at the Point $c$ with values on the Euclidean plane $E^2$

```sage
n.display()
n = -cos(chi) e_x - sin(chi) e_y
```

Notice that this is just one normal field with arbitrary direction, in this particular case $n$ points inwards whereas $-n$ points outwards. However, the vector field $n$ is indeed non-vanishing and hence the sphere admits an orientation (as all spheres do):

```sage
orient = S1c.orientation(); orient
[f = orient[1]]
f[1].display()
f_1 = -∂/∂y1
```

Notice that the orientation is chosen is such a way that $(n, -n)$ is oriented in the ambient Euclidean space, i.e. the last entry is the normal vector field pointing outwards. Henceforth, the manifold admits a volume form:

```sage
g = S1c.induced_metric()
g.display()
g = dchi⊗dchi
g = g.display()
geps = g.volume_form()
eps.display()
geps = -dchi
```

```class sage.manifolds.differentiable.examples.sphere.Sphere(n, radius=1, ambient_space=None, center=None, name=None, latex_name=None, coordinates='spherical', names=None, category=None, init_coord_methods=None, unique_tag=None)

Bases: PseudoRiemannianSubmanifold

Sphere smoothly embedded in Euclidean Space.

An $n$-sphere of radius $r$ smoothly embedded in Euclidean space $E^{n+1}$ is a smooth $n$-dimensional manifold smoothly embedded into $E^{n+1}$, such that the embedding constitutes a standard $n$-sphere of radius $r$ in that Euclidean space (possibly shifted by a point).

- $n$ – positive integer representing dimension of the sphere
- `radius` – (default: 1) positive number that states the radius of the sphere
- `name` – (default: None) string; name (symbol) given to the sphere; if `None`, the name will be set according to the input (see convention above)
- `ambient_space` – (default: None) Euclidean space in which the sphere should be embedded; if `None`, a new instance of Euclidean space is created
- `center` – (default: None) the barycenter of the sphere as point of the ambient Euclidean space; if `None` the barycenter is set to the origin of the ambient space’s standard Cartesian coordinates
```
• `latex_name` – *(default: `None`)* string; LaTeX symbol to denote the space; if `None`, it will be set according to the input (see convention above)

• `coordinates` – *(default: `'spherical'`)* string describing the type of coordinates to be initialized at the sphere’s creation; allowed values are
  
  - `'spherical'` spherical coordinates (see `spherical_coordinates()`)
  - `'stereographic'` stereographic coordinates given by the stereographic projection (see `stereographic_coordinates()`)

• `names` – *(default: `None`)* must be a tuple containing the coordinate symbols (this guarantees the shortcut operator `<, >` to function); if `None`, the usual conventions are used (see examples below for details)

• `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 `PseudoRiemannianManifold` would return the previously constructed object corresponding to these arguments)

**EXAMPLES:**

A 2-sphere embedded in Euclidean space:

```python
sage: S2 = manifolds.Sphere(2); S2
2-sphere S^2 of radius 1 smoothly embedded in the Euclidean space E^3

sage: latex(S2)
\mathbb{S}^{2}
```

The ambient Euclidean space is constructed incidentally:

```python
sage: S2.ambient()
Euclidean space E^3
```

Another call creates another sphere and hence another Euclidean space:

```python
sage: S2 is manifolds.Sphere(2)
False
sage: S2.ambient() is manifolds.Sphere(2).ambient()
False
```

By default, the barycenter is set to the coordinate origin of the standard Cartesian coordinates in the ambient Euclidean space:

```python
sage: c = S2.center(); c
Point on the Euclidean space E^3
sage: c.coord()
(0, 0, 0)
```

Each \( n \)-sphere is a compact manifold and a complete metric space:

```python
sage: S2.category()
Join of Category of compact topological spaces and Category of smooth manifolds over Real Field with 53 bits of precision and Category of connected manifolds over Real Field with 53 bits of precision and Category of complete metric spaces
```

If not stated otherwise, each \( n \)-sphere is automatically endowed with spherical coordinates:
Notice that the spherical coordinates do not cover the whole sphere. To cover the entire sphere with charts, use stereographic coordinates instead:

```python
sage: stereoN, stereoS = S2.coordinate_charts('stereographic')
sage: stereoN, stereoS
(Chart (S^2-{NP}, (y1, y2)), Chart (S^2-{SP}, (yp1, yp2)))
sage: list(S2.open_covers())
[Set {S^2} of open subsets of the 2-sphere S^2 of radius 1 smoothly embedded in...
Set {S^2-{NP}, S^2-{SP}) of open subsets of the 2-sphere S^2 of radius 1...
smoothly embedded in the Euclidean space E^3]
```

**Note:** Keep in mind that the initialization process of stereographic coordinates and their transition maps is computational complex in higher dimensions. Henceforth, high computation times are expected with increasing dimension.

---

**center()**

Return the barycenter of self in the ambient Euclidean space.

**EXAMPLES:**

2-sphere embedded in Euclidean space centered at (1, 2, 3) in Cartesian coordinates:

```python
sage: E3 = EuclideanSpace(3)
sage: c = E3.point((1,2,3), name = 'c')
sage: S2c = manifolds.Sphere(2, ambient_space=E3, center=c); S2c
2-sphere S^2(c) of radius 1 smoothly embedded in the Euclidean space
E^3 centered at the Point c
sage: S2c.center()
Point c on the Euclidean space E^3
```

We can see that the embedding is shifted accordingly:

```python
sage: S2c.embedding().display()

iota: S^2(c) → E^3
on A: (theta, phi) ↦ (x, y, z) = (cos(phi)*sin(theta) + 1,
   sin(phi)*sin(theta) + 2,
   cos(theta) + 3)
```

**coordinate_charts(coord_name, names=None)**

Return a list of all charts belonging to the coordinates coord_name.

**INPUT:**

- coord_name – string describing the type of coordinates
- names = (default: None) must be a tuple containing the coordinate symbols for the first chart in the list; if None, the standard convention is used

**EXAMPLES:**

---

3.2. Euclidean Spaces and Vector Calculus
Spherical coordinates on $S^1$:

```
sage: S1 = manifolds.Sphere(1)
sage: S1.coordinate_charts('spherical')
[Chart (A, (phi,))]```

Stereographic coordinates on $S^1$:

```
sage: stereo_charts = S1.coordinate_charts('stereographic', names=['a'])
sage: stereo_charts
[Chart (S^1-{NP}, (a,)), Chart (S^1-{SP}, (ap,))]```

**dist** $(p, q)$

Return the great circle distance between the points $p$ and $q$ on `self`.

**INPUT:**

- $p$ – an element of `self`
- $q$ – an element of `self`

**OUTPUT:**

- the great circle distance $d(p, q)$ on `self`

The great circle distance $d(p, q)$ of the points $p, q \in S^n(c)$ is the length of the shortest great circle segment on $S^n(c)$ that joins $p$ and $q$. If we choose Cartesian coordinates $(x_1, \ldots, x_{n+1})$ of the ambient Euclidean space such that the center lies in the coordinate origin, i.e. $x(c) = 0$, the great circle distance can be expressed in terms of the following formula:

$$d(p, q) = r \arccos \left( \frac{x(i(p)) \cdot x(i(q))}{r^2} \right).$$

**EXAMPLES:**

Define a 2-sphere with unspecified radius:

```
sage: r = var('r')
sage: S2_r = manifolds.Sphere(2, radius=r); S2_r
2-sphere $S^2_r$ of radius $r$ smoothly embedded in the Euclidean space $E^3$
```

Given two antipodal points in spherical coordinates:

```
sage: p = S2_r.point((pi/2, pi/2), name='p'); p
Point p on the 2-sphere $S^2_r$ of radius $r$ smoothly embedded in the Euclidean space $E^3$
sage: q = S2_r.point((pi/2, -pi/2), name='q'); q
Point q on the 2-sphere $S^2_r$ of radius $r$ smoothly embedded in the Euclidean space $E^3$
```

The distance is determined as the length of the half great circle:

```
sage: S2_r.dist(p, q)
pi*r
```

**minimal_triangulation**()

Return the minimal triangulation of `self` as a simplicial complex.

**EXAMPLES:**

Minimal triangulation of the 2-sphere:
**radius()**

Return the radius of self.

**EXAMPLES:**

3-sphere with radius 3:

```python
sage: S3_2 = manifolds.Sphere(3, radius=2); S3_2
3-sphere S^3_2 of radius 2 smoothly embedded in the 4-dimensional Euclidean space E^4
sage: S3_2.radius()
2
```

2-sphere with unspecified radius:

```python
sage: r = var('r')
sage: S2_r = manifolds.Sphere(3, radius=r); S2_r
3-sphere S^3_r of radius r smoothly embedded in the 4-dimensional Euclidean space E^4
sage: S2_r.radius()
r
```

**spherical_coordinates(names=None)**

Return the spherical coordinates of self.

**INPUT:**

- names - (default: None) must be a tuple containing the coordinate symbols (this guarantees the usage of the shortcut operator <, >)

**OUTPUT:**

- the chart of spherical coordinates, as an instance of `RealDiffChart`

Let \( S^n_r(c) \) be an \( n \)-sphere of radius \( r \) smoothly embedded in the Euclidean space \( E^{n+1} \) centered at \( c \in E^{n+1} \). We say that \((\varphi_1, \ldots, \varphi_n)\) define **spherical coordinates** on the open subset \( A \subset S^n_r(c) \) for the Cartesian coordinates \((x_1, \ldots, x_{n+1})\) on \( E^{n+1} \) (not necessarily centered at \( c \)) if

\[
\begin{align*}
x_1 \circ \iota|_A & = r \cos(\varphi_n) \sin(\varphi_{n-1}) \cdots \sin(\varphi_1) + x_1(c), \\
x_1 \circ \iota|_A & = r \sin(\varphi_n) \sin(\varphi_{n-1}) \cdots \sin(\varphi_1) + x_1(c), \\
x_2 \circ \iota|_A & = r \cos(\varphi_{n-1}) \sin(\varphi_{n-2}) \cdots \sin(\varphi_1) + x_2(c), \\
x_3 \circ \iota|_A & = r \cos(\varphi_{n-2}) \sin(\varphi_{n-3}) \cdots \sin(\varphi_1) + x_3(c), \\
& \quad \vdots \\
x_{n+1} \circ \iota|_A & = r \cos(\varphi_1) + x_{n+1}(c),
\end{align*}
\]

where \( \varphi_i \) has range \((0, \pi)\) for \( i = 1, \ldots, n-1 \) and \( \varphi_n \) lies in \((-\pi, \pi)\). Notice that the above expressions together with the ranges of the \( \varphi_i \) fully determine the open set \( A \).
Note: Notice that our convention slightly differs from the one given on the Wikipedia article N-sphere#Spherical_coordinates. The definition above ensures that the conventions for the most common cases $n = 1$ and $n = 2$ are maintained.

EXAMPLES:

The spherical coordinates on a 2-sphere follow the common conventions:

```python
sage: S2 = manifolds.Sphere(2)
sage: spher = S2.spherical_coordinates(); spher
Chart (A, (theta, phi))
```

The coordinate range of spherical coordinates:

```python
sage: spher.coord_range()
theta: (0, pi); phi: [-pi, pi] (periodic)
```

Spherical coordinates do not cover the 2-sphere entirely:

```python
sage: A = spher.domain(); A
Open subset A of the 2-sphere $S^2$ of radius 1 smoothly embedded in the Euclidean space $E^3$
```

The embedding of a 2-sphere in Euclidean space via spherical coordinates:

```python
sage: S2.embedding().display()  
iota: S^2 → E^3  
on A: (theta, phi) ↦ (x, y, z) =
(cos(phi)*sin(theta), sin(phi)*sin(theta), cos(theta))
```

Now, consider spherical coordinates on a 3-sphere:

```python
sage: S3 = manifolds.Sphere(3)
sage: spher = S3.spherical_coordinates(); spher
Chart (A, (chi, theta, phi))
sage: S3.embedding().display()  
iota: S^3 → E^4  
on A: (chi, theta, phi) ↦ (x1, x2, x3, x4) =
(cos(phi)*sin(chi)*sin(theta), sin(chi)*sin(phi)*sin(theta), cos(theta)*sin(chi), cos(chi))
```

By convention, the last coordinate is periodic:

```python
sage: spher.coord_range()
chi: (0, pi); theta: (0, pi); phi: [-pi, pi] (periodic)
```

`spheric_coordinates (pole='north', names=None)`

Return stereographic coordinates given by the stereographic projection of `self` w.r.t. to a given pole.

INPUT:

- `pole` - (default: 'north') the pole determining the stereographic projection; possible options are 'north' and 'south'
names – (default: None) must be a tuple containing the coordinate symbols (this guarantees the usage of the shortcut operator <, >)

OUTPUT:

• the chart of stereographic coordinates w.r.t. to the given pole, as an instance of *RealDiffChart*

Let $S^n_r(c)$ be an $n$-sphere of radius $r$ smoothly embedded in the Euclidean space $E^{n+1}$ centered at $c \in E^{n+1}$. We denote the north pole of $S^n_r(c)$ by NP and the south pole by SP. These poles are uniquely determined by the requirement

$$x(\iota(NP)) = (0, \ldots, 0, r) + x(c),$$
$$x(\iota(SP)) = (0, \ldots, 0, -r) + x(c).$$

The coordinates $(y_1, \ldots, y_n)$ $(y'_1, \ldots, y'_n)$ respectively define stereographic coordinates on $S^n_r(c)$ for the Cartesian coordinates $(x_1, \ldots, x_{n+1})$ on $E^{n+1}$ if they arise from the stereographic projection from $\iota(NP)$ ($\iota(SP)$) to the hypersurface $x_n = x_n(c)$. In concrete formulas, this means:

$$x \circ \iota|_{S^n_r(c) \setminus \{NP\}} = \left(\frac{2y_1r^2}{r^2 + \sum_{i=1}^{n} y_i^2}, \ldots, \frac{2y_nr^2}{r^2 + \sum_{i=1}^{n} y_i^2}, \frac{r \sum_{i=1}^{n} y_i^2 - r^3}{r^2 + \sum_{i=1}^{n} y_i^2} \right) + x(c),$$
$$x \circ \iota|_{S^n_r(c) \setminus \{SP\}} = \left(\frac{2y'_1r^2}{r^2 + \sum_{i=1}^{n} y'_i^2}, \ldots, \frac{2y'_n r^2}{r^2 + \sum_{i=1}^{n} y'_i^2}, \frac{r^3 - r \sum_{i=1}^{n} y'_i^2}{r^2 + \sum_{i=1}^{n} y'_i^2} \right) + x(c).$$

EXAMPLES:

Initialize a 1-sphere centered at $(1, 0)$ in the Euclidean plane using the shortcut operator:

```
sage: E2 = EuclideanSpace(2)
sage: c = E2.point((1,0), name = 'c')
sage: S1.<a> = E2.sphere(center=c, coordinates = 'stereographic'); S1
1-sphere S^1(c) of radius 1 smoothly embedded in the Euclidean plane E^2 centered at the Point c
```

By default, the shortcut variables belong to the stereographic projection from the north pole:

```
sage: S1.coordinate_charts('stereographic')
[Chart (S^1(c)-{NP}, (a,)), Chart (S^1(c)-{SP}, (ap,))]
sage: S1.embedding().display()
iota: S^1(c) → E^2
on S^1(c)-{NP}: (a, y) = (2*a/(a^2 + 1) + 1, (a^2 - 1)/(a^2 + 1))
on S^1(c)-{SP}: (ap, y) = (2*ap/(ap^2 + 1) + 1, -(ap^2 - 1)/(ap^2 + 1))
```

Initialize a 2-sphere from scratch:

```
sage: S2 = manifolds.Sphere(2)
sage: S2.atlas()
[Chart (A, (theta, phi))]
```

In the previous block, the stereographic coordinates have not been initialized. This happens subsequently with the invocation of *stereographic_coordinates*:

```
sage: stereoS.<u,v> = S2.stereographic_coordinates(pole='south')
sage: S2.coordinate_charts('stereographic')
[Chart (S^2-{NP}, (up, vp)), Chart (S^2-{SP}, (u, v))]```

If not specified by the user, the default coordinate names are given by $(y_1, \ldots, y_n)$ and $(y'_1, \ldots, y'_n)$ respectively:
sage: S3 = manifolds.Sphere(3, coordinates='stereographic')
sage: S3.stereographic_coordinates(pole='north')
Chart (S^3-{NP}, (y1, y2, y3))
sage: S3.stereographic_coordinates(pole='south')
Chart (S^3-{SP}, (yp1, yp2, yp3))

3.2.3 Operators for vector calculus

This module defines the following operators for scalar, vector and tensor fields on any pseudo-Riemannian manifold (see `pseudo_riemannian`), and in particular on Euclidean spaces (see `euclidean`):

- **grad()**: gradient of a scalar field
- **div()**: divergence of a vector field, and more generally of a tensor field
- **curl()**: curl of a vector field (3-dimensional case only)
- **laplacian()**: Laplace-Beltrami operator acting on a scalar field, a vector field, or more generally a tensor field
- **dalembertian()**: d’Alembert operator acting on a scalar field, a vector field, or more generally a tensor field, on a Lorentzian manifold

All these operators are implemented as functions that call the appropriate method on their argument. The purpose is to allow one to use standard mathematical notations, e.g. to write `curl(v)` instead of `v.curl()`.

Note that the **norm()** operator is defined in the module `functional`.

See also:

Examples 1 and 2 in `euclidean` for examples involving these operators in the Euclidean plane and in the Euclidean 3-space.

AUTHORS:

- Eric Gourgoulhon (2018): initial version

```python
sage.manifolds.operators.curl(vector)
```

Curl operator.

The **curl** of a vector field `v` on an orientable pseudo-Riemannian manifold `(M, g)` of dimension 3 is the vector field defined by

\[
\text{curl } v = (\ast (\text{d} v^\flat))^\sharp
\]

where `v^\flat` is the 1-form associated to `v` by the metric `g` (see `down()`), `\ast (\text{d} v^\flat)` is the Hodge dual with respect to `g` of the 2-form `\text{d} v^\flat` (exterior derivative of `v^\flat`) (see `hodge_dual()`), and `(\ast (\text{d} v^\flat))^\sharp` is corresponding vector field by `g`-duality (see `up()`).

An alternative expression of the curl is

\[
(\text{curl } v)^i = \epsilon^{ijk} \nabla_j v_k
\]

where `\nabla` is the Levi-Civita connection of `g` (cf. `LeviCivitaConnection`) and `\epsilon` the volume 3-form (Levi-Civita tensor) of `g` (cf. `volume_form()`)

**INPUT:**

- `vector` – vector field on an orientable 3-dimensional pseudo-Riemannian manifold, as an instance of `VectorField`

**OUTPUT:**
- instance of `VectorField` representing the curl of `vector`

EXAMPLES:

Curl of a vector field in the Euclidean 3-space:

```python
sage: E.<x,y,z> = EuclideanSpace()
sage: v = E.vector_field(sin(y), sin(x), 0, name='v')
sage: v.display()
v = sin(y) e_x + sin(x) e_y
sage: from sage.manifolds.operators import curl
data: curl(v); s
Vector field curl(v) on the Euclidean space E^3
data: s.display()
curl(v) = (cos(x) - cos(y)) e_z
data: s[:]
[0, 0, cos(x) - cos(y)]
```

See the method `curl()` of `VectorField` for more details and examples.

```python
sage.manifolds.operators.dalembertian(field)
```

d’Alembert operator.

The d’Alembert operator or d’Alembertian on a Lorentzian manifold \((M, g)\) is nothing but the Laplace-Beltrami operator:

\[
\Box = \nabla_i \nabla^i = g^{ij} \nabla_i \nabla_j
\]

where \(\nabla\) is the Levi-Civita connection of the metric \(g\) (cf. `LeviCivitaConnection`) and \(\nabla_i := g^{ij} \nabla_j\)

INPUT:

- `field`-- a scalar field \(f\) (instance of `DiffScalarField`) or a tensor field \(f\) (instance of `TensorField`) on a pseudo-Riemannian manifold

OUTPUT:

- \(\Box f\), as an instance of `DiffScalarField` or of `TensorField`

EXAMPLES:

d’Alembertian of a scalar field in the 2-dimensional Minkowski spacetime:

```python
sage: M = Manifold(2, 'M', structure='Lorentzian')
sage: X.<t,x> = M.chart()
sage: g = M.metric()
sage: g[0,0], g[1,1] = -1, 1
sage: f = M.scalar_field((x-t)^3 + (x+t)^2, name='f')
sage: from sage.manifolds.operators import dalembertian
data: dalembertian(f); Df
Scalar field Box(f) on the 2-dimensional Lorentzian manifold M
data: Df.display()
Box(f): M → ℝ
t, x → 0
```

See the method `dalembertian()` of `DiffScalarField` and the method `dalembertian()` of `TensorField` for more details and examples.

```python
sage.manifolds.operators.div(tensor)
```

Divergence operator.

3.2. Euclidean Spaces and Vector Calculus 891
Let $t$ be a tensor field of type $(k, 0)$ with $k \geq 1$ on a pseudo-Riemannian manifold $(M, g)$. The divergence of $t$ is the tensor field of type $(k - 1, 0)$ defined by

$$(\text{div } t)^{a_1 \ldots a_{k-1}} = \nabla_i t^{a_1 \ldots a_{k-1} i} = (\nabla t)^{a_1 \ldots a_{k-1} i}$$

where $\nabla$ is the Levi-Civita connection of $g$ (cf. \texttt{LeviCivitaConnection}).

Note that the divergence is taken on the last index of the tensor field. This definition is extended to tensor fields of type $(k, l)$ with $k \geq 0$ and $l \geq 1$, by raising the last index with the metric $g$: $\text{div } t$ is then the tensor field of type $(k, l - 1)$ defined by

$$(\text{div } t)^{a_1 \ldots a_k}_{b_1 \ldots b_{l-1}} = \nabla_i (g^{ij} t^{a_1 \ldots a_k}_{b_1 \ldots b_{l-1} j}) = (\nabla t)^{a_1 \ldots a_k}_{b_1 \ldots b_{l-1} i}$$

where $t^i$ is the tensor field deduced from $t$ by raising the last index with the metric $g$ (see \texttt{up()}).

**INPUT:**

- tensor – tensor field $t$ on a pseudo-Riemannian manifold $(M, g)$, as an instance of \texttt{TensorField} (possibly via one of its derived classes, like \texttt{VectorField})

**OUTPUT:**

- the divergence of tensor as an instance of either \texttt{DiffScalarField} if $(k, l) = (1, 0)$ (tensor is a vector field) or $(k, l) = (0, 1)$ (tensor is a 1-form) or of \texttt{TensorField} if $k + l \geq 2$

**EXAMPLES:**

Divergence of a vector field in the Euclidean plane:

```
sage: E.<x,y> = EuclideanSpace()
sage: v = E.vector_field(cos(x*y), sin(x*y), name='v')
sage: v.display()
v = cos(x*y) e_x + sin(x*y) e_y
sage: from sage.manifolds.operators import div
sage: s = div(v); s
Scalar field div(v) on the Euclidean plane $E^2$
sage: s.display()
  (x, y) ↦ x*cos(x*y) - y*sin(x*y)
sage: s.expr()
x*cos(x*y) - y*sin(x*y)
```

See the method \texttt{divergence()} of \texttt{TensorField} for more details and examples.

sage.manifolds.operators.grad(scalar)

Gradient operator.

The gradient of a scalar field $f$ on a pseudo-Riemannian manifold $(M, g)$ is the vector field $\text{grad } f$ whose components in any coordinate frame are

$$(\text{grad } f)^i = g^{ij} \frac{\partial F}{\partial x^j}$$

where the $x^j$’s are the coordinates with respect to which the frame is defined and $F$ is the chart function representing $f$ in these coordinates: $f(p) = F(x^1(p), \ldots, x^n(p))$ for any point $p$ in the chart domain. In other words, the gradient of $f$ is the vector field that is the $g$-dual of the differential of $f$.

**INPUT:**

- scalar – scalar field $f$, as an instance of \texttt{DiffScalarField}

**OUTPUT:**
• instance of \texttt{VectorField} representing \texttt{grad} \( f \)

EXAMPLES:

Gradient of a scalar field in the Euclidean plane:

```
sage: E.<x,y> = EuclideanSpace()
sage: f = E.scalar_field(sin(x*y), name='f')
sage: from sage.manifolds.operators import grad
sage: grad(f)
Vector field grad(f) on the Euclidean plane \( E^2 \)
sage: grad(f).display()
grad(f) = y \text{cos}(x*y) \, e_x + x \text{cos}(x*y) \, e_y
sage: grad(f)[:]
[y \text{cos}(x*y), x \text{cos}(x*y)]
```

See the method \texttt{gradient()} of \texttt{DiffScalarField} for more details and examples.

```
sage.manifolds.operators.laplacian(field)
```

Laplace-Beltrami operator.

The \emph{Laplace-Beltrami operator} on a pseudo-Riemannian manifold \((M,g)\) is the operator

\[ \Delta = \nabla^i \nabla_i = g^{ij} \nabla_i \nabla_j \]

where \( \nabla \) is the Levi-Civita connection of the metric \( g \) (cf. \texttt{LeviCivitaConnection}) and \( \nabla^i := g^{ij} \nabla_j \)

INPUT:

• \texttt{field} – a scalar field \( f \) (instance of \texttt{DiffScalarField}) or a tensor field \( f \) (instance of \texttt{TensorField}) on a pseudo-Riemannian manifold

OUTPUT:

• \( \Delta f \), as an instance of \texttt{DiffScalarField} or of \texttt{TensorField}

EXAMPLES:

Laplacian of a scalar field on the Euclidean plane:

```
sage: E.<x,y> = EuclideanSpace()
sage: f = E.scalar_field(sin(x*y), name='f')
sage: from sage.manifolds.operators import laplacian
sage: Df = laplacian(f); Df
Scalar field Delta(f) on the Euclidean plane \( E^2 \)
sage: Df.display()
Delta(f): \( E^2 \to \mathbb{R} \)
\((x, y) \mapsto -(x^2 + y^2) \text{sin}(x*y)\)
sage: Df.expr()
-(x^2 + y^2) \text{sin}(x*y)
```

The Laplacian of a scalar field is the divergence of its gradient:

```
sage: from sage.manifolds.operators import div, grad
sage: Df == div(grad(f))
True
```

See the method \texttt{laplacian()} of \texttt{DiffScalarField} and the method \texttt{laplacian()} of \texttt{TensorField} for more details and examples.
3.3 Pseudo-Riemannian Metrics and Degenerate Metrics

The class `PseudoRiemannianMetric` implements pseudo-Riemannian metrics on differentiable manifolds over \( \mathbb{R} \). The derived class `PseudoRiemannianMetricParal` is devoted to metrics with values on a parallelizable manifold.

The class `DegenerateMetric` implements degenerate (or null or lightlike) metrics on differentiable manifolds over \( \mathbb{R} \). The derived class `DegenerateMetricParal` is devoted to metrics with values on a parallelizable manifold.

AUTHORS:

- Florentin Jaffredo (2018): series expansion for the inverse metric
- Hans Fotsing Tetsing (2019): degenerate metrics
- Marius Gerbershagen (2022): compute volume forms with contravariant indices only as needed

REFERENCES:

- [KN1963]
- [Lee1997]
- [ONe1983]
- [DB1996]
- [DS2010]

```python
class sage.manifolds.differentiable.metric.DegenerateMetric(vector_field_module, name, signature=None, latex_name=None)
```

Bases: `TensorField`

Degenerate (or null or lightlike) metric with values on an open subset of a differentiable manifold.

An instance of this class is a field of degenerate symmetric bilinear forms (metric field) along a differentiable manifold \( U \) with values on a differentiable manifold \( M \) over \( \mathbb{R} \), via a differentiable mapping \( \Phi : U \to M \). The standard case of a degenerate 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 \( \mathbb{R} \)).

A degenerate metric \( g \) is a field on \( U \), such that at each point \( p \in U \), \( g(p) \) is a bilinear map of the type:

\[
g(p) : T_qM \times T_qM \to \mathbb{R}
\]

where \( T_qM \) stands for the tangent space to the manifold \( M \) at the point \( q = \Phi(p) \), such that \( g(p) \) is symmetric: \( \forall (u, v) \in T_qM \times T_qM \), \( g(p)(v, u) = g(p)(u, v) \) and degenerate: \( \exists v \in T_qM ; \ g(p)(u, v) = 0 \ \forall u \in T_qM \).

**Note:** If \( M \) is parallelizable, the class `DegenerateMetricParal` 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 tuple: \( S = (n_+, n_-, n_0) \), where \( n_+ \) (resp. \( n_- \), resp. \( n_0 \)) is the number of positive terms (resp. negative terms, resp. zero terms) in any diagonal writing of the metric components; if `signature` is not provided, \( S \) is set to \( (\text{ndim} - 1, 0, 1) \), being \( \text{ndim} \) the manifold’s dimension
• \texttt{latex\_name} – (default: None) LaTeX symbol to denote the metric; if None, it is formed from \texttt{name}

EXAMPLES:

Lightlike cone:

\begin{verbatim}
\texttt{sage: M = Manifold(3, 'M'); X.<x,y,z> = M.chart()}
\texttt{sage: g = M.metric('g', signature=(2,0,1)); g}
\end{verbatim}

degenerate metric \(g\) on the 3-dimensional differentiable manifold \(M\)

\texttt{sage: det(g)}
Scalar field zero on the 3-dimensional differentiable manifold \(M\)

\texttt{sage: g.parent()}
Free module \(T^{(0,2)}(M)\) of type-(0,2) tensors fields on the 3-dimensional differentiable manifold \(M\)

\texttt{sage: g[0,0], g[0,1], g[0,2] = (y^2 + z^2)/(x^2 + y^2 + z^2), \}
.....: - x*y/(x^2 + y^2 + z^2), - x*z/(x^2 + y^2 + z^2)
\texttt{sage: g[1,1], g[1,2], g[2,2] = (x^2 + z^2)/(x^2 + y^2 + z^2), \}
.....: - y*z/(x^2 + y^2 + z^2), (x^2 + y^2)/(x^2 + y^2 + z^2)
\texttt{sage: g.disp()}
g = (y^2 + z^2)/(x^2 + y^2 + z^2) dx⊗dx - x*y/(x^2 + y^2 + z^2) dx⊗dy
- x*z/(x^2 + y^2 + z^2) dx⊗dz - x*y/(x^2 + y^2 + z^2) dy⊗dx
+ (x^2 + z^2)/(x^2 + y^2 + z^2) dy⊗dy - y*z/(x^2 + y^2 + z^2) dy⊗dz
- x*z/(x^2 + y^2 + z^2) dz⊗dx - y*z/(x^2 + y^2 + z^2) dz⊗dy
+ (x^2 + y^2)/(x^2 + y^2 + z^2) dz⊗dz
\end{verbatim}

The position vector is a lightlike vector field:

\begin{verbatim}
\texttt{sage: v = M.vector\_field()}
\texttt{sage: v[0], v[1], v[2] = x, y, z}
\texttt{sage: g(v, v).disp()}
\end{verbatim}

\texttt{det()}  
Determinant of a degenerate metric is always ‘0’

EXAMPLES:

\begin{verbatim}
\texttt{sage: S = Manifold(2, 'S')}
\texttt{sage: g = S.metric('g', signature=[0,1,1])}
\texttt{sage: g.determinant()}
Scalar field zero on the 2-dimensional differentiable manifold \(S\)
\end{verbatim}

\texttt{determinant()}  
Determinant of a degenerate metric is always ‘0’

EXAMPLES:

\begin{verbatim}
\texttt{sage: S = Manifold(2, 'S')}
\texttt{sage: g = S.metric('g', signature=[0,1,1])}
\texttt{sage: g.determinant()}
Scalar field zero on the 2-dimensional differentiable manifold \(S\)
\end{verbatim}

\texttt{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 \texttt{DifferentiableManifold})

• dest_map – (default: None) destination map \( \Phi : U \rightarrow V \), where \( V \) is a subdomain of \texttt{self}.

\_codomain (type: \texttt{DiffMap}) If None, the restriction of \texttt{self\_vmodule\_dest\_map} to \( U \) is used.

OUTPUT:

• instance of \texttt{DegenerateMetric} representing the restriction.

EXAMPLES:

```sage
M = Manifold(5, 'M')
g = M.metric('g', signature=(3,1,1))
U = M.open_subset('U')
g.restrict(U)
```

degenerate metric \( g \) on the Open subset \( U \) of the 5-dimensional differentiable manifold \( M \)

```
g.restrict(U).signature()
(3, 1, 1)
```

See the top documentation of \texttt{DegenerateMetric} for more examples.

\texttt{set}(\texttt{symbiform})

Defines the metric from a field of symmetric bilinear forms

INPUT:

• \texttt{symbiform} – instance of \texttt{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')
U = M.open_subset('U') ; V = M.open_subset('V')
M.declare_union(U,V) # M is the union of U and V
c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart()
xy_to_uv = c_xy.transition_map(c_uv, (x + y, x - y), intersection_name='W',
....: restrictions1= x>0, restrictions2= u+v>0)
uv_to_xy = xy_to_uv.inverse()
w = U.intersection(V)
ceU = c_xy.frame() ; eV = c_uv.frame()
h = M.sym_bilin_form_field(name='h')
h[eU,0,0], h[eU,0,1], h[eU,1,1] = 1+x, x*y, 1-y
h.add_comp_by_continuation(eV, W, c_uv)
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*v dv⊗du + (-1/8*u^2 + 1/8*v^2 + 1/4*v + 1/2) dv⊗dv
```

```sage
g = M.metric('g')
g.set(h)
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*v dv⊗du + (-1/8*u^2 + 1/8*v^2 + 1/4*v + 1/2) dv⊗dv
```
signature()  
Signature of the metric.

OUTPUT:
• signature of a degenerate metric is defined as the tuple \((n_+, n_-, n_0)\), where \(n_+\) (resp. \(n_-\), resp. \(n_0\)) is the number of positive terms (resp. negative terms, resp. zero terms) eigenvalues

EXAMPLES:
Signatures on a 3-dimensional manifold:

```python
sage: M = Manifold(3, 'M')
sage: g = M.metric('g', signature=(1,1,1))
sage: g.signature()
(1, 1, 1)
sage: M = Manifold(3, 'M', structure='degenerate_metric')
sage: g = M.metric()
sage: g.signature()
(0, 2, 1)
```

class sage.manifolds.differentiable.metric.DegenerateMetricParal(vector_field_module, name, signature=None, latex_name=None)

Bases: DegenerateMetric, TensorFieldParal

Degenerate (or null or lightlike) metric with values on an open subset of a differentiable manifold.

An instance of this class is a field of degenerate symmetric bilinear forms (metric field) along a differentiable manifold \(U\) with values on a differentiable manifold \(M\) over \(\mathbb{R}\), via a differentiable mapping \(\Phi : U \rightarrow M\). The standard case of a degenerate 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 \(\mathbb{R}\)).

A **degenerate metric** \(g\) is a field on \(U\), such that at each point \(p \in U\), \(g(p)\) is a bilinear map of the type:

\[
g(p) : T_qM \times T_qM \rightarrow \mathbb{R}
\]

where \(T_qM\) stands for the tangent space to the manifold \(M\) at the point \(q = \Phi(p)\), such that \(g(p)\) is symmetric: \(\forall (v, u) \in T_qM \times T_qM\), \(g(p)(v, u) = g(p)(u, v)\) and degenerate: \(\exists v \in T_qM ; g(p)(u, v) = 0 \ \forall u \in T_qM\).

**Note:** If \(M\) is not parallelizable, the class `DegenerateMetric` 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 tuple: \(S = (n_+, n_-, n_0)\), where \(n_+\) (resp. \(n_-\), resp. \(n_0\)) is the number of positive terms (resp. negative terms, resp. zero terms) in any diagonal writing of the metric components; if `signature` is not provided, \(S\) is set to \((\text{ndim} - 1, 0, 1)\), being \(\text{ndim}\) the manifold’s dimension
• `latex_name` – (default: None) LaTeX symbol to denote the metric; if None, it is formed from `name`

EXAMPLES:
Lightlike cone:
The position vector is a lightlike vector field:

```
sage: v = M.vector_field()
sage: v[0], v[1], v[2] = x, y, z
sage: g(v, v).disp()
M → ℝ
(x, y, z) ↦ 0
```

**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.

**OUTPUT:**

- instance of *DegenerateMetric* representing the restriction.

**EXAMPLES:**

```
sage: M = Manifold(5, 'M')
sage: g = M.metric('g', signature=(3,1,1))
sage: U = M.open_subset('U')
sage: g.restrict(U)
degenerate metric on the Open subset U of the 5-dimensional differentiable
→ manifold M
sage: g.restrict(U).signature()((3, 1, 1)
```

See the top documentation of *DegenerateMetric* for more examples.

**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 parallelizable 3-dimensional manifold:

```python
sage: M = Manifold(3, 'M', start_index =1);
sage: X.<x,y,z> = M.chart()
sage: dx, dy = X.coframe()[1], X.coframe()[2]
sage: b = dx*dx + dy*dy
sage: g = M.metric(g, signature =(1,1,1)); g
degenerate metric g on the 3-dimensional differentiable manifold M
sage: g.set(b)
sage: g.display()
g = dx⊗dx + dy⊗dy
```

```python
class sage.manifolds.differentiable.metric.PseudoRiemannianMetric(vector_field_module, name,
signature=None, latex_name=None)
```

Bases: 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 $\mathbb{R}$, via a differentiable mapping $\Phi : U \to M$. 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 $\mathbb{R}$).

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:

$$g(p) : T_qM \times T_qM \longrightarrow \mathbb{R}$$

where $T_qM$ stands for the tangent space to the manifold $M$ at the point $q = \Phi(p)$, such that $g(p)$ is symmetric: $\forall (u,v) \in T_qM \times T_qM$, $g(p)(v,u) = g(p)(u,v)$ and nondegenerate: $\forall v \in T_qM$, $g(p)(u,v) = 0 \implies u = 0$.

Note: If $M$ is parallelizable, the class PseudoRiemannianMetricParal should be used instead.

INPUT:
• vector_field_module – module $\mathfrak{X}(U, \Phi)$ 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:
Let us construct the standard metric on the sphere $S^2$, described in terms of stereographic coordinates, from the North pole (open subset $U$) and from the South pole (open subset $V$):
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: eU = c_xy.frame() ; eV = c_uv.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 \( U \):

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 coordinates \((u, v)\) are obtained by applying (i) the tensor transformation law on \( W = U ∩ V \) and (ii) some analytical continuation:

sage: W = U.intersection(V)
sage: g.add_comp_by_continuation(eV, W, chart=c_uv)
sage: g.apply_map(factor, frame=eV, keep_other_components=True)  # for a nicer...
    →display
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 vector fields, yielding a scalar field:

```python
sage: a = M.vector_field({eU: [x, 2+y]}, name='a')
sage: a.add_comp_by_continuation(eV, W, chart=c_uv)
sage: b = M.vector_field({eU: [-y, x]}, name='b')
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
don U: (x, y) ↦ 8*x/(x^4 + y^4 + 2*(x^2 + 1)*y^2 + 2*x^2 + 1)
don 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:

```python
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)
inv_g = (1/4*x^4 + 1/4*y^4 + 1/2*(x^2 + 1)*y^2 + 1/2*x^2 + 1/4) ∂/∂x⊗∂/∂x
     + (1/4*x^4 + 1/4*y^4 + 1/2*(x^2 + 1)*y^2 + 1/2*x^2 + 1/4) ∂/∂y⊗∂/∂y
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) ∂/∂u⊗∂/∂u
     + (1/4*u^4 + 1/4*v^4 + 1/2*(u^2 + 1)*v^2 + 1/2*u^2 + 1/4) ∂/∂v⊗∂/∂v
```

We have:

```python
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
```

To get the volume form (Levi-Civita tensor) associated with $g$, we have first to define an orientation on $S^2$. The standard orientation is that in which $eV$ is right-handed; indeed, once supplemented by the outward unit normal, $eV$ give birth to a right-handed frame with respect to the standard orientation of the ambient Euclidean space $E^3$. With such an orientation, $eU$ is then left-handed and in order to define an orientation on the whole of $S^2$, we introduce a vector frame on $U$ by swapping $eU$'s vectors:

```python
sage: f = U.vector_frame('f', (eU[2], eU[1]))
sage: M.set_orientation([eV, f])
```

We have then, factorizing the components for a nicer display:

```python
sage: eps = g.volume_form() ; eps
2-form eps_g on the 2-dimensional differentiable manifold S^2
sage: eps.apply_map(factor, frame=eU, keep_other_components=True)
sage: eps.apply_map(factor, frame=eV, keep_other_components=True)
sage: eps.display(eU)
eps_g = -4/(x^2 + y^2 + 1)^2 dx∧dy
sage: eps.display(eV)
eps_g = 4/(u^2 + v^2 + 1)^2 du∧dv
```
The unique non-trivial component of the volume form is, up to a sign depending of the chosen orientation, nothing but the square root of the determinant of $g$ in the corresponding frame:

```python
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 $g$:

```python
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 Christoffels symbols $\Gamma^i_{jk}$ associated with some coordinates:

```python
sage: g.christoffel_symbols(c_xy)
3-indices components w.r.t. Coordinate frame (U, (∂/∂x,∂/∂y)), 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 Christoffels symbols are nothing but the connection coefficients w.r.t. the coordinate frame:

```python
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 $g$:

```python
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 $g$:

```python
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) ∂/∂x⊗dy⊗dx⊗dy
```

(continues on next page)
- $\frac{4}{(x^4 + y^4 + 2*(x^2 + 1)*y^2 + 2*x^2 + 1)} \partial/\partial x \otimes dy \otimes dy \otimes dx$
- $\frac{4}{(x^4 + y^4 + 2*(x^2 + 1)*y^2 + 2*x^2 + 1)} \partial/\partial y \otimes dx \otimes dx \otimes dy$
- $\frac{4}{(x^4 + y^4 + 2*(x^2 + 1)*y^2 + 2*x^2 + 1)} \partial/\partial y \otimes dx \otimes dy \otimes dx$

sage: riem.display(eV)
Riem(g) = 4/(u^4 + v^4 + 2*(u^2 + 1)*v^2 + 2*u^2 + 1) ∂/∂u ⊗ dv ⊗ du ⊗ dv
- 4/(u^4 + v^4 + 2*(u^2 + 1)*v^2 + 2*u^2 + 1) ∂/∂u ⊗ dv ⊗ dv ⊗ du
- 4/(u^4 + v^4 + 2*(u^2 + 1)*v^2 + 2*u^2 + 1) ∂/∂v ⊗ du ⊗ du ⊗ dv
+ 4/(u^4 + v^4 + 2*(u^2 + 1)*v^2 + 2*u^2 + 1) ∂/∂v ⊗ 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): $S^2 \to \mathbb{R}$
on U: $(x, y) \mapsto 2$
on V: $(u, v) \mapsto 2$

In dimension 2, the Riemann tensor can be expressed entirely in terms of the Ricci scalar $r$:

$$R^i_{jlk} = \frac{r}{2} (\delta^i_{[kl} g_{jl]} - \delta^i_{l} g_{jk})$$

This formula can be checked here, with the r.h.s. rewritten as $-rg_{[j} \delta^i_{l]}$:

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 $R^3$ 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
\begin{align*}
g &= dr \otimes dr + r^2 d\theta \otimes d\theta + r^2 \sin(\theta)^2 d\phi \otimes d\phi
\end{align*}

sage: Gam = g.christoffel_symbols() ; Gam
3-indices components w.r.t. Coordinate frame (U, (∂/∂r,∂/∂θ,∂/∂φ)),
with symmetry on the index positions (1, 2)

sage: type(Gam)
<class 'sage.tensor.modules.comp.CompWithSym'>

sage: Gam[:]
\begin{align*}
\Gamma^r_{\theta\theta} &= -r \\
\Gamma^r_{\phi\phi} &= -r \sin(\theta)^2 \\
\Gamma^\theta_{r\theta} &= 1/r \\
\Gamma^\theta_{\phi\phi} &= -\cos(\theta) \sin(\theta) \\
\Gamma^\phi_{r\phi} &= 1/r \\
\Gamma^\phi_{\theta\phi} &= \cos(\theta)/\sin(\theta)
\end{align*}

Note that a better display of the Christoffel symbols is provided by the method christoffel_symbols_display():

\begin{align*}
\text{christoffel_symbols_display}():
\begin{align*}
\Gamma^r_{\theta\theta} &= -r \\
\Gamma^r_{\phi\phi} &= -r \sin(\theta)^2 \\
\Gamma^\theta_{r\theta} &= 1/r \\
\Gamma^\theta_{\phi\phi} &= -\cos(\theta) \sin(\theta) \\
\Gamma^\phi_{r\phi} &= 1/r \\
\Gamma^\phi_{\theta\phi} &= \cos(\theta)/\sin(\theta)
\end{align*}
\end{align*}

\textbf{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 $\mathbb{R}^3$ with respect to spherical coordinates:

```python
sage: M = Manifold(3, 'R3', r'\mathbb{R}^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(rr:(0,+oo) th:(0,pi):θ ph:(0,2*pi):φ)
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...
\mathrm{~d}r \otimes \mathrm{d}r + r^2 \mathrm{~d}\theta \otimes \mathrm{d}\theta + r^2 \sin(\theta)^2 \mathrm{~d}\phi \otimes \mathrm{d}\phi
sage: g.christoffel_symbols_display()
\Gamma^r_{\theta \theta} = -r
\Gamma^r_{\phi \phi} = -r \sin(\theta)^2
\Gamma^\theta_{r \theta} = 1/r
\Gamma^\theta_{\phi \phi} = -\cos(\theta) \sin(\theta)
\Gamma^\phi_{r \phi} = 1/r
\Gamma^\phi_{\phi \theta} = \cos(\theta)/\sin(\theta)
```

To list all nonzero Christoffel symbols, including those that can be deduced by symmetry, use `only_nonredundant=False`:

```python
sage: g.christoffel_symbols_display(only_nonredundant=False)
\Gamma^r_{\theta \theta} = -r
\Gamma^r_{\phi \phi} = -r \sin(\theta)^2
\Gamma^\theta_{r \theta} = 1/r
\Gamma^\theta_{\phi \phi} = -\cos(\theta) \sin(\theta)
\Gamma^\phi_{r \phi} = 1/r
\Gamma^\phi_{\phi \theta} = \cos(\theta)/\sin(\theta)
```

Listing all Christoffel symbols (except those that can be deduced by symmetry), including the vanishing one:

```python
sage: g.christoffel_symbols_display(only_nonzero=False)
\Gamma^r_{r r} = 0
\Gamma^r_{r r} = 0
\Gamma^r_{r \theta} = 0
\Gamma^r_{r \phi} = 0
\Gamma^r_{\theta r} = -r
\Gamma^r_{\phi r} = 0
\Gamma^r_{\phi \phi} = -r \sin(\theta)^2
\Gamma^\theta_{r r} = 0
\Gamma^\theta_{r \theta} = 1/r
```

(continues on next page)
Gam^th_r,ph = 0
Gam^th_th,th = 0
Gam^th_th,ph = 0
Gam^th_ph,ph = -\cos(th)\cdot\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\cdot\sin(th)^2
Gam^2_12 = 1/r
Gam^2_33 = -\cos(th)\cdot\sin(th)
Gam^3_13 = 1/r
Gam^3_23 = \cos(th)/\sin(th)

connection (name=None, latex_name=None, init_coef=True)
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 is formed from the symbol ∇ and the metric symbol

• init_coef – (default: True) determines whether the connection coefficients are initialized, as Christoffel symbols in the top charts of the domain of self (i.e. disregarding the subcharts)

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)

Let us use spherical coordinates on \( \mathbb{R}^3 \):

sage: U = M.open_subset('U') # the complement of the half-plane (y=0, x>0)
sage: c_spher.<r,th,ph> = U.chart(rr:(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\cdot\sin(th))^2 # the Euclidean metric
go: 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
\[
\begin{align*}
\Gamma_{r\theta,\theta} &= -r \\
\Gamma_{r\phi,\phi} &= -r \sin(\theta)^2 \\
\Gamma_{\theta r,\theta} &= 1/r \\
\Gamma_{\theta r,\theta} &= 1/r \\
\Gamma_{\theta\phi,\phi} &= -\cos(\theta) \sin(\theta) \\
\Gamma_{\phi r,\phi} &= 1/r \\
\Gamma_{\phi\theta,\phi} &= \cos(\theta)/\sin(\theta) \\
\Gamma_{\phi r,\phi} &= 1/r \\
\Gamma_{\phi\phi,\theta} &= \cos(\theta)/\sin(\theta)
\end{align*}
\]

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 \texttt{schouten}):

\[
C_{ijk} = (n - 2) \left( \nabla_k S_{ij} - \nabla_j S_{ik} \right)
\]

INPUT:

- \(\text{name} \) – (default: None) name given to the Cotton conformal tensor; if None, it is set to \(\text{“Cot}(g)\)”, where \(\text{“g}\) is the metric’s name
- \(\text{latex_name} \) – (default: None) LaTeX symbol to denote the Cotton conformal tensor; if None, it is set to \(\text{“mathrm}(\text{Cot}(g))\), where \(\text{“g}\) is the metric’s name

OUTPUT:

- the Cotton conformal tensor \(\text{Cot}\), as an instance of \texttt{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(rrh:(0,+oo):ho th:(0,pi):	heta 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(\theta))^2
sage: g.display() # standard metric on H^3:
g = b^2 \, drh\otimes drh + b^2*\sinh(rh)^2 \, dth\otimes dth
+ b^2*\sin(\theta)^2*\sinhill(rh)^2 \, dph\otimes dph
sage: Cot = g.cotton() ; Cot # long time
Tensor field Cot(g) of type (0,3) on the Open subset U of the
```

(continues on next page)
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)
X.<x,y,z> = M.chart()
g = M.riemannian_metric('g')
g[1,1], g[2,2], g[2,3], g[3,3] = 1, 1+x^2, -x, 1
g.display()
g = dx⊗dx + (x^2 + 1) dy⊗dy - x dy⊗dz - x dz⊗dy + dz⊗dz
CY = g.cotton_york() ; CY
# long time
Tensor field CY(g) of type (0,2) on the 3-dimensional differentiable manifold Nil
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
```

**det** *(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:
Manifolds, Release 10.3

```
sage: M = Manifold(2, 'M', start_index=1)
sage: X.<x,y> = M.chart()
sage: g = M.metric('g')
g[1,1], g[1, 2], g[2, 2] = 1+x, x*y, 1-y
g[1:]
[ 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
```

A shortcut is `det()`:

```
sage: g.det() == g.determinant()
True
```

The notation `det(g)` can be used:

```
sage: det(g) == g.determinant()
True
```

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))

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)  # determinant in M's default frame
True
```

The metric determinant depends on the frame:

```
sage: g.determinant(X.frame()) == g.determinant(Y.frame())
False
```

Using SymPy as symbolic engine:

```
sage: M.set_calculus_method('sympy')
sage: g = M.metric('g')
g[1,1], g[1, 2], g[2, 2] = 1+x, x*y, 1-y
sage: s = g.determinant()  # determinant in M's default frame
sage: s.expr()
-x**2*y**2 + x - y*(x + 1) + 1
```

**determinant** (*frame=None*)

Determinant of the metric components in the specified frame.

3.3. Pseudo-Riemannian Metrics and Degenerate Metrics 909
INPUT:
- \textit{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 \textit{DiffScalarField}

EXAMPLES:

Metric determinant on a 2-dimensional manifold:

\begin{verbatim}
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[:]
\begin{bmatrix}
x + 1 & x*y \\
x*y -y + 1
\end{bmatrix}
sage: s = g.determinant()  # determinant in M's default frame
sage: s.expr()  
-\(x^2*y^2 - (x + 1)*y + x + 1\)
\end{verbatim}

A shortcut is \textbf{det()}:

\begin{verbatim}
sage: g.det() == g.determinant()
True
\end{verbatim}

The notation \textbf{det(g)} can be used:

\begin{verbatim}
sage: det(g) == g.determinant()
True
\end{verbatim}

Determinant in a frame different from the default's one:

\begin{verbatim}
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]
\begin{bmatrix}
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
\end{bmatrix}
\end{verbatim}

\begin{verbatim}
sage: g.determinant(Y.frame()).expr() 
-1/4*x^2*y^2 - 1/4*(x + 1)*y + 1/4*x + 1/4
\end{verbatim}

A chart can be passed instead of a frame:

\begin{verbatim}
sage: g.determinant(X) is g.determinant(X.frame())
True
sage: g.determinant(Y) is g.determinant(Y.frame())
True
\end{verbatim}

The metric determinant depends on the frame:

\begin{verbatim}
sage: g.determinant(X.frame()) == g.determinant(Y.frame())
False
\end{verbatim}
Using SymPy as symbolic engine:

```python
sage: M.set_calculus_method('sympy')
sage: g = M.metric('g')
sage: g[1,1], g[2,2], g[3,3] = 1+x, x*y , 1-y
sage: s = g.determinant()  # determinant in M's default frame
sage: s.expr()
-x**2*y**2 + x - y*(x + 1) + 1
```

### 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 \emph{Hodge dual} with respect to the metric \( g \) is the \((n - p)\)-form \( *A \) defined by

\[
* A_{i_1...i_{n-p}} = \frac{1}{p!} A_{k_1...k_p} \epsilon^{k_1...k_p}_{i_1...i_{n-p}}
\]

where \( n \) is the manifold’s dimension, \( \epsilon \) is the volume \( n \)-form associated with \( g \) (see `volume_form()`) and the indices \( k_1, \ldots, k_p \) are raised with \( g \).

Notice that the hodge star dual requires an orientable manifold with a preferred orientation, see `orientation()` for details.

**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 \( \mathbb{R}^3 \):

```python
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: var('Ax Ay Az')
(Ax, Ay, Az)
sage: a = M.one_form(Ax, Ay, Az, name='A')
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 \( \mathbb{R}^3 \):

```python
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
```

(continues on next page)
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: var('At Ax Ay Az')
(At, Ax, Ay, Az)
sage: a = M.one_form(At, Ax, Ay, Az, name =A)
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, name =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
(continues on next page)
Test of the standard identity

\[ *(A \wedge B) = \epsilon(A^i, B^i, \ldots) \]

where \( A \) and \( B \) are any 1-forms and \( A^i \) and \( B^i \) the vectors associated to them by the metric \( g \) (index raising):

\begin{verbatim}
sage: var('Bt Bx By Bz')
(Bt, Bx, By, Bz)
sage: b = M.one_form(Bt, Bx, By, Bz, name='B')
sage: 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
\end{verbatim}
to this method will return a cached value, even when called with the default value (to enable computation of derived quantities). To reset, use \_del\_derived().

**OUTPUT:**

- instance of `TensorField` with `tensor_type = (2,0)` representing the inverse metric

**EXAMPLES:**

Inverse of the standard metric on the 2-sphere:

```python
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) ∂/∂x⊗∂/∂x
    + (1/4*x^4 + 1/4*y^4 + 1/2*(x^2 + 1)*y^2 + 1/2*x^2 + 1/4) ∂/∂y⊗∂/∂y
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) ∂/∂u⊗∂/∂u
    + (1/4*u^4 + 1/4*v^4 + 1/2*(u^2 + 1)*v^2 + 1/2*u^2 + 1/4) ∂/∂v⊗∂/∂v

Let us check that `ginv` is indeed the inverse of `g`:

```python
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:**
Manifolds, Release 10.3

```python
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

**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 $\text{Ric}$ 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 “\text{Ric}(g)”, where “g” is the metric’s name

**OUTPUT:**

- the Ricci tensor $\text{Ric}$, as an instance of `TensorField` of tensor type (0,2) and symmetric

**EXAMPLES:**

Ricci tensor of the standard metric on the 2-sphere:

```python
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'\theta:(0,pi):\theta \phi:(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 \ d\theta \otimes d\theta + a^2 \sin(\theta)^2 \ d\phi \otimes d\phi
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(\theta)^2]
```

### ricci_scalar

**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

$$r = g^{ij} Ric_{ij}$$

3.3. Pseudo-Riemannian Metrics and Degenerate Metrics
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 “\text{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:

```python
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'\theta:(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 d\theta \otimes d\theta + a^2 \sin(\theta)^2 d\phi \otimes d\phi
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
(\theta, \phi) ↦ 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

$$R(\omega, u, v, w) = \left(\omega, \nabla_u \nabla_v w - \nabla_v \nabla_u w - \nabla_{[u,v]} w\right)$$

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 “\text{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:

```python
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')
(continues on next page)
```
In dimension 2, the Riemann tensor can be expressed entirely in terms of the Ricci scalar $r$:

\[ R^i_{jkl} = \frac{r}{2} \left( \delta^i_{jk} g_{il} - \delta^i_{jl} g_{ik} \right) \]

This formula can be checked here, with the r.h.s. rewritten as $-rg_{jk} \delta^i_{li}$:

\[
\text{sage: } g.\text{riemann()} == \text{...: } -g.\text{ricci_scalar()} \ast (g*U.\text{tangent_identity_field()}).\text{antisymmetrize}(2,3)
\]

True

Using SymPy as symbolic engine:

\[
\text{sage: } M.\text{set_calculus_method('sympy')}
\]

\[
\text{sage: } g = U.\text{metric('g')}
\]

\[
\text{sage: } g[1,1], g[2,2] = a^2, a^2 \sin(\theta)^2
\]

\[
\text{sage: } g.\text{riemann()[:,!]}
\]

\[
\text{[[[0, 0], [0, 0]], [[0, \sin(\theta)^2], [-\sin(\theta)^2, 0]]],}
\]

\[
\text{[[[0, -1], [1, 0]], [[0, 0], [0, 0]]]]}
\]

**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 “\text{mathrm}(\text{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:
Manifolds, Release 10.3

```
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(h)
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 $\mathbb{R}^3$ with spherical coordinates:

```sage
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(rr:(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
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
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]
```

(continues on next page)
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
```

Using SymPy as symbolic engine:

```
sage: g = M.metric(g)
sage: g[1,1], g[1,2], g[2,2] = 1+x, x*y, 1-y
sage: g.sqrt_abs_det().expr()
sqrt(-x^2*y^2 - x*y + x - y + 1)
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)
```

**volume_form**(contra=0)

Volume form (Levi-Civita tensor) $\epsilon$ associated with the metric.

The volume form $\epsilon$ is an $n$-form ($n$ being the manifold’s dimension) such that for any oriented vector basis $(e_i)$ which is orthonormal with respect to the metric, the condition

$$\epsilon(e_1, \ldots, e_n) = 1$$

holds.

Notice that a volume form requires an orientable manifold with a preferred orientation, see `orientation()` for details.

**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, or of the subclass `MultivectorField` if $k = n$

**EXAMPLES:**

Volume form on $\mathbb{R}^3$ with spherical coordinates, using the standard orientation, which is predefined:

```
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(rr:(0,+oo) th:(0,pi): \theta ph:(0,2*pi):
```

(continues on next page)
\[ \phi \]

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()
\[ g = dr\otimes dr + r^2 dth\otimes dth + r^2*\sin(th)^2 dph\otimes dph \]

sage: eps = g.volume_form() ; eps
3-form \( \epsilon_g \) on the Open subset \( U \) of the 3-dimensional differentiable manifold \( M \)

sage: eps.display()
\[ \epsilon_g = r^2*\sin(th) \, dr\wedge dth\wedge dph \]

sage: eps[1,2,3] == g.sqrt_abs_det()
True

sage: latex(eps)
\[ \varepsilon \]

The tensor field of components \( \varepsilon^i_{jk} \) (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 \( \varepsilon^i_{jk} \) (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 \( \varepsilon^i_{jk} \) (contra=3):

sage: eps3 = g.volume_form(3) ; eps3
3-vector field on the Open subset \( U \) of the 3-dimensional differentiable manifold \( M \)

sage: eps3.tensor_type()
(3, 0)

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]] * g.sqrt_abs_det() == 1
True
If the manifold has no predefined orientation, an orientation must be set before invoking `volume_form()`. For instance let consider the 2-sphere described by the stereographic charts from the North and South pole:

```python
sage: M = Manifold(2, 'M', structure='Riemannian')
sage: U = M.open_subset('U'); V = M.open_subset('V')
sage: M.declare_union(U, V)
sage: c_xy.<x,y> = U.chart()  # stereographic chart from the North pole
sage: c_uv.<u,v> = V.chart()  # stereographic chart 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: eU = c_xy.frame(); eV = c_uv.frame()
sage: g = M.metric()
sage: g[eU,0,0], g[eU, 1,1] = 4/(1+x^2+y^2)^2, 4/(1+x^2+y^2)^2
sage: g.add_comp_by_continuation(eV, U.intersection(V), chart=c_uv)
sage: eps = g.volume_form()
```

```
Traceback (most recent call last):
  ...:
  ...: restrictions2= u^2+v^2!=0)
  ...: intersection_name='W', restrictions1= x^2+y^2!=0,
  ...: restrictions2= u^2+v^2!=0)
  ...: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
  ...: uv_to_xy = xy_to_uv.inverse()
  ...: eU = c_xy.frame(); eV = c_uv.frame()
  ...: g = M.metric()
  ...: g[eU,0,0], g[eU, 1,1] = 4/(1+x^2+y^2)^2, 4/(1+x^2+y^2)^2
  ...: g.add_comp_by_continuation(eV, U.intersection(V), chart=c_uv)
  ...: eps = g.volume_form()
ValueError: 2-dimensional Riemannian manifold M must admit an orientation
```

Let us define the orientation of $M$ such that $eU$ is right-handed; $eV$ is then left-handed and in order to define an orientation on the whole of $M$, we introduce a vector frame on $V$ by swapping $eV$'s vectors:

```python
sage: f = V.vector_frame('f', (eV[1], eV[0]))
sage: M.set_orientation([eU, f])
```

We have then, factorizing the components for a nicer display:

```python
sage: eps = g.volume_form()
sage: eps.apply_map(factor, frame=eU, keep_other_components=True)
sage: eps.display(eU)
eps_g = 4/(x^2 + y^2 + 1)^2 dx\wedge dy
sage: eps.display(eV)
eps_g = -4/(u^2 + v^2 + 1)^2 du\wedge dv
```

Note the minus sign in the above expression, reflecting the fact that $eV$ is left-handed with respect to the chosen orientation.

**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 “\text{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 \( H^3 \):

```python
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):ho th:(0,pi):	heta 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: 

\( g = b^2 \, drh \otimes drh + b^2 \sinh(rh) \, dth \otimes dth \\
+ b^2 \sin(th)^2 \sinh(rh) \, dph \otimes dph \)

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`

Bases: `PseudoRiemannianMetric, 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 \( \mathbb{R} \), via a differentiable mapping \( \Phi : U \to M \). 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 \( \mathbb{R} \)).

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:

\[
g(p) : T_q U \times T_q U \longrightarrow \mathbb{R}
\]

where \( T_q U \) stands for the tangent space to manifold \( M \) at the point \( q = \Phi(p) \), such that \( g(p) \) is symmetric: \( \forall (u,v) \in T_q U \times T_q U, \; g(p)(v,u) = g(p)(u,v) \) 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 \( \mathfrak{X}(U, \Phi) \) 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[:]
\[
\begin{array}{cc}
x + 1 & x*y \\
x*y & -x + 1
\end{array}
\]
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, (∂/∂x,∂/∂y)), Coordinate frame (M, (∂/∂u,∂/∂v))]
sage: g[c_uv.frame(),:] # metric components in frame c_uv.frame() expressed in M
˓
→
sage: g.display(c_uv.frame())
g = (1/2*x*y^2 - 1/8*u^2 + 1/2) du⊗du + 1/4*u du⊗dv + (1/4*u + 1/4*v) dv⊗du
+ (1/4*u + 1/4*v) dv⊗dv

As a shortcut of the above command, one can pass just the chart `c_uv` to `display`, the vector frame being then assumed to be the coordinate frame associated with the chart:

sage: g.display(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 \texttt{inverse()}:

```python
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))]
```

\texttt{inverse}(\texttt{expansion_symbol=None}, \texttt{order=1})

Return the inverse metric.

INPUT:

- \texttt{expansion_symbol} -- (default: \texttt{None}) symbolic variable; if specified, the inverse will be expanded in power series with respect to this variable (around its zero value)
- \texttt{order} -- integer (default: 1); the order of the expansion if \texttt{expansion_symbol} is not \texttt{None}; the \texttt{order} is defined as the degree of the polynomial representing the truncated power series in \texttt{expansion_symbol}; currently only first order inverse is supported

If \texttt{expansion_symbol} is set, then the zeroth order metric must be invertible. Moreover, subsequent calls to this method will return a cached value, even when called with the default value (to enable computation of derived quantities). To reset, use \texttt{_del_derived()}.

OUTPUT:

- instance of \texttt{TensorFieldParal} with \texttt{tensor_type} = (2,0) representing the inverse metric

EXAMPLES:

Inverse metric on a 2-dimensional manifold:

```python
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:

```python
sage: g[1,2] = 0 ; g[:]
[(x + 1) 0]
[ 0 -x + 1]
sage: g.inverse()[:]
[(1/(x + 1)) 0]
[ 0 -1/(x - 1)]
```

Using SymPy as symbolic engine:
Manifolds, Release 10.3

```python
sage: M.set_calculus_method('sympy')
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  1 - x]
sage: g.inverse()[:]
(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)]

Demonstration of the series expansion capabilities:

```python
sage: M = Manifold(4, 'M', structure='Lorentzian')
sage: C.<t,x,y,z> = M.chart()
sage: e = var('e')
sage: g = M.metric()
sage: h = M.tensor_field(0, 2, sym=(0,1))
sage: g[0, 0], g[1, 1], g[2, 2], g[3, 3] = -1, 1, 1, 1
sage: h[0, 1], h[1, 2], h[2, 3] = 1, 1, 1
sage: g.set(g + e*h)
If e is a small parameter, g is a tridiagonal approximation of the Minkowski metric:

```python
sage: g[:]
[-1  e  0  0]
[ e  1  e  0]
[ 0  e  1  e]
[ 0  0  e  1]

The inverse, truncated to first order in e, is:

```python
sage: g.inverse(expansion_symbol=e)[:]
[-1  e  0  0]
[ e  1 -e  0]
[ 0 -e  1 -e]
[ 0  0  e  1]

If inverse() is called subsequently, the result will be the same. This allows for all computations to be made to first order:

```python
sage: g.inverse()[:]
[-1  e  0  0]
[ e  1 -e  0]
[ 0 -e  1 -e]
[ 0  0  e  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 \( \Phi : U \to V \), where \( V \) is a subdomain of self._codomain (type: DiffMap) If None, the restriction of self._vmodule._dest_map to \( U \) is used.

926 Chapter 3. Pseudo-Riemannian Manifolds
OUTPUT:

- instance of `PseudoRiemannianMetricParal` representing the restriction.

EXAMPLES:

Restriction of a Lorentzian metric on $\mathbb{R}^2$ 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 “\text{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'\theta:(0,pi):	heta \phi:(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:

```python
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
```

3.4 Levi-Civita Connections

The class LeviCivitaConnection implements the Levi-Civita connection associated with some pseudo-Riemannian metric on a smooth manifold.

AUTHORS:

• Eric Gourgoulhon, Michal Bejger (2013-2015) : initial version
• Marco Mancini (2015) : parallelization of some computations
• Marius Gerbershagen (2022) : use the first Bianchi identity in the computation of the Riemann tensor

REFERENCES:

• [KN1963]
• [Lee1997]
• [ONe1983]

class sage.manifolds.differentiable.levi_civita_connection.LeviCivitaConnection (metric, name, latex_name=None, init_coef=True)

Bases: AffineConnection

Levi-Civita connection on a pseudo-Riemannian manifold.

Let $M$ be a differentiable manifold of class $C^\infty$ (smooth manifold) over $\mathbb{R}$ endowed with a pseudo-Riemannian metric $g$. Let $C^\infty(M)$ be the algebra of smooth functions $M \to \mathbb{R}$ (cf. DiffScalarFieldAlgebra) and let $\mathfrak{X}(M)$ be the $C^\infty(M)$-module of vector fields on $M$ (cf. VectorFieldModule). The Levi-Civita connection associated with $g$ is the unique operator

$$\nabla : \mathfrak{X}(M) \times \mathfrak{X}(M) \to \mathfrak{X}(M)$$

that

• is $\mathbb{R}$-bilinear, i.e. is bilinear when considering $\mathfrak{X}(M)$ as a vector space over $\mathbb{R}$
• is $C^\infty(M)$-linear w.r.t. the first argument: $\forall f \in C^\infty(M), \nabla_f uv = f\nabla u v$
• obeys Leibniz rule w.r.t. the second argument: $\forall f \in C^\infty(M), \nabla_u(fv) = df(u)v + f\nabla u v$
• is torsion-free
• is compatible with $g$: $\forall (u, v, w) \in \mathfrak{X}(M)^3, u(g(v, w)) = g(\nabla_u v, w) + g(v, \nabla_u w)$

The Levi-Civita connection $\nabla$ gives birth to the covariant derivative operator acting on tensor fields, denoted by the same symbol:

$$\nabla : T^{(k,l)}(M) \rightarrow T^{(k,l+1)}(M)$$

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

$$\forall u \in \mathfrak{X}(M), \nabla_u v = \nabla v(., u)$$

More generally for any tensor field $t \in T^{(k,l)}(M)$, we have

$$\forall u \in \mathfrak{X}(M), \nabla_u t = \nabla t(., u)$$

**Note:** The above convention means that, in terms of index notation, the “derivation index” in $\nabla t$ is the last one:

$$\nabla e^{a_1 \ldots a_k}_{b_1 \ldots b_l} = (\nabla t)^{a_1 \ldots a_k}_{b_1 \ldots b_l c}$$

**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 $\mathbb{R}^3$ expressed in spherical coordinates:

```plaintext
sage: forget() # for doctests only
sage: M = Manifold(3, 'R^3', start_index = 1)
sage: c_spher.<r,th,ph> = M.chart(rr:(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:
and that it is torsionless:

```
sage: nab.torsion() == 0
True
```

As a check, let us enforce the computation of the torsion:

```
  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, (∂/∂r,∂/∂θ,∂/∂φ))
sage: nab.coef()
3-indices components w.r.t. Coordinate frame (R^3, (∂/∂r,∂/∂θ,∂/∂φ)),
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 an array
[[[0, 0, 0], [0, -r, 0], [0, 0, -r*\sin(\theta)^2]],
 [[0, 1/r, 0], [1/r, 0, 0], [0, 0, -\cos(\theta)*\sin(\theta)]],
 [[0, 0, 1/r], [0, 0, \cos(\theta)/\sin(\theta)], [1/r, \cos(\theta)/\sin(\theta), 0]]
sage: nab.display()
# display only the non-vanishing symbols
\Gamma^r_{\theta \theta} = -r
\Gamma^r_{\phi \phi} = -r*\sin(\theta)^2
\Gamma^\theta_{r \theta} = 1/r
\Gamma^\theta_{\theta \theta} = 1/r
\Gamma^\theta_{\phi \phi} = -\cos(\theta)*\sin(\theta)
\Gamma^\phi_{r \phi} = 1/r
\Gamma^\phi_{\phi \theta} = \cos(\theta)/\sin(\theta)
\Gamma^\phi_{\phi \phi} = 1/r
\Gamma^\phi_{\phi \phi} = \cos(\theta)/\sin(\theta)
sage: nab.display(only_nonredundant=True)  # skip redundancy due to symmetry
\Gamma^r_{\theta \theta} = -r
\Gamma^r_{\phi \phi} = -r*\sin(\theta)^2
\Gamma^\theta_{r \theta} = 1/r
\Gamma^\theta_{\phi \phi} = -\cos(\theta)*\sin(\theta)
\Gamma^\phi_{r \phi} = 1/r
\Gamma^\phi_{\phi \theta} = \cos(\theta)/\sin(\theta)
\Gamma^\phi_{\phi \phi} = 1/r
\Gamma^\phi_{\phi \phi} = \cos(\theta)/\sin(\theta)
```

The same display can be obtained via the function `christoffel_symbols_display()` acting on the metric:

```
sage: g.christoffel_symbols_display(chart=c_spher)
\Gamma^r_{\theta \theta} = -r
\Gamma^r_{\phi \phi} = -r*\sin(\theta)^2
```
\[ \Gamma^\theta_{\phi,\theta} = -\cos(\theta)\sin(\theta) \]
\[ \Gamma^\phi_{\theta,\phi} = \cos(\theta)/\sin(\theta) \]

**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 \( \Gamma^k_{ij} \) defined by

\[
\nabla_{e_j} e_i = \Gamma^k_{ij} e_k
\]

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 \( \mathbb{R}^3 \) expressed in spherical coordinates:

```sage
M = Manifold(3, 'R^3', start_index=1)
c_spher.<r,th,ph> = M.chart(rr:(0,+oo) th:(0,pi): \theta ph:(0,2*pi): \phi)
g = M.metric(g)
g[1,1], g[2,2], g[3,3] = 1, r^2 , (r*sin(th))^2
g.display()
g = dr⊗dr + r^2 dth⊗dth + r^2*sin(th)^2 dph⊗dph
nab = g.connection()
gam = nab.coef() ; gam
3-indices components w.r.t. Coordinate frame \((\mathbb{R}^3, (\partial/\partial r,\partial/\partial \theta,\partial/\partial \phi))\), with symmetry on the index positions \((1, 2)\)
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]]]
```

The only non-zero Christoffel symbols:

```sage
gam[1,2,2], gam[1,3,3]
(-r, -r*sin(th)^2)
gam[2,1,2], gam[2,3,3]
(1/r, -cos(th)*sin(th))
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]]
```

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 “\text{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:

```python
sage: M = Manifold(2, 'S^2', start_index=1)
sage: c_spher.<th,ph> = M.chart(r'\theta:(0,\pi):	heta \phi:(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 = d\theta\otimes d\theta + \sin(\theta)^2 d\phi\otimes d\phi
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) = d\theta\otimes d\theta + \sin(\theta)^2 d\phi\otimes d\phi
```

Checking that the Ricci tensor of the Levi-Civita connection associated to Schwarzschild metric is identically zero (as a solution of the Einstein equation):

```python
sage: M = Manifold(4, 'M')
sage: c_BL.<t,r,th,ph> = M.chart(r't:(0,\infty) \theta:(0,\pi):\theta \phi:(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)
```

(continues on next page)
Manifolds, Release 10.3

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 \texttt{sage.manifolds.differentiable.affine_connection.AffineConnection.riemann()} to take into account the symmetry of the Riemann tensor for a Levi-Civita connection.

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:

- \texttt{name} -- (default: None) name given to the Riemann tensor; if none, it is set to “Riem(g)”, where “g” is the metric’s name
- \texttt{latex_name} -- (default: None) \LaTeX{} symbol to denote the Riemann tensor; if none, it is set to “\texttt{mathrm(Riem)}(g)”, where “g” is the metric’s name

OUTPUT:

- the Riemann curvature tensor $R$, as an instance of \texttt{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)), coord_restrictions=lambda x,y: x^2+y^2<1))
....: # Cartesian coord. on the Poincaré disk
sage: g = M.metric('g')
\begin{verbatim}
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
\end{verbatim}

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

sage: riem.display_comp()
\begin{verbatim}
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_xyx = -4/(x^4 + y^4 + 2*(x^2 - 1)*y^2 - 2*x^2 + 1)
\end{verbatim}

The same computation parallelized on 2 cores:

sage: Parallelism().set(nproc=2)
sage: riem_backup = riem
sage: g = M.metric('g')
\begin{verbatim}
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()
\end{verbatim}
sage: riem = nab.riemann(); riem
Tensor field Riem(g) of type (1,3) on the 2-dimensional differentiable manifold M
sage: riem == riem_backup
True
sage: Parallelism().set(nproc=1)  # switch off parallelization

torsion()

Return the connection’s torsion tensor (identically zero for a Levi-Civita connection).

See \texttt{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 \texttt{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

3.5 Pseudo-Riemannian submanifolds

An embedded (resp. immersed) submanifold of a pseudo-Riemannian manifold $(M, g)$ is an embedded (resp. immersed) submanifold $N$ of $M$ as a differentiable manifold (see \texttt{differentiable_submanifold}) such that pull back of the metric tensor $g$ via the embedding (resp. immersion) endows $N$ with the structure of a pseudo-Riemannian manifold.

The following examples show how to compute the various quantities related to the intrinsic and extrinsic geometries of a hyperbolic slicing of the 3-dimensional Minkowski space.

We start by declaring the ambient manifold $M$ and the submanifold $N$:

sage: M = Manifold(3, 'M', structure="Lorentzian")
sage: N = Manifold(2, 'N', ambient=M, structure="Riemannian", start_index=1)

The considered slices being spacelike hypersurfaces, they are Riemannian manifolds.

Let us introduce the Minkowskian coordinates $(w, x, y)$ on $M$ and the polar coordinates $(\rho, \theta)$ on the submanifold $N$:

sage: E.<w,x,y> = M.chart()
sage: C.<rh,th> = N.chart(rrh:(0,+oo):ho th:(0,2*pi):	heta)

Let $b$ be the hyperbola semi-major axis and $t$ the parameter of the foliation:
One can then define the embedding $\phi_t$:

```sage
sage: phi = N.diff_map(M, {(C,E)}: [b*cosh(rh)*t, 
....: b*sinh(rh)*cos(th), 
....: b*sinh(rh)*sin(th)]})
```

as well as its inverse (when considered as a diffeomorphism onto its image):

```sage
sage: phi_inv = M.diff_map(N, {(E,C)}: [log(sqrt((b^2 + x^2 + y^2)/b^2 - 1)), 
....: atan2(y, x)])
```

and the partial inverse expressing the foliation parameter $t$ as a scalar field on $M$:

```sage
sage: phi_inv_t = M.scalar_field({E: w - sqrt(b^2 + x^2 + y^2)})
```

One can check that the inverse is correct with:

```sage
sage: (phi*phi_inv).display() 
M → M 
(w, x, y) ↦ ((b^2 + x^2 + y^2 + sqrt(b^2 + x^2 + y^2)*(t + sqrt(x^2 + y^2)) + sqrt(x^2 + y^2)*t)/(sqrt(b^2 + x^2 + y^2) + sqrt(x^2 + y^2)), x, y)
```

The first item of the 3-uple in the right-hand does not appear as $w$ because $t$ has not been replaced by its value provided by $\phi_\text{inv}_t$. Once this is done, we do get $w$:

```sage
sage: (phi*phi_inv).expr()[0].subs({t: phi_inv_t.expr()}).simplify_full() 
w
```

The embedding can then be declared:

```sage
sage: N.set_embedding(phi, inverse=phi_inv, var=t, 
....: t_inverse = {t: phi_inv_t})
```

This line does not perform any calculation yet. It just check the coherence of the arguments, but not the inverse, the user is trusted on this point.

Finally, we initialize the metric of $M$ to be that of Minkowski space:

```sage
sage: g = M.metric() 
sage: g[0,0], g[1,1], g[2,2] = -1, 1, 1 
sage: g.display() 
g = -dw⊗dw + dx⊗dx + dy⊗dy
```
With this, the declaration the ambient manifold and its foliation parametrized by $t$ is finished, and calculations can be performed.

The first step is always to find a chart adapted to the foliation. This is done by the method “adapted_chart”:

```sage
T = N.adapted_chart(); T
[Chart (M, (rh_M, th_M, t_M))]
```

$T$ contains a new chart defined on $M$. By default, the coordinate names are constructed from the names of the submanifold coordinates and the foliation parameter indexed by the name of the ambient manifold. By this can be customized, see `adapted_chart()`.

One can check that the adapted chart has been added to $M$’s atlas, along with some coordinates changes:

```sage
M.atlas()
[Chart (M, (w, x, y)), Chart (M, (rh_M, th_M, t_M))]
```

```sage
len(M.coord_changes())
2
```

Let us compute the induced metric (or first fundamental form):

```sage
# long time
gamma = N.induced_metric()
gamma.display()
gamma = b^2 drh⊗drh + b^2*sinh(rh)^2 dth⊗dth
gamma[:]
[ b^2 0]
[ 0 b^2*sinh(rh)^2]
gamma[1,1]
b^2
```

the normal vector:

```sage
N.normal().display()  # long time
n = sqrt(b^2 + x^2 + y^2)/b ∂/∂w + x/b ∂/∂x + y/b ∂/∂y
```

Check that the hypersurface is indeed spacelike, i.e. that its normal is timelike:

```sage
N.ambient_metric()(N.normal(), N.normal()).display()  # long time
g(n,n):(w, x, y) ↦ -1
(rh_M, th_M, t_M) ↦ -1
```

The lapse function is:

```sage
N.lapse().display()  # long time
N: M → ℝ
(w, x, y) ↦ sqrt(b^2 + x^2 + y^2)/b
(rh_M, th_M, t_M) ↦ cosh(rh_M)
```

while the shift vector is:

```sage
N.shift().display()  # long time
beta = -(x^2 + y^2)/b^2 ∂/∂w - sqrt(b^2 + x^2 + y^2)*x/b^2 ∂/∂x
- sqrt(b^2 + x^2 + y^2)*y/b^2 ∂/∂y
```

The extrinsic curvature (or second fundamental form) as a tensor field on the ambient manifold:

3.5. Pseudo-Riemannian submanifolds
The extrinsic curvature as a tensor field on the submanifold:

```
sage: N.extrinsic_curvature()[:,:,] # long time
```

```
[ -b 0]
[ 0 -b*sinh(rh)^2]
```

AUTHORS:

- Florentin Jaffredo (2018): initial version
- Matthias Koeppe (2021): open subsets of submanifolds

REFERENCES:

- B. O'Neill: *Semi-Riemannian Geometry* [ONe1983]
Pseudo-Riemannian submanifold.

An embedded (resp. immersed) submanifold of a pseudo-Riemannian manifold \((M, g)\) is an embedded (resp. immersed) submanifold \(N\) of \(M\) as a differentiable manifold such that pull back of the metric tensor \(g\) via the embedding (resp. immersion) endows \(N\) with the structure of a pseudo-Riemannian manifold.

**INPUT:**

- \(n\) – positive integer; dimension of the submanifold
- \(\text{name}\) – string; name (symbol) given to the submanifold
- \(\text{ambient}\) – (default: None) pseudo-Riemannian manifold \(M\) in which the submanifold is embedded (or immersed). If None, it is set to self
- \(\text{metric\_name}\) – (default: None) string; name (symbol) given to the metric; if None, 'gamma' is used
- \(\text{signature}\) – (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 submanifold’s dimension (Riemannian signature)
- \(\text{base\_manifold}\) – (default: None) if not None, must be a differentiable manifold; the created object is then an open subset of base\_manifold
- \(\text{diff\_degree}\) – (default: infinity) degree of differentiability
- \(\text{latex\_name}\) – (default: None) string; LaTeX symbol to denote the submanifold; if none is provided, it is set to name
- \(\text{metric\_latex\_name}\) – (default: None) string; LaTeX symbol to denote the metric; if none is provided, it is set to metric\_name if the latter is not None and to r'\'gamma' otherwise
- \(\text{start\_index}\) – (default: 0) integer; lower value of the range of indices used for “indexed objects” on the submanifold, e.g. coordinates in a chart
- \(\text{category}\) – (default: None) to specify the category; if None, Manifolds(RR).Differentiable() (or Manifolds(RR).Smooth() if diff\_degree = infinity) is assumed (see the category Manifolds)
- \(\text{unique\_tag}\) – (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 DifferentiableManifold and TopologicalManifold, would return the previously constructed object corresponding to these arguments).

**EXAMPLES:**

Let \(N\) be a 2-dimensional submanifold of a 3-dimensional Riemannian manifold \(M\):

```sage
sage: M = Manifold(3, 'M', structure ="Riemannian")
sage: N = Manifold(2, 'N', ambient=M, structure="Riemannian")
sage: N
2-dimensional Riemannian submanifold N immersed in the 3-dimensional Riemannian manifold M
sage: CM.<x,y,z> = M.chart()
sage: CN.<u,v> = N.chart()
```

Let us define a 1-dimension foliation indexed by \(t\). The inverse map is needed in order to compute the adapted chart in the ambient manifold:

```sage
sage: t = var('t')
sage: phi = N.diff_map(M, {(CN,CM):[u, v, t+u^2+v^2]}); phi
```

(continues on next page)
immersed in the 3-dimensional Riemannian manifold $M$ to the 3-dimensional Riemannian manifold $M$:

$sage$: \( \phi_{\text{inv}} = M.\text{diff}_\text{map}(N, ((\text{CM, CN}): [x, y])) \)

$sage$: \( \phi_{\text{inv\_t}} = M.\text{scalar}_\text{field}((\text{CM, CN}): [z-x^2-y^2]) \)

\( \phi \) can then be declared as an embedding $N \to M$:

$sage$: \( N.\text{set}_\text{embedding}(\phi, \text{inverse}=\phi_{\text{inv}}, \text{var}=t, \ldots: \quad t_{\text{inverse}}=[t: \phi_{\text{inv\_t}}]) \)

The foliation can also be used to find new charts on the ambient manifold that are adapted to the foliation, ie in which the expression of the immersion is trivial. At the same time, the appropriate coordinate changes are computed:

$sage$: \( N.\text{adapted}_\text{chart}() \)

[Chart (M, (u_M, v_M, t_M))]

$sage$: \( \text{len}(M.\text{coord}_\text{changes}()) \)

2

See also:

\textit{manifold} and \textit{differentiable_submanifold}

\textbf{ambient_extrinsic_curvature}()

Return the second fundamental form of the submanifold as a tensor field on the ambient manifold. The result is cached, so calling this method multiple times always returns the same result at no additional cost.

**OUTPUT:**

- (0,2) tensor field on the ambient manifold equal to the second fundamental form once orthogonally projected onto the submanifold

**EXAMPLES:**

A unit circle embedded in the Euclidean plane:

$sage$: \( M.<X, Y> = \text{EuclideanSpace}() \)

$sage$: \( N = \text{Manifold}(1, 'N', \text{ambient}=M, \text{structure}="\text{Riemannian}\") \)

$sage$: \( U = N.\text{open}_\text{subset}('U') \)

$sage$: \( V = N.\text{open}_\text{subset}('V') \)

$sage$: \( N.\text{declare}_\text{union}(U, V) \)

$sage$: \( \text{stereo}_N.<x> = U.\text{chart}() \)

$sage$: \( \text{stereo}_S.<y> = V.\text{chart}() \)

$sage$: \( \text{stereo}_N.\text{to}_S = \text{stereo}_N.\text{transition}_\text{map}(\text{stereo}_S, (4/x), \ldots: \quad \text{intersection}_\text{name}='W', \ldots: \quad \text{restrictions}1=x!=0, \text{restrictions}2=y!=0) \)

$sage$: \( \text{stereo}_S.\text{to}_N = \text{stereo}_N.\text{to}_S.\text{inverse}() \)

$sage$: \( E = M.\text{cartesian}_\text{coordinates}() \)

$sage$: \( \phi = N.\text{diff}_\text{map}(M, \ldots: \quad ((\text{stereo}_N, E): [1/\sqrt{1+x^2/4}, x/2/\sqrt{1+x^2/4}]), \ldots: \quad ((\text{stereo}_S, E): [1/\sqrt{1+4/y^2}, 2/y/\sqrt{1+4/y^2}])) \)

$sage$: \( N.\text{set}_\text{embedding}(\phi) \)

$sage$: \( N.\text{ambient}_\text{second}_\text{fundamental}_\text{form}() \quad \# \text{long time} \)

Field of symmetric bilinear forms $K$ along the 1-dimensional Riemannian submanifold $N$ embedded in the Euclidean plane $E^2$ with values on the Euclidean plane $E^2$:

$sage$: \( N.\text{ambient}_\text{second}_\text{fundamental}_\text{form()}[:] \quad \# \text{long time} \)

\([-x^2/([x^2 + 4]) 2*x/([x^2 + 4])]

\[2*x/([x^2 + 4]) -4/([x^2 + 4])\]
An alias is `ambient_extrinsic_curvature`:

```sage
N.ambient_extrinsic_curvature()[:]
```

# long time

```
[-x^2/(x^2 + 4)  2*x/(x^2 + 4)]
[ 2*x/(x^2 + 4)  -4/(x^2 + 4)]
```

### ambient_first_fundamental_form()

Return the first fundamental form of the submanifold as a tensor of the ambient manifold.

The result is cached, so calling this method multiple times always returns the same result at no additional cost.

**OUTPUT:**

- (0,2) tensor field on the ambient manifold describing the induced metric before projection on the submanifold

**EXAMPLES:**

A unit circle embedded in the Euclidean plane:

```sage
M.<X,Y> = EuclideanSpace()
N = Manifold(1, 'N', ambient=M, structure="Riemannian")
U = N.open_subset('U')
V = N.open_subset('V')
N.declare_union(U,V)
U.stereoN.<x> = U.chart()
V.stereoS.<y> = V.chart()
N.stereoN_to_S = stereoN.transition_map(stereoS, (4/x),
....: intersection_name='W',
....: restrictions1=x!=0, restrictions2=y!=0)
N.stereoS_to_N = stereoN_to_S.inverse()
E = M.cartesian_coordinates()
phi = N.diff_map(M,
....: {(stereoN, E): [1/sqrt(1+x^2/4), x/2/sqrt(1+x^2/4)],
....: (stereoS, E): [1/sqrt(1+4/y^2), 2/y/sqrt(1+4/y^2)])
N.set_embedding(phi)
N.ambient_first_fundamental_form()
```

Tensor field gamma of type (0,2) along the 1-dimensional Riemannian submanifold N embedded in the Euclidean plane E^2 with values on the Euclidean plane E^2

```sage
N.ambient_first_fundamental_form()[:]
```

```
[ x^2/(x^2 + 4) -2*x/(x^2 + 4)]
[ -2*x/(x^2 + 4)  4/(x^2 + 4)]
```

An alias is `ambient_induced_metric`:

```sage
N.ambient_induced_metric()[:]
```

```
[ x^2/(x^2 + 4) -2*x/(x^2 + 4)]
[ -2*x/(x^2 + 4)  4/(x^2 + 4)]
```

### ambient_induced_metric()

Return the first fundamental form of the submanifold as a tensor of the ambient manifold.

The result is cached, so calling this method multiple times always returns the same result at no additional cost.

**OUTPUT:**

- (0,2) tensor field on the ambient manifold describing the induced metric before projection on the submanifold

**EXAMPLES:**
A unit circle embedded in the Euclidean plane:

```python
sage: M.<X,Y> = EuclideanSpace()
sage: N = Manifold(1, 'N', ambient=M, structure="Riemannian")
sage: U = N.open_subset('U')
sage: V = N.open_subset('V')
sage: N.declare_union(U,V)
sage: stereoN.<x> = U.chart()
sage: stereoS.<y> = V.chart()
sage: stereoN_to_S = stereoN.transition_map(stereoS, (4/x),
       intersection_name='W',
       restrictions1=x!=0, restrictions2=y!=0)
sage: stereoS_to_N = stereoN_to_S.inverse()
sage: E = M.cartesian_coordinates()
sage: phi = N.diff_map(M,
       {(stereoN, E): [1/sqrt(1+x^2/4), x/2/sqrt(1+x^2/4)],
        (stereoS, E): [1/sqrt(1+4/y^2), 2/y/sqrt(1+4/y^2)])}
sage: N.set_embedding(phi)
sage: N.ambient_first_fundamental_form()
Tensor field gamma of type (0,2) along the 1-dimensional Riemannian submanifold N embedded in the Euclidean plane E^2 with values on the Euclidean plane E^2

[[ x^2/(x^2 + 4) -2*x/(x^2 + 4)]
[-2*x/(x^2 + 4) 4/(x^2 + 4)]
```

An alias is `ambient_induced_metric`:

```python
sage: N.ambient_induced_metric()[:]
[[ x^2/(x^2 + 4) -2*x/(x^2 + 4)]
[-2*x/(x^2 + 4) 4/(x^2 + 4)]
```

**ambient_metric()**

Return the metric of the ambient manifold.

**OUTPUT:**

- the metric of the ambient manifold

**EXAMPLES:**

```python
sage: M.<x,y,z> = EuclideanSpace()
sage: N = Manifold(2, 'N', ambient=M, structure="Riemannian")
sage: N.ambient_metric()
Riemannian metric g on the Euclidean space E^3
sage: N.ambient_metric().display()
g = dx⊗dx + dy⊗dy + dz⊗dz
sage: N.ambient_metric() is M.metric()
True
```

**ambient_second_fundamental_form()**

Return the second fundamental form of the submanifold as a tensor field on the ambient manifold.

The result is cached, so calling this method multiple times always returns the same result at no additional cost.

**OUTPUT:**

- (0,2) tensor field on the ambient manifold equal to the second fundamental form once orthogonally projected onto the submanifold
EXAMPLES:

A unit circle embedded in the Euclidean plane:

```python
sage: M.<X,Y> = EuclideanSpace()
sage: N = Manifold(1, 'N', ambient=M, structure="Riemannian")
sage: U = N.open_subset('U')
sage: V = N.open_subset('V')
sage: N.declare_union(U,V)
sage: stereoN.<x> = U.chart()
sage: stereoS.<y> = V.chart()
sage: stereoN_to_S = stereoN.transition_map(stereoS, (4/x),
            intersection_name='W',
            restrictions1=x!=0, restrictions2=y!=0)
sage: stereoS_to_N = stereoN_to_S.inverse()
sage: E = M.cartesian_coordinates()
sage: phi = N.diff_map(M,
            {(stereoN, E): [1/sqrt(1+x^2/4), x/2/sqrt(1+x^2/4)],
            (stereoS, E): [1/sqrt(1+4/y^2), 2/y/sqrt(1+4/y^2)]})
sage: N.set_embedding(phi)
sage: N.ambient_second_fundamental_form() # long time
Field of symmetric bilinear forms K along the 1-dimensional
Riemannian submanifold N embedded in the Euclidean plane E^2 with
values on the Euclidean plane E^2
sage: N.ambient_second_fundamental_form()[:]
[-x^2/(x^2 + 4) 2*x/(x^2 + 4)]
[ 2*x/(x^2 + 4) -4/(x^2 + 4)]
```

An alias is `ambient_extrinsic_curvature`:

```python
sage: N.ambient_extrinsic_curvature()[:]
[-x^2/(x^2 + 4) 2*x/(x^2 + 4)]
[ 2*x/(x^2 + 4) -4/(x^2 + 4)]
```

clear_cache()

Reset all the cached functions and the derived quantities.

Use this function if you modified the immersion (or embedding) of the submanifold. Note that when calling a calculus function after clearing, new Python objects will be created.

EXAMPLES:

```python
sage: M.<x,y,z> = EuclideanSpace()
sage: N = Manifold(2, 'N', ambient=M, structure="Riemannian")
sage: C.<th,ph> = N.chart(r'\theta:(0,\pi):\phi:(-\pi,\pi):')
sage: r = var('r', domain='real')
# foliation parameter
sage: assume(r>0)
sage: E = M.cartesian_coordinates()
sage: phi = N.diff_map(M, {(C,E): [r*sin(th)*cos(ph),
            r*sin(th)*sin(ph),
            r*cos(th)]})
sage: phi_inv = M.diff_map(N, {(E,C): [arccos(z/r), atan2(y,x)]})
sage: phi_inv_r = M.scalar_field({E: sqrt(x^2+y^2+z^2)})
sage: N.set_embedding(phi, inverse=phi_inv, var=r,
            t_inverse={r: phi_inv_r})
sage: T = N.adapted_chart()
sage: n = N.normal()
sage: n is N.normal()
True
```

(continues on next page)
\texttt{diff()} 

Return the differential of the scalar field on the ambient manifold representing the first parameter of the foliation associated to \texttt{self}.

The result is cached, so calling this method multiple times always returns the same result at no additional cost.

\textbf{OUTPUT:}

- 1-form field on the ambient manifold

\textbf{EXAMPLES:}

Foliation of the Euclidean 3-space by 2-spheres parametrized by their radii:

\begin{verbatim}
sage: M.<x,y,z> = EuclideanSpace()
sage: N = Manifold(2, 'N', ambient=M, structure="Riemannian")
sage: C.<th,ph> = N.chart(r'\theta:(0,pi):\phi:(-pi,pi):')(continues on next page)
\end{verbatim}
sage: stereoN_to_S = stereoN.transition_map(stereoS, (4/x),
.....:  intersection_name='W',
.....:  restrictions1=x!=0, restrictions2=y!=0)
sage: stereoS_to_N = stereoN_to_S.inverse()
sage: E = M.cartesian_coordinates()
sage: phi = N.diff_map(M,
.....:  {(stereoN, E): [1/sqrt(1+x^2/4), x/2/sqrt(1+x^2/4)],
sage: N.set_embedding(phi)
sage: N.second_fundamental_form().display()  # long time
Field of symmetric bilinear forms K on the 1-dimensional Riemannian
submanifold N embedded in the Euclidean plane E^2
sage: N.second_fundamental_form().display()  # long time
K = -4/(x^4 + 8*x^2 + 16) dx⊗dx

An alias is extrinsic_curvature:

sage: N.extrinsic_curvature().display()  # long time
K = -4/(x^4 + 8*x^2 + 16) dx⊗dx

An example with a non-Euclidean ambient metric:

sage: M = Manifold(2, 'M', structure='Riemannian')
sage: N = Manifold(1, 'N', ambient=M, structure='Riemannian',
.....:  start_index=1)
sage: CM.<x,y> = M.chart()
sage: CN.<u> = N.chart()
sage: g = M.metric()
sage: g[0, 0], g[1, 1] = 1, 1/(1 + y^2)^2
sage: phi = N.diff_map(M, (u, u))
sage: N.set_embedding(phi)
sage: N.second_fundamental_form().display()  # long time
K = 2*sqrt(u^4 + 2*u^2 + 2)*u/(u^6 + 3*u^4 + 4*u^2 + 2) du⊗du

first_fundamental_form()

Return the first fundamental form of the submanifold.

The result is cached, so calling this method multiple times always returns the same result at no additional cost.

OUTPUT:

• the first fundamental form, as an instance of PseudoRiemannianMetric

EXAMPLES:

A sphere embedded in Euclidean space:

sage: M.<x,y,z> = EuclideanSpace()
sage: N = Manifold(2, 'N', ambient=M, structure='Riemannian')
sage: C.<th,ph> = N.chart(r'th:(0,pi):	heta ph:(-pi,pi):\phi')
sage: r = var('r', domain='real')
sage: assume(r>0)
sage: E = M.cartesian_coordinates()
sage: phi = N.diff_map(M, (C,E): [r*sin(th)*cos(ph),
.....:  r*sin(th)*sin(ph),
....: N.set_embedding(phi)
sage: N.first_fundamental_form() # long time
Riemannian metric gamma on the 2-dimensional Riemannian
submanifold N embedded in the Euclidean space E^3
sage: N.first_fundamental_form()[:]
# long time
\[
\begin{bmatrix}
  r^2 & 0 \\
  0 & r^2 \sin(\theta)^2
\end{bmatrix}
\]

An alias is induced_metric:

sage: N.induced_metric()[:]
# long time
\[
\begin{bmatrix}
  r^2 & 0 \\
  0 & r^2 \sin(\theta)^2
\end{bmatrix}
\]

By default, the first fundamental form is named gamma, but this can be customized by means of the argument
metric_name when declaring the submanifold:

sage: P = Manifold(1, 'P', ambient=M, structure='Riemannian',
               metric_name='g')
sage: CP.<t> = P.chart()
sage: F = P.diff_map(M, [t, 2*t, 3*t])
sage: P.set_embedding(F)
sage: P.induced_metric()
Riemannian metric g on the 1-dimensional Riemannian submanifold P
embedded in the Euclidean space E^3
sage: P.induced_metric().display()
g = 14 dt\otimes dt

\textbf{gauss_curvature()}

Return the Gauss curvature of the submanifold.

The \textit{Gauss curvature} is the product of the principal curvatures, or equivalently the determinant of the projection
operator.

The result is cached, so calling this method multiple times always returns the same result at no additional cost.

\textbf{OUTPUT:}

\begin{itemize}
  \item the Gauss curvature as a scalar field on the submanifold
\end{itemize}

\textbf{EXAMPLES:}

A unit circle embedded in the Euclidean plane:

sage: M.<X,Y> = EuclideanSpace()
sage: N = Manifold(1, 'N', ambient=M, structure="Riemannian")
sage: U = N.open_subset('U')
sage: V = N.open_subset('V')
sage: N.declare_union(U,V)
sage: stereoN.<x> = U.chart()
\begin{small}
\begin{verbatim}
  sage: stereoS.<y> = V.chart(
  sage: stereoN.transition_map(stereoS, (4/x),
  ....: intersection_name='W',
  ....: restrictions1=x!=0, restrictions2=y!=0)
  sage: stereoS_to_N = stereoN.transition_map(stereoS, (4/x),
  sage: phi = N.diff_map(M,
\end{verbatim}
\end{small}
\[
\{(\text{stereoN}, E): \left[ \frac{1}{\sqrt{1+x^2/4}}, \frac{x}{2\sqrt{1+x^2/4}} \right], \\
\{(\text{stereoS}, E): \left[ \frac{1}{\sqrt{1+4/y^2}}, \frac{2/y}{\sqrt{1+4/y^2}} \right]\}
\]

sage: \text{N.set\_embedding(\phi)}

sage: \text{N.gauss\_curvature()}  \quad \# \text{long time}
Scalar field on the 1-dimensional Riemannian submanifold N embedded in the Euclidean plane \(E^2\)

sage: \text{N.gauss\_curvature().display()}  \quad \# \text{long time}

\[\text{N} \rightarrow \mathbb{R} \text{ on } U: x \mapsto -1\]
on \(V: y \mapsto -1\)

**\text{gradt}()**

Return the gradient of the scalar field on the ambient manifold representing the first parameter of the foliation associated to \text{self}.

The result is cached, so calling this method multiple times always returns the same result at no additional cost.

**OUTPUT:**

- vector field on the ambient manifold

**EXAMPLES:**

Foliation of the Euclidean 3-space by 2-spheres parametrized by their radii:

sage: \text{M.<x,y,z> = \text{EuclideanSpace}()} 

sage: \text{N = \text{Manifold}(2, \text{\textquoteleft N\textquoteleft}, ambient=M, structure=\textquoteleft Riemannian\textquoteleft)}

sage: \text{C.<th,ph> = N.chart(r\textbf{th}:(0,pi):\theta \textbf{ph}:(-pi,pi): \phi)}

sage: r = \text{var(\textbf{r}, domain=\text{real})}

sage: \text{assume(r>0)}

sage: \text{E = M.cartesian\_coordinates()}

sage: \text{phi = N.diff\_map(M, \{(C,E): [r*sin(th)*cos(ph), \}}

\[
\frac{r*sin(th)*sin(ph), \ldots;}{r^2*\cos(th)}]}

sage: \text{phi\_inv = M.diff\_map(N, \{(E,C): [arccos(z/r), atan2(y,x)]})}

sage: \text{phi\_inv\_r = M.scalar\_field({E: sqrt(x^2+y^2+z^2)})}

sage: \text{N.set\_embedding(phi, inverse=phi\_inv, var=r, \ldots; t\_inverse={r: phi\_inv\_r})}

sage: \text{N.gradt()}

Vector field \text{grad(r)} on the Euclidean space \(E^3\)

sage: \text{N.gradt().display()}

\[
\text{grad(r)} = \frac{x}{\sqrt{x^2+y^2+z^2}} \, e_x + \frac{y}{\sqrt{x^2+y^2+z^2}} \, e_y + \frac{z}{\sqrt{x^2+y^2+z^2}} \, e_z
\]

**\text{induced\_metric}()**

Return the first fundamental form of the submanifold.

The result is cached, so calling this method multiple times always returns the same result at no additional cost.

**OUTPUT:**

- the first fundamental form, as an instance of \text{PseudoRiemannianMetric}

**EXAMPLES:**

A sphere embedded in Euclidean space:

\[
\text{sage: M.<x,y,z> = \text{EuclideanSpace}()} \\
\text{sage: N = \text{Manifold}(2, \text{\textquoteleft N\textquoteleft}, ambient=M, structure=\textquoteleft Riemannian\textquoteleft)}
\]

(continues on next page)
```python
sage: C.<th,ph> = N.chart(r'\theta:(0,pi):	heta ph:(-pi,pi):\phi')
sage: r = var('r', domain='real')
sage: assume(r>0)
sage: E = M.cartesian_coordinates()
sage: phi = N.diff_map(M, {(C,E): [r*sin(th)*cos(ph), r*sin(th)*sin(ph), r*cos(th)]})
sage: N.set_embedding(phi)
sage: N.first_fundamental_form()  # long time
Riemannian metric gamma on the 2-dimensional Riemannian submanifold N embedded in the Euclidean space E^3
sage: N.first_fundamental_form()[:]
# long time

[[r^2 0]
 [0 r^2*sin(th)^2]
]

An alias is `induced_metric`:

```python
sage: N.induced_metric()[:]
# long time

[[r^2 0]
 [0 r^2*sin(th)^2]]
```

By default, the first fundamental form is named `gamma`, but this can be customized by means of the argument `metric_name` when declaring the submanifold:

```python
sage: P = Manifold(1, 'P', ambient=M, structure='Riemannian',
.....: metric_name='g')
sage: CP.<t> = P.chart()
sage: F = P.diff_map(M, [t, 2*t, 3*t])
sage: P.set_embedding(F)
sage: P.induced_metric()
Riemannian metric g on the 1-dimensional Riemannian submanifold P embedded in the Euclidean space E^3
sage: P.induced_metric().display()
g = 14 dt⊗dt
```

**lapse()**

Return the lapse function of the foliation.

The result is cached, so calling this method multiple times always returns the same result at no additional cost.

**OUTPUT:**

- the lapse function, as a scalar field on the ambient manifold

**EXAMPLES:**

Foliation of the Euclidean 3-space by 2-spheres parametrized by their radii:

```python
sage: M.<x,y,z> = EuclideanSpace()
sage: N = Manifold(2, 'N', ambient=M, structure="Riemannian")
sage: C.<th,ph> = N.chart(r'\theta:(0,pi):	heta ph:(-pi,pi):\phi')
sage: r = var('r', domain='real')  # foliation parameter
sage: assume(r>0)
sage: E = M.cartesian_coordinates()
sage: phi = N.diff_map(M, {(C,E): [r*sin(th)*cos(ph),
.....: r*sin(th)*sin(ph),
.....: r*cos(th)]})
sage: phi_inv = M.diff_map(N, {(E,C): [arccos(z/r), atan2(y,x)\n\n[\n]]})
(continues on next page)```
mean_curvature()  
Return the mean curvature of the submanifold.

The mean curvature is the arithmetic mean of the principal curvatures, or equivalently the trace of the projection operator.

The result is cached, so calling this method multiple times always returns the same result at no additional cost.

OUTPUT:
• the mean curvature, as a scalar field on the submanifold

EXAMPLES:
A unit circle embedded in the Euclidean plane:

metric(name=None, signature=None, latex_name=None, dest_map=None)  
Return the induced metric (first fundamental form) or define a new metric tensor on the submanifold.

A new (uninitialized) metric is returned only if the argument name is provided and differs from the metric name declared at the construction of the submanifold; otherwise, the first fundamental form is returned.

INPUT:
• name – (default: None) name given to the metric; if name is None or equals the metric name declared when constructing the submanifold, the first fundamental form is returned (see \texttt{first_fundamental_form()})

• signature – (default: None; ignored if name is 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 submanifold’s dimension (Riemannian signature)

• latex_name – (default: None; ignored if name is None) LaTeX symbol to denote the metric; if None, it is formed from name

• dest_map – (default: None; ignored if name is None) instance of class \texttt{DiffMap} representing the destination map $\Phi : U \to M$, where $U$ is the current submanifold; if None, the identity map is assumed (case of a metric tensor field on $U$)

OUTPUT:

• instance of \texttt{PseudoRiemannianMetric}

EXAMPLES:

Induced metric on a straight line of the Euclidean plane:

```python
sage: M.<x,y> = EuclideanSpace()
sage: N = Manifold(1, 'N', ambient=M, structure='Riemannian')
sage: CN.<t> = N.chart()
sage: F = N.diff_map(M, [t, 2*t])
sage: N.set_embedding(F)
sage: N.metric()
Riemannian metric gamma on the 1-dimensional Riemannian submanifold N embedded in the Euclidean plane E^2
sage: N.metric().display()
gamma = 5 \, dt \otimes dt
```

Setting the argument name to that declared while constructing the submanifold (default = 'gamma') yields the same result:

```python
sage: N.metric(name='gamma') is N.metric()
True
```

while using a different name allows one to define a new metric on the submanifold:

```python
sage: h = N.metric(name='h'); h
Riemannian metric h on the 1-dimensional Riemannian submanifold N embedded in the Euclidean plane E^2
sage: h[0, 0] = 1  # initialization
sage: h.display()
h = dt@dt
```

\texttt{mixed_projection}(tensor, indices=0)

Return de $n+1$ decomposition of a tensor on the submanifold and the normal vector.

The $n+1$ decomposition of a tensor of rank $k$ can be obtained by contracting each index either with the normal vector or the projection operator of the submanifold (see \texttt{projector()}).

INPUT:

• tensor – any tensor field, eventually along the submanifold if no foliation is provided.
• indices – (default: 0) list of integers containing the indices on which the projection is made on the normal vector. By default, all projections are made on the submanifold. If an integer $n$ is provided, the $n$ first contractions are made with the normal vector, all the other ones with the orthogonal projection operator.

OUTPUT:

• tensor field of rank $k - \text{len(indices)}$

EXAMPLES:

Foliation of the Euclidean 3-space by 2-spheres parametrized by their radii:

```python
sage: M.<x,y,z> = EuclideanSpace()
```

```python
sage: N = Manifold(2, 'N', ambient=M, structure="Riemannian")
```

```python
sage: C.<th,ph> = N.chart(r'\theta:(0,pi):\theta ph:(-pi,pi): ph')
```

```python
sage: r = var('r', domain='real')  # foliation parameter
```

```python
sage: assume(r>0)
```

```python
sage: E = M.cartesian_coordinates(); E
```

```python
sage: phi = N.diff_map(M, {(C,E): [r *sin(th)*cos(ph),
....: r*sin(th)*sin(ph),
....: r*cos(th)]})
```

```python
sage: phi_inv = M.diff_map(N, {(E,C): [arccos(z /r), atan2(y,x)]})
```

```python
sage: phi_inv_r = M.scalar_field({E: sqrt(x ^2+y^2+z^2)})
```

```python
sage: N.set_embedding(phi, inverse=phi_inv, var=r,
....: t_inverse={r: phi_inv_r})
```

```python
sage: T = N.adapted_chart()
```

If `indices` is not specified, the mixed projection of the ambient metric coincides with the first fundamental form:

```python
sage: g = M.metric()
```

```python
sage: gpp = N.mixed_projection(g); gpp  # long time
Tensor field of type (0,2) on the Euclidean space $E^3$
```

```python
sage: gpp == N.ambient_first_fundamental_form()  # long time
True
```

The other non-redundant projections are:

```python
sage: gnp = N.mixed_projection(g, [0]); gnp  # long time
1-form on the Euclidean space $E^3$
```

and:

```python
sage: gnn = N.mixed_projection(g, [0,1]); gnn
Scalar field on the Euclidean space $E^3$
```

which is constant and equal to 1 (the norm of the unit normal vector):

```python
sage: gnn.display()
E^3 → ℝ
(x, y, z) ↦ 1
(th_E3, ph_E3, r_E3) ↦ 1
```

`normal()`

Return a normal unit vector to the submanifold.

If a foliation is defined, it is used to compute the gradient of the foliation parameter and then the normal vector:

```python
sage: normal()
Return a normal unit vector to the submanifold.
```

3.5. Pseudo-Riemannian submanifolds
vector. If not, the normal vector is computed using the following formula:
\[
\mathbf{n} = \mathbf{\tilde{v}}(dx_0 \wedge dx_1 \wedge \cdots \wedge dx_{n-1})
\]

where the star stands for the Hodge dual operator and the wedge for the exterior product.

This formula does not always define a proper vector field when multiple charts overlap, because of the arbitrariness of the direction of the normal vector. To avoid this problem, the method `normal()` considers the graph defined by the atlas of the submanifold and the changes of coordinates, and only calculate the normal vector once by connected component. The expression is then propagate by restriction, continuation, or change of coordinates using a breadth-first exploration of the graph.

The result is cached, so calling this method multiple times always returns the same result at no additional cost.

**OUTPUT:**
- vector field on the ambient manifold (case of a foliation) or along the submanifold with values in the ambient manifold (case of a single submanifold)

**EXAMPLES:**

**Foliation of the Euclidean 3-space by 2-spheres parametrized by their radii:**

```sage
M.<x,y,z> = EuclideanSpace()
N = Manifold(2, 'N', ambient=M, structure="Riemannian")
C.<th,ph> = N.chart((r'th:(0,pi):\theta ph:(-pi,pi):\phi')
r = var('r', domain='real') # foliation parameter
assume(r>0)
E = M.cartesian_coordinates()
phi = N.diff_map(M, {(C,E): [r*sin(th)*cos(ph),
.....: r*sin(th)*sin(ph),
.....: r*cos(th)]})
phi_inv = M.diff_map(N, {(E,C): [arccos(z /r), atan2(y,x)]})
N.set_embedding(phi, inverse=phi_inv, var=r,
....: t_inverse={r: phi_inv_r})
T = N.adapted_chart()
N.normal() # long time
Vector field n on the Euclidean space E^3
N.normal().display() # long time
n = x/sqrt(x^2 + y^2 + z^2) e_x + y/sqrt(x^2 + y^2 + z^2) e_y
+ z/sqrt(x^2 + y^2 + z^2) e_z
```

Or in spherical coordinates:

```sage
N.normal().display(T[0].frame(),T[0]) # long time
n = \partial/\partial r_E3
```

Let us now consider a sphere of constant radius, i.e. not assumed to be part of a foliation, in stereographic coordinates:

```sage
M.<X,Y,Z> = EuclideanSpace()
N = Manifold(2, 'N', ambient=M, structure="Riemannian")
U = N.open_subset('U')
V = N.open_subset('V')
N.declare_union(U, V)
stereoN.<x,y> = U.chart()
stereoS.<xp,yp> = V.chart("xp:x yp:y")
stereoN_to_S = stereoN.transition_map(stereoS,
```
(continued from previous page)

\[
\begin{align*}
(x/(x^2+y^2), y/(x^2+y^2)),
\end{align*}
\]

\[
\begin{align*}
intersection\_name='W',
\end{align*}
\]

\[
\begin{align*}
restrictions1= x^2+y^2!=0,
\end{align*}
\]

\[
\begin{align*}
restrictions2= xp^2+yp^2!=0)
\end{align*}
\]

\[
\begin{align*}
sage: \text{stereoS}\_to\_N = \text{stereoN}\_to\_S\_inverse()
\end{align*}
\]

\[
\begin{align*}
sage: W = U.intersection(V)
\end{align*}
\]

\[
\begin{align*}
sage: \text{stereoN}\_W = \text{stereoN}\_restrict(W)
\end{align*}
\]

\[
\begin{align*}
sage: \text{stereoS}\_W = \text{stereoS}\_restrict(W)
\end{align*}
\]

\[
\begin{align*}
sage: A = W.open\_subset('A', \text{coord\_def}={\text{stereoN}\_W: (y!=0, x<0),}
\end{align*}
\]

\[
\begin{align*}
\text{stereoS}\_W: (yp!=0, xp<0})
\end{align*}
\]

\[
\begin{align*}
sage: \text{stereoN}\_W = \text{stereoN}\_restrict(W)
\end{align*}
\]

\[
\begin{align*}
sage: \text{stereoS}\_W = \text{stereoS}\_restrict(W)
\end{align*}
\]

\[
\begin{align*}
sage: \text{stereoN}\_A = \text{stereoN}\_restrict(A)
\end{align*}
\]

\[
\begin{align*}
sage: \text{stereoS}\_A = \text{stereoS}\_restrict(A)
\end{align*}
\]

\[
\begin{align*}
sage: \text{stereoN}\_A = \text{stereoN}\_restrict(A)
\end{align*}
\]

\[
\begin{align*}
sage: \text{stereoS}\_A = \text{stereoS}\_restrict(A)
\end{align*}
\]

\[
\begin{align*}
sage: E = M.cartesian\_coordinates()
\end{align*}
\]

\[
\begin{align*}
sage: \text{phi} = N.diff\_map(M, \{(\text{stereoN}, E): [2*x/(1+x^2+y^2),
\end{align*}
\]

\[
\begin{align*}
2*y/(1+x^2+y^2),
\end{align*}
\]

\[
\begin{align*}
\text{stereoS}, E): [2*xp/(1+xp^2+yp^2),
\end{align*}
\]

\[
\begin{align*}
2*yp/(1+xp^2+yp^2),
\end{align*}
\]

\[
\begin{align*}
\text{stereoS}, E): (1-xp^2-yp^2)/(1+xp^2+yp^2))},
\end{align*}
\]

\[
\begin{align*}
\text{name}='\Phi', \text{latex\_name}=r'\Phi')
\end{align*}
\]

\[
\begin{align*}
sage: N.set\_embedding(\text{phi})
\end{align*}
\]

The method \texttt{normal()} now returns a tensor field along \texttt{N}:

\[
\begin{align*}
sage: n = N.normal() \quad \# \text{long time}
sage: n \quad \# \text{long time}
\end{align*}
\]

Vector field \texttt{n} along the 2-dimensional Riemannian submanifold \texttt{N} embedded in the Euclidean space E^3 with values on the Euclidean space E^3

Let us check that the choice of orientation is coherent on the two top frames:

\[
\begin{align*}
sage: n.restrict(V).\text{display}(\text{format\_spec=spher}) \quad \# \text{long time}
n = -\text{cos}(\text{phi})*\text{sin}(\text{the}) \text{ e}_X - \text{sin}(\text{phi})*\text{sin}(\text{the}) \text{ e}_Y - \text{cos}(\text{the}) \text{ e}_Z
\end{align*}
\]

\[
\begin{align*}
sage: n.restrict(U).\text{display}(\text{format\_spec=spher}) \quad \# \text{long time}
n = -\text{cos}(\text{phi})*\text{sin}(\text{the}) \text{ e}_X - \text{sin}(\text{phi})*\text{sin}(\text{the}) \text{ e}_Y - \text{cos}(\text{the}) \text{ e}_Z
\end{align*}
\]

\[
\begin{align*}
\text{open\_subset} \quad \text{(name}, \text{latex\_name}=\text{None}, \text{coord\_def}={}}, \text{supersets}=\text{None})
\end{align*}
\]

Create an open subset of \texttt{self}.

An open subset is a set that is (i) included in the manifold and (ii) open with respect to the manifold's topology.

3.5. Pseudo-Riemannian submanifolds
It is a differentiable manifold by itself. Moreover, equipped with the restriction of the manifold metric to itself, it is a pseudo-Riemannian manifold.

As self is a submanifold of its ambient manifold, the new open subset is also considered a submanifold of that. Hence the returned object is an instance of \texttt{PseudoRiemannianSubmanifold}.

**INPUT:**

- \texttt{name} – name given to the open subset
- \texttt{latex\_name} – (default: None) LaTeX symbol to denote the subset; if none is provided, it is set to \texttt{name}
- \texttt{coord\_def} – (default: \{\}) definition of the subset in terms of coordinates; \texttt{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.
- \texttt{supersets} – (default: only \texttt{self}) list of sets that the new open subset is a subset of

**OUTPUT:**

- instance of \texttt{PseudoRiemannianSubmanifold} representing the created open subset

**EXAMPLES:**

```plaintext
sage: M = Manifold(3, 'M', structure='Riemannian')
sage: N = Manifold(2, 'N', ambient=M, structure='Riemannian'); N
2-dimensional Riemannian submanifold N immersed in the
3-dimensional Riemannian manifold M
sage: S = N.subset('S'); S
Subset S of the
2-dimensional Riemannian submanifold N immersed in the
3-dimensional Riemannian manifold M
sage: O = N.subset('O', is_open=True); O  # indirect doctest
Open subset O of the
2-dimensional Riemannian submanifold N immersed in the
3-dimensional Riemannian manifold M
sage: phi = N.diff_map(M)
sage: N.set_embedding(phi)
sage: N
2-dimensional Riemannian submanifold N embedded in the
3-dimensional Riemannian manifold M
sage: S = N.subset('S'); S
Subset S of the
2-dimensional Riemannian submanifold N embedded in the
3-dimensional Riemannian manifold M
sage: O = N.subset('O', is_open=True); O  # indirect doctest
Open subset O of the
2-dimensional Riemannian submanifold N embedded in the
3-dimensional Riemannian manifold M
```

\texttt{principal\_curvatures} \texttt{(chart)}

Return the principal curvatures of the submanifold.

The \texttt{principal curvatures} are the eigenvalues of the projection operator. The resulting scalar fields are named \texttt{k\_i} with the index \texttt{i} ranging from 0 to the submanifold dimension minus one.

The result is cached, so calling this method multiple times always returns the same result at no additional cost.

**INPUT:**

- \texttt{chart} – chart in which the principal curvatures are to be computed
OUTPUT:

- the principal curvatures, as a list of scalar fields on the submanifold

EXAMPLES:

A unit circle embedded in the Euclidean plane:

```
sage: M.<X,Y> = EuclideanSpace()
sage: N = Manifold(1, 'N', ambient=M, structure="Riemannian")
sage: U = N.open_subset('U')
sage: V = N.open_subset('V')
sage: N.declare_union(U,V)
sage: stereoN.<x> = U.chart()
sage: stereoS.<y> = V.chart()
sage: stereoN_to_S = stereoN.transition_map(stereoS, 4/x,
.....:     intersection_name='W',
.....:     restrictions1=x!=0, restrictions2=y!=0)
sage: stereoS_to_N = stereoN_to_S.inverse()
sage: E = M.cartesian_coordinates()
sage: phi = N.diff_map(M,
.....:     {(stereoN, E): [1/sqrt(1+x^2/4), x/2/sqrt(1+x^2/4)],
sage: N.set_embedding(phi)
sage: N.principal_curvatures(stereoN)
# long time
[Scalar field k_0 on the 1-dimensional Riemannian submanifold N
embedded in the Euclidean plane E^2]
sage: N.principal_curvatures(stereoN)[0].display()  # long time
k_0: N → ℝ
on U: x ↦ -1
on W: y ↦ -1
```

**principal_directions**(chart)

Return the principal directions of the submanifold.

The *principal directions* are the eigenvectors of the projection operator. The result is formatted as a list of pairs (eigenvector, eigenvalue).

The result is cached, so calling this method multiple times always returns the same result at no additional cost.

INPUT:

- chart – chart in which the principal directions are to be computed

OUTPUT:

- list of pairs (vector field, scalar field) representing the principal directions and the associated principal curvatures

EXAMPLES:

A unit circle embedded in the Euclidean plane:

```
sage: M.<X,Y> = EuclideanSpace()
sage: N = Manifold(1, 'N', ambient=M, structure="Riemannian")
sage: U = N.open_subset('U')
sage: V = N.open_subset('V')
sage: N.declare_union(U,V)
sage: stereoN.<x> = U.chart()
sage: stereoS.<y> = V.chart()
sage: stereoN_to_S = stereoN.transition_map(stereoS, 4/x),
(continues on next page)```
intersection_name='W',
restrictions1=x!=0, restrictions2=y!=0)
sage: stereoS_to_N = stereoN_to_S.inverse()
sage: E = M.cartesian_coordinates()
sage: phi = N.diff_map(M,
{(stereoN, E): [1/sqrt(1+x^2/4), x/2/sqrt(1+x^2/4)],
(stereoS, E): [1/sqrt(1+4/y^2), 2/y/sqrt(1+4/y^2)]})
sage: N.set_embedding(phi)

# long time
[(Vector field e_0 on the 1-dimensional Riemannian submanifold N
embedded in the Euclidean plane E^2, -1)]

sage: N.principal_directions(stereoN)[0][0].display()  # long time
e_0 = ∂/∂x

project (tensor)
Return the orthogonal projection of a tensor field onto the submanifold.

INPUT:

• tensor – any tensor field to be projected onto the submanifold. If no foliation is provided, must be a
tensor field along the submanifold.

OUTPUT:

• orthogonal projection of tensor onto the submanifold, as a tensor field of the ambient manifold

EXAMPLES:
Foliation of the Euclidean 3-space by 2-spheres parametrized by their radii:

Let us perform the projection of the ambient metric and check that it is equal to the first fundamental form:

Note that the output of project() is not cached.

projector ()
Return the orthogonal projector onto the submanifold.

The result is cached, so calling this method multiple times always returns the same result at no additional cost.

OUTPUT:
• the orthogonal projector onto the submanifold, as tensor field of type (1,1) on the ambient manifold

EXAMPLES:
Foliation of the Euclidean 3-space by 2-spheres parametrized by their radii:

```
sage: M.<x,y,z> = EuclideanSpace()
sage: N = Manifold(2, 'N', ambient=M, structure="Riemannian")
sage: C.<th,ph> = N.chart(r'\theta:(0,pi):\theta ph:(-pi,pi): \phi')
sage: r = var('r', domain='real')  # foliation parameter
sage: assume(r>0)
sage: E = M.cartesian_coordinates()
sage: phi = N.diff_map(M, {C,E}: [r*\sin(th)*\cos(ph),
.....: r*\sin(th)*\sin(ph),
.....: r*\cos(th)])
sage: phi_inv = M.diff_map(N, {E,C}: [arccos(z/r), atan2(y,x)])
sage: phi_inv_r = M.scalar_field({E: sqrt(x^2+y^2+z^2)})
sage: N.set_embedding(phi, inverse=phi_inv, var=r,
.....: t_inverse={r: phi_inv_r})
sage: T = N.adapted_chart()
```

The orthogonal projector onto $N$ is a type-(1,1) tensor field on $M$:

```
sage: N.projector()  # long time
Tensor field gamma of type (1,1) on the Euclidean space $E^3$
```

Check that the orthogonal projector applied to the normal vector is zero:

```
sage: N.projector().contract(N.normal()).display()  # long time
0
```

`second_fundamental_form()`

Return the second fundamental form of the submanifold.

The result is cached, so calling this method multiple times always returns the same result at no additional cost.

OUTPUT:

• the second fundamental form, as a symmetric tensor field of type (0,2) on the submanifold

EXAMPLES:
A unit circle embedded in the Euclidean plane:

```
sage: M.<X,Y> = EuclideanSpace()
sage: N = Manifold(1, 'N', ambient=M, structure="Riemannian")
sage: U = N.open_subset('U')
sage: V = N.open_subset('V')
sage: N.declare_union(U,V)
sage: stereoN.<x> = U.chart()
sage: stereoS.<y> = V.chart()
sage: stereoN_to_S = stereoN.transition_map(stereoS, (4/x),
.....: intersection_name='W',
.....: restrictions1=x!=0, restrictions2=y!=0)
sage: stereoS_to_N = stereoN_to_S.inverse()
sage: E = M.cartesian_coordinates()
sage: phi = N.diff_map(M,
.....: {(stereoN, E): [1/sqrt(1+x^2/4), x/2/sqrt(1+x^2/4)],
.....: (stereoS, E): [1/sqrt(1+y^2), 2/y/sqrt(1+y^2)]})
sage: N.set_embedding(phi)
```

(continues on next page)
An alias is `extrinsic_curvature`:

```sage
N.extrinsic_curvature().display()  # long time
K = -4/(x^4 + 8*x^2 + 16) dx⊗dx
```

An example with a non-Euclidean ambient metric:

```sage
M = Manifold(2, 'M', structure='Riemannian')
N = Manifold(1, 'N', ambient=M, structure='Riemannian',
....:     start_index=1)
CM.<x,y> = M.chart()
CN.<u> = N.chart()
g = M.metric()
g[0, 0], g[1, 1] = 1, 1/(1 + y^2)^2
phi = N.diff_map(M, (u, u))
N.set_embedding(phi)
N.second_fundamental_form()
Field of symmetric bilinear forms K on the 1-dimensional Riemannian
submanifold N embedded in the 2-dimensional Riemannian manifold M
N.second_fundamental_form().display()
K = 2*sqrt(u^4 + 2*u^2 + 2)*u/(u^6 + 3*u^4 + 4*u^2 + 2) du⊗du
```

**shape_operator()**

Return the shape operator of the submanifold.

The shape operator is equal to the second fundamental form with one of the indices upped.

The result is cached, so calling this method multiple times always returns the same result at no additional cost.

**OUTPUT:**

- the shape operator, as a tensor field of type (1,1) on the submanifold

**EXAMPLES:**

A unit circle embedded in the Euclidean plane:

```sage
M.<X,Y> = EuclideanSpace()
N = Manifold(1, 'N', structure='Riemannian')
U = N.open_subset('U')
V = N.open_subset('V')
N.declare_union(U,V)
stereoN.<x> = U.chart()
stereoS.<y> = V.chart()
stereoN_to_S = stereoN.transition_map(stereoS, (4/x),
.....:     intersection_name='W',
.....:     restrictions1=x!=0, restrictions2=y!=0)
stereoS_to_N = stereoN_to_S.inverse()
E = M.cartesian_coordinates()
phi = N.diff_map(M,
.....:     {(stereoN, E): [1/sqrt(1+x^2/4), x/2/sqrt(1+x^2/4)],
.....:      (stereoS, E): [1/sqrt(1+y^2), 2/y/sqrt(1+y^2)]})
```
sage: N.set_embedding(phi)
sage: N.shape_operator()  # long time
Tensor field of type (1,1) on the 1-dimensional Riemannian
submanifold N embedded in the Euclidean plane E^2
sage: N.shape_operator().display()  # long time
\frac{\partial}{\partial x} \otimes dx

shift()

Return the shift vector associated with the first adapted chart of the foliation.

The result is cached, so calling this method multiple times always returns the same result at no additional cost.

OUTPUT:

- shift vector field on the ambient manifold

EXAMPLES:

Foliation of the Euclidean 3-space by 2-spheres parametrized by their radii:

sage: M.<x,y,z> = EuclideanSpace()
sage: N = Manifold(2, 'N', ambient=M, structure="Riemannian")
sage: C.<th,ph> = N.chart(r'th:(0,pi):\theta ph:(-pi,pi): \phi')
sage: r = var('r', domain='real')  # foliation parameter
sage: assume(r>0)
sage: E = M.cartesian_coordinates()
sage: phi = N.diff_map(M, {(C,E):
    [r*sin(th)*cos(ph),
     r*sin(th)*sin(ph),
     r*cos(th)]})
sage: phi_inv = M.diff_map(N, {(E,C):
    [arccos(z /r), atan2(y,x)]})
sage: phi_inv_r = M.scalar_field({E: sqrt(x^2+y^2+z^2)})
sage: N.set_embedding(phi, inverse=phi_inv, var=r,
    t_inverse={r: phi_inv_r})
sage: T = N.adapted_chart()
sage: N.shift()  # long time
Vector field beta on the Euclidean space E^3
sage: N.shift().display()  # long time
beta = 0

3.6 Degenerate Metric Manifolds

3.6.1 Degenerate manifolds
class sage.manifolds.differentiable.degenerate.DegenerateManifold(n, name, metric_name=None, signature=None, base_manifold=None, diff_degree=+Infinity, latex_name=None, metric_latex_name=None, start_index=0, category=None, unique_tag=None)

Bases: DifferentiableManifold

Degenerate Manifolds

A degenerate manifold (or a null manifold) is a pair \((M, g)\) where \(M\) is a real differentiable manifold (see DifferentiableManifold) and \(g\) is a field of degenerate symmetric bilinear forms on \(M\) (see DegenerateMetric).

INPUT:

- \(n\) – positive integer; dimension of the manifold
- \(name\) – string; name (symbol) given to the manifold
- \(metric\_name\) – (default: None) string; name (symbol) given to the metric; if None, \('g'\) is used
- \(signature\) – (default: None) signature \(S\) of the metric as a tuple: \(S = (n_+, n_-, n_0)\), where \(n_+\) (resp. \(n_-\), resp. \(n_0\)) is the number of positive terms (resp. negative terms, resp. zero terms) in any diagonal writing of the metric components; if \(signature\) is not provided, \(S\) is set to \((ndim - 1, 0, 1)\), being \(ndim\) the manifold's dimension
- \(ambient\) – (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
- \(metric\_latex\_name\) – (default: None) string; LaTeX symbol to denote the metric; if none is provided, it is set to metric\_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(RR).Differentiable() (or Manifolds(RR).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 DifferentiableManifold and TopologicalManifold, would return the previously constructed object corresponding to these arguments).

EXAMPLES:

A degenerate manifold is constructed via the generic function Manifold(), using the keyword structure:
The metric associated with \( M \) is:

```python
sage: g = M.metric()
sage: g
degenerate metric g on the 3-dimensional degenerate_metric manifold M
sage: g.signature()
(0, 2, 1)
```

Its value has to be initialized either by setting its components in various vector frames (see the above examples regarding the 2-sphere and Minkowski spacetime) or by making it equal to a given field of symmetric bilinear forms (see the method `set()` of the metric class). Both methods are also covered in the documentation of method `metric()` below.

REFERENCES:

• [DB1996]
• [DS2010]

**metric** *(name=None, signature=None, latex_name=None, dest_map=None)*  
Return the metric giving the null manifold structure to the manifold, or define a new metric tensor on the manifold.

**INPUT:**

- `name` – (default: `None`) name given to the metric; if `name` is `None` or matches the name of the metric defining the null manifold structure of `self`, the latter metric is returned
- `signature` – (default: `None`; ignored if `name` is `None`) signature \( S \) of the metric as a tuple: \( S = (n_+, n_-, n_0) \), where \( n_+ \) (resp. \( n_- \), resp. \( n_0 \)) is the number of positive terms (resp. negative terms, resp. zero terms) in any diagonal writing of the metric components; if `signature` is not provided, \( S \) is set to \( (\text{ndim} - 1, 0, 1) \), being `ndim` the manifold's dimension
- `latex_name` – (default: `None`; ignored if `name` is `None`) LaTeX symbol to denote the metric; if `None`, it is formed from `name`
- `dest_map` – (default: `None`; ignored if `name` is `None`) instance of class `DiffMap` representing the destination map \( \Phi : U \to M \), where \( U \) is the current manifold; if `None`, the identity map is assumed (case of a metric tensor field on \( U \))

**OUTPUT:**

- instance of `DegenerateMetric`

**EXAMPLES:**

Metric of a 3-dimensional degenerate manifold:

```python
sage: M = Manifold(3, 'M', structure='degenerate_metric', start_index=1)
sage: X.<x,y,z> = M.chart()
sage: g = M.metric(); g
degenerate metric g on the 3-dimensional degenerate_metric manifold M
```
The metric remains to be initialized, for instance by setting its components in the coordinate frame associated to the chart $X$:

```
sage: g[1,1], g[2,2] = -1, 1
sage: g.display()
g = -dx⊗dx + dy⊗dy
sage: g[:]
[-1  0  0]
[ 0  1  0]
[ 0  0  0]
```

Alternatively, the metric can be initialized from a given field of degenerate symmetric bilinear forms; we may create the former object by:

```
sage: X.coframe()
Coordinate coframe (M, (dx,dy,dz))
sage: dx, dy = X.coframe()[1], X.coframe()[2]
sage: b = dx*dx + dy*dy
sage: b
Field of symmetric bilinear forms dx⊗dx+dy⊗dy on the 3-dimensional degenerate_metric manifold M
```

We then use the metric method `set()` to make $g$ being equal to $b$ as a symmetric tensor field of type $(0,2)$:

```
sage: g.set(b)
sage: g.display()
g = dx⊗dx + dy⊗dy
```

Another metric can be defined on $M$ by specifying a metric name distinct from that chosen at the creation of the manifold (which is $g$ by default, but can be changed thanks to the keyword `metric_name` in `Manifold()`):

```
sage: h = M.metric('h'); h
degenerate metric h on the 3-dimensional degenerate_metric manifold M
sage: h[1,1], h[2,2], h[3,3] = 1+y^2, 1+z^2, 1+x^2
sage: h.display()
h = (y^2 + 1) dx⊗dx + (z^2 + 1) dy⊗dy + (x^2 + 1) dz⊗dz
```

The metric tensor $h$ is distinct from the metric entering in the definition of the degenerate manifold $M$:

```
sage: h is M.metric()
False
```

while we have of course:

```
sage: g is M.metric()
True
```

Providing the same name as the manifold’s default metric returns the latter:

```
sage: M.metric('g') is M.metric()
True
```

`open_subset (name, latex_name=None, coord_def={})`

Create an open subset of `self`.

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. Moreover, equipped with the restriction of the manifold metric to itself,
it is a null manifold. Hence the returned object is an instance of *DegenerateManifold*.

**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:**

- instance of *DegenerateManifold* representing the created open subset

**EXAMPLES:**

Open subset of a 3-dimensional degenerate manifold:

```python
sage: M = Manifold(3, 'M', structure='degenerate_metric', start_index=1)
sage: X.<x,y,z> = M.chart()
sage: U = M.open_subset('U', coord_def={X: [x > 0, y > 0]}); U
Open subset U of the 3-dimensional degenerate_metric manifold M
sage: type(U)
<class 'sage.manifolds.differentiable.degenerate.DegenerateManifold_with_category'>
```

We initialize the metric of $M$:

```python
sage: g = M.metric()
sage: g[1,1], g[2,2] = -1, 1
```

Then the metric on $U$ is determined as the restriction of $g$ to $U$:

```python
sage: gU = U.metric(); gU
degenerate metric g on the Open subset U of the 3-dimensional degenerate_metric manifold M
sage: gU.display()
g = -dx*dx + dy*dy
sage: gU is g.restrict(U)
True
```

### 3.6. Degenerate Metric Manifolds

class *sage.manifolds.differentiable.degenerate.TangentTensor*(tensor, embedding, screen=None)

**Bases:** *TensorFieldParal*

Let $S$ be a lightlike submanifold embedded in a pseudo-Riemannian manifold $(M, g)$ with $\Phi$ the embedding map. Let $T_1$ be a tensor on $M$ along $S$ or not. $\text{TangentTensor}(T_1, \Phi)$ returns the restriction $T_2$ of $T_1$ along $S$ that in addition can be applied only on vector fields tangent to $S$, when $T_1$ has a covariant part.

**INPUT:**

- tensor – a tensor field on the ambient manifold
- embedding – the embedding map $\Phi$

**EXAMPLES:**

Section of the lightcone of the Minkowski space with a hyperplane passing through the origin:
sage: M = Manifold(4, 'M', structure="Lorentzian")
sage: X.<t,x,y,z> = M.chart()
sage: S = Manifold(2, 'S', ambient=M, structure='degenerate_metric')
sage: X_S.<u,v> = S.chart()
sage: Phi = S.diff_map(M, {(X_S, X): [sqrt(u^2+v^2), u, v, 0]},
                      name='Phi', latex_name=r'$\Phi'$)
sage: Phi_inv = M.diff_map(S, {(X, X_S): [x, y]}, name='Phi_inv',
                      latex_name=r'$\Phi^{-1}$')
sage: S.set_immersion(Phi, inverse=Phi_inv); S.declare_embedding()
sage: g = M.metric()
sage: g[0,0], g[1,1], g[2,2], g[3,3] = -1, 1, 1, 1
sage: V = M.vector_field(0,0,0,1)
sage: S.set_transverse(rigging=t, normal=V)
sage: xi = M.vector_field(sqrt(x^2+y^2+z^2), x, y, 0)
sage: U = M.vector_field(0, -y, x, 0)
sage: Sc = S.screen(xi, U, Sc);

sage: T1 = M.tensor_field(1,1).along(Phi); T1[0,0] = 1
sage: V1 = M.vector_field().along(Phi); V1[0] = 1; V1[1]=1
sage: T1(V1).display()
∂/∂t

sage: from sage.manifolds.differentiable.degenerate_submanifold import
    TangentTensor

sage: T2 = TangentTensor(T1, Phi)
sage: T2
Tensor field of type (1,1) along the 2-dimensional degenerate
submanifold S embedded in 4-dimensional differentiable manifold M
with values on the 4-dimensional Lorentzian manifold M

Of course $T_1$ and $T_2$ give the same output on vector fields tangent to $S$:

sage: T1(xi.along(Phi)).display()
sqrt(u^2 + v^2) ∂/∂t
sage: T2(xi.along(Phi)).display()
sqrt(u^2 + v^2) ∂/∂t

extension()

Return initial tensor

EXAMPLES:

Section of the lightcone of the Minkowski space with a hyperplane passing through the origin:

sage: M = Manifold(4, 'M', structure="Lorentzian")
sage: X.<t,x,y,z> = M.chart()
sage: S = Manifold(2, 'S', ambient=M, structure='degenerate_metric')
sage: X_S.<u,v> = S.chart()
sage: Phi = S.diff_map(M, {(X_S, X): [sqrt(u^2+v^2), u, v, 0]},
                      name='Phi', latex_name=r'$\Phi'$)
sage: Phi_inv = M.diff_map(S, {(X, X_S): [x, y]}, name='Phi_inv',
                      latex_name=r'$\Phi^{-1}$')
sage: S.set_immersion(Phi, inverse=Phi_inv); S.declare_embedding()
sage: g = M.metric()
sage: g[0,0], g[1,1], g[2,2], g[3,3] = -1, 1, 1, 1
sage: V = M.vector_field(); V[3] = 1
sage: S.set_transverse(rigging=t, normal=V)
3.6.2 Degenerate submanifolds

An embedded (resp. immersed) degenerate submanifold of a proper pseudo-Riemannian manifold \((M, g)\) is an embedded (resp. immersed) submanifold \(H\) of \(M\) as a differentiable manifold such that pull back of the metric tensor \(g\) via the embedding (resp. immersion) endows \(H\) with the structure of a degenerate manifold.

Degenerate submanifolds are study in many fields of mathematics and physics, for instance in Differential Geometry (especially in geometry of lightlike submanifold) and in General Relativity. In geometry of lightlike submanifolds, according to the dimension \(r\) of the radical distribution (see below for definition of radical distribution), degenerate submanifolds have been classified into 4 subgroups: \(r\)-lightlike submanifolds, Coisotropic submanifolds, Isotropic submanifolds and Totally lightlike submanifolds. (See the book of Krishan L. Duggal and Aurel Bejancu [DS2010].)

In the present module, you can define any of the 4 types but most of the methods are implemented only for degenerate hypersurfaces who belong to \(r\)-lightlike submanifolds. However, they might be generalized to \(1\)-lightlike submanifolds.

In the literature there is a new approach (the rigging technique) for studying \(1\)-lightlike submanifolds but here we use the method of Krishan L. Duggal and Aurel Bejancu based on the screen distribution.

Let \(H\) be a lightlike hypersurface of a pseudo-Riemannian manifold \((M, g)\). Then the normal bundle \(TH^\perp\) intersect the tangent bundle \(TH\). The radical distribution is defined as \(\text{Rad}(TH) = TH \cap TH^\perp\). In case of hypersurfaces, and more generally \(1\)-lightlike submanifolds, this is a rank 1 distribution. A screen distribution \(S(TH)\) is a complementary of \(\text{Rad}(TH)\) in \(TH\).

Giving a screen distribution \(S(TH)\) and a null vector field \(\xi\) locally defined and spanning \(\text{Rad}(TH)\), there exists a unique transversal null vector field ‘\(N\)’ locally defined and such that \(g(N, \xi) = 1\). From a transverse vector ‘\(v\)’, the null rigging ‘\(N\)’ is giving by the formula

\[
N = \frac{1}{g(\xi, v)} \left( v - \frac{g(v, v)}{2g(\xi, v)} \xi \right)
\]

Tensors on the ambient manifold \(M\) are projected on \(H\) along \(N\) to obtain induced objects. For instance, induced connection is the linear connection defined on \(H\) through the Levi-Civitta connection of \(g\) along \(N\). To work on a degenerate submanifold, after defining \(H\) as an instance of \(\text{DifferentiableManifold}\), with the keyword \text{structure='degenerate_metric'}, you have to set a transverse vector \(v\) and screen distribution together with the radical distribution.

An example of degenerate submanifold from General Relativity is the horizon of the Schwarzschild black hole. Allow us to recall that Schwarzschild black hole is the first non-trivial solution of Einstein’s equations. It describes the metric inside a star of radius \(R = 2m\), being \(m\) the inertial mass of the star. It can be seen as an open ball in a Lorentzian manifold structure on \(\mathbb{R}^4\):

\[
sage: M = Manifold(4, 'M', structure="Lorentzian")
sage: X_M.<t, r, th, ph> = \\
\]
....: M.chart(r"t r:(0,oo) th:(0,pi): \theta ph:(0,2*pi): \phi")

sage: var('m'); assume(m>0)

m

sage: g = M.metric()

sage: g[0,0], g[0,1], g[1,1], g[2,2], g[3,3] = 

-1+2*m/r, 2*m/r, 1+2*m/r, r^2, r^2*sin(th)^2

Let us define the horizon as a degenerate hypersurface:

sage: H = Manifold(3, 'H', ambient=M, structure='degenerate_metric')

sage: H
degenerate hypersurface H embedded in 4-dimensional differentiable manifold M

A 2-dimensional degenerate submanifold of a Lorentzian manifold:

sage: M = Manifold(4, 'M', structure="Lorentzian")

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

sage: S = Manifold(2, 'S', ambient=M, structure='degenerate_metric')

sage: S
2-dimensional degenerate submanifold S embedded in 4-dimensional differentiable manifold M

sage: X_S.<u,v> = S.chart()

sage: Phi = S.diff_map(M, {(X_S, X): [u, u, u, v]},

name='Phi', latex_name=r'\Phi');

sage: Phi_inv = M.diff_map(S, {(X, X_S): [x,y]}, name='Phi_inv',

latex_name=r'\Phi^{-1}');

sage: S.set_immersion(Phi, inverse=Phi_inv); S.declare_embedding()

sage: g = M.metric()

sage: g[0,0], g[1,1], g[2,2], g[3,3] = -1,1,1,1

sage: S.set_transverse(rigging=[x,y])

sage: xi = M.vector_field(); xi[0] = 1; xi[1] = 1

sage: V = M.vector_field(); V[3] = 1

sage: Sc = S.screen(Sc, V, xi)

sage: S.default_screen()
screen distribution Sc along the 2-dimensional degenerate submanifold S embedded in 4-dimensional differentiable manifold M mapped into the 4-dimensional Lorentzian manifold M

sage: S.ambient_metric()
Lorentzian metric g on the 4-dimensional Lorentzian manifold M

sage: S.induced_metric()
degenerate metric gamma on the 2-dimensional degenerate submanifold S embedded in 4-dimensional differentiable manifold M

sage: S.first_fundamental_form()
Field of symmetric bilinear forms g|S along the 2-dimensional degenerate submanifold S embedded in 4-dimensional differentiable manifold M with values on the 4-dimensional Lorentzian manifold M

sage: S.adapted_frame()
Vector frame (S, (vv_0,vv_1,vv_2,vv_3)) with values on the 4-dimensional Lorentzian manifold M

sage: S.projection(V)
Tensor field of type (1,0) along the 2-dimensional degenerate submanifold $S$ embedded in 4-dimensional differentiable manifold $M$ with values on the 4-dimensional Lorentzian manifold $M$

```python
sage: S.weingarten_map()  # long time
Tensor field nabla_g(xi)|X(S) of type (1,1) along the 2-dimensional degenerate submanifold $S$ embedded in 4-dimensional differentiable manifold $M$ with values on the 4-dimensional Lorentzian manifold $M$

sage: SO = S.shape_operator()  # long time
sage: SO.display()  # long time
A^* = 0

sage: S.is_tangent(xi.along(Phi))
True
sage: v = M.vector_field(); v[1] = 1
sage: S.is_tangent(v.along(Phi))
False
```

AUTHORS:
- Hans Fotsing Tetsing (2019) : initial version

REFERENCES:
- [DB1996]
- [DS2010]
- [FNO2019]
class sage.manifolds.differentiable.degenerate_submanifold.DegenerateSubmanifold(n, name, ambient=None, metric_name=None, signature=None, base_manifold=None, diff_degree=+Infinity, latex_name=None, metric_latex_name=None, start_index=0, category=None, unique_tag=None)

Bases: DegenerateManifold, DifferentiableSubmanifold

Degenerate submanifolds

An embedded (resp. immersed) degenerate submanifold of a proper pseudo-Riemannian manifold \((M, g)\) is an embedded (resp. immersed) submanifold \(H\) of \(M\) as a differentiable manifold such that pull back of the metric tensor \(g\) via the embedding (resp. immersion) endows \(H\) with the structure of a degenerate manifold.

INPUT:

- \(n\) – positive integer; dimension of the manifold
- \(name\) – string; name (symbol) given to the manifold
- \(ambient\) – (default: None) pseudo-Riemannian manifold \(M\) in which the submanifold is embedded (or immersed). If None, it is set to self
- \(metric_name\) – (default: None) string; name (symbol) given to the metric; if None, \('g'\) is used
- \(signature\) – (default: None) signature \(S\) of the metric as a tuple: \(S = (n_+, n_-, n_0)\), where \(n_+\) (resp. \(n_-\), resp. \(n_0\)) is the number of positive terms (resp. negative terms, resp. zero terms) in any diagonal writing of the metric components; if \(signature\) is not provided, \(S\) is set to \((ndim - 1, 0, 1)\), being \(ndim\) the manifold’s dimension
- \(base_manifold\) – (default: None) if not None, must be a topological manifold; the created object is then an open subset of \(base_manifold\)
- \(diff_degree\) – (default: infinity) degree of differentiability
- \(latex_name\) – (default: None) string; LaTeX symbol to denote the manifold; if none are provided, it is set to name
- `metric_latex_name` – (default: `None`) string; LaTeX symbol to denote the metric; if none is provided, it is set to `metric_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)

**See also:**

`manifold` and `differentiable_submanifold`

**adapted_frame**(`screen=None`)

Return a frame \((e_1, \ldots, e_p, \xi_1, \ldots, \xi_r, v_1, \ldots, v_q, N_1, \ldots, N_n)\) of the ambient manifold along the submanifold, being \(e_i\) vector fields spanning the giving screen distribution, \(\xi_i\) vector fields spanning radical distribution, \(v_i\) normal transversal vector fields, \(N_i\) rigging vector fields corresponding to the giving screen.

**INPUT:**

- `screen` – (default: `None`) an instance of `Screen`. If `None` default screen is used.

**OUTPUT:**

- a frame on the ambient manifold along the submanifold

**EXAMPLES:**

A degenerate hyperplane the 4-dimensional Minkowski space \(\mathbb{R}^4_1\):

```
sage: M = Manifold(4, 'M', structure='Lorentzian')
sage: X.<t,x,y,z> = M.chart()
sage: S = Manifold(3, 'S', ambient=M, structure='degenerate_metric')
sage: X_S.<u,v,w> = S.chart()
sage: Phi = S.diff_map(M, {(X_S, X): [u, u, v, w]},
      name=Phi, latex_name=r'\Phi');
sage: Phi_inv = M.diff_map(S, {(X, X_S): [x,y, z]}, name=Phi_inv,
      latex_name=r'\Phi^{-1}');
sage: S.set_immersion(Phi, inverse=Phi_inv); S.declare_embedding()
sage: g = M.metric()
sage: g[0,0], g[1,1], g[2,2], g[3,3] = -1,1,1,1
sage: S.set_transverse(rigging=x)
sage: xi = M.vector_field(); xi[0] = 1; xi[1] = 1
sage: Sc = S.screen('Sc', (U,V), xi); # long time
sage: T = S.adapted_frame(); T # long time
Vector frame (S, (vv_0,vv_1,vv_2,vv_3)) with values on the 4-dimensional Lorentzian manifold M
```

**ambient_metric()**

Return the metric of the ambient manifold. The submanifold has to be embedded

**OUTPUT:**

- the metric of the ambient manifold

**EXAMPLES:**

The lightcone of the 3D Minkowski space:
default_screen()

Return the default screen distribution

OUTPUT:

• an instance of Screen

EXAMPLES:

A degenerate hyperplane the 4-dimensional Minkowski space $\mathbb{R}^4_1$:

```python
sage: M = Manifold(4, 'M', structure="Lorentzian")
sage: X.<t,x,y,z> = M.chart()
sage: S = Manifold(3, 'S', ambient=M, structure='degenerate_metric')
sage: X_S.<u,v,w> = S.chart()
sage: Phi = S.diff_map(M, {(X_S, X): [u, u, v, w]},
.....: name='Phi', latex_name=r'\Phi');
sage: Phi_inv = M.diff_map(S, {(X, X_S): [x, y, z]}, name='Phi_inv',
.....: latex_name=r'\Phi^{-1}');
sage: S.set_immersion(Phi, inverse=Phi_inv); S.declare_embedding()
sage: g = M.metric()
sage: g[0,0], g[1,1], g[2,2], g[3,3] = -1,1,1,1
sage: S.set_transverse(rigging=x)
sage: xi = M.vector_field(); xi[0] = 1; xi[1] = 1
sage: Sc = S.screen('Sc', (U,V), xi) # long time
sage: S.default_screen() # long time
```

screen distribution $\mathcal{S}_c$ along the degenerate hypersurface $S$ embedded in 4-dimensional differentiable manifold $M$ mapped into the 4-dimensional Lorentzian manifold $M$

extrinsic_curvature(screen=None)

This method is implemented only for null hypersurfaces. The method returns a tensor $B$ of type $(0,2)$ instance of TangentTensor such that for two vector fields $U, V$ on the ambient manifold along the null hypersurface, one has:

$$\nabla U V = D(U, V) + B(U, V) N$$

being $\nabla$ the ambient connection, $D$ the induced connection and $N$ the chosen rigging.

INPUT:

• screen – (default: None) an instance of Screen. If None, the default screen is used

OUTPUT:

• an instance of TangentTensor
EXAMPLES:

A degenerate hyperplane the 4-dimensional Minkowski space $\mathbb{R}^4_1$:

```python
sage: M = Manifold(4, 'M', structure="Lorentzian")
sage: X.<t,x,y,z> = M.chart()
sage: S = Manifold(3, 'S', ambient=M, structure='degenerate_metric')
sage: X_S.<u,v,w> = S.chart()
sage: Phi = S.diff_map(M, {(X_S, X): [u, u, v, w]},
...: name='Phi', latex_name=r'\Phi');
sage: Phi_inv = M.diff_map(S, {(X, X_S): [x,y, z]}, name='Phi_inv',
...: latex_name=r'\Phi^{-1}');
sage: S.set_immersion(Phi, inverse=Phi_inv); S.declare_embedding()
sage: g = M.metric()
sage: g[0,0], g[1,1], g[2,2], g[3,3] = -1,1,1,1
sage: S.set_transverse(rigging=x)
sage: xi = M.vector_field(); xi[0] = 1; xi[1] = 1
sage: Sc = S.screen('Sc', (U,V), xi);  # long time
sage: B = S.second_fundamental_form();  # long time
sage: B.display()  # long time
B = 0
```

`first_fundamental_form()`

Return the restriction of the ambient metric on vector field along the submanifold and tangent to it. It is difference from induced metric who gives the pullback of the ambient metric on the submanifold.

OUTPUT:

- the first fundamental form, as an instance of `TangentTensor`

EXAMPLES:

A degenerate hyperplane the 4-dimensional Minkowski space $\mathbb{R}^4_1$:

```python
sage: M = Manifold(4, 'M', structure="Lorentzian")
sage: X.<t,x,y,z> = M.chart()
sage: S = Manifold(3, 'S', ambient=M, structure='degenerate_metric')
sage: X_S.<u,v,w> = S.chart()
sage: Phi = S.diff_map(M, {(X_S, X): [u, u, v, w]},
...: name='Phi', latex_name=r'\Phi');
sage: Phi_inv = M.diff_map(S, {(X, X_S): [x,y, z]}, name='Phi_inv',
...: latex_name=r'\Phi^{-1}');
sage: S.set_immersion(Phi, inverse=Phi_inv); S.declare_embedding()
sage: g = M.metric()
sage: g[0,0], g[1,1], g[2,2], g[3,3] = -1,1,1,1
sage: S.set_transverse(rigging=x)
sage: xi = M.vector_field(); xi[0] = 1; xi[1] = 1
sage: Sc = S.screen('Sc', (U,V), xi);  # long time
sage: h = S.first_fundamental_form()  # long time
```

`gauss_curvature` (screen=None)

Gauss curvature is the product of all eigenfunctions of the shape operator.

INPUT:

- `screen` (default: None) an instance of `Screen`. If None the default screen is used.

OUTPUT:
• a scalar function on self

EXAMPLES:

A degenerate hyperplane the 4-dimensional Minkowskian space $\mathbb{R}_1^4$:

```python
sage: M = Manifold(4, 'M', structure='Lorentzian')
sage: X.<t,x,y,z> = M.chart()
sage: S = Manifold(3, 'S', ambient=M, structure='degenerate_metric')
sage: X_S.<u,v,w> = S.chart()
sage: Phi = S.diff_map(M, {(X_S, X): [u, u, v, w]}, name='Phi', latex_name=r'\Phi');
.....:   latex_name=r'\Phi');
sage: Phi_inv = M.diff_map(S, {(X, X_S): [x,y, z]}, name='Phi_inv',
.....:   latex_name=r'\Phi^{-1}');
sage: S.set_immersion(Phi, inverse=Phi_inv); S.declare_embedding()
sage: g = M.metric()
sage: g[0,0], g[1,1], g[2,2], g[3,3] = -1,1,1,1
sage: S.set_transverse(rigging=x)
sage: xi = M.vector_field(); xi[0] = 1; xi[1] = 1
sage: Sc = S.screen('Sc', (U,V), xi); # long time
sage: K = S.gauss_curvature(); # long time
sage: K.display(); # long time
S → ℝ
(u, v, w) ↦ 0
```

**induced_metric()**

Return the pullback of the ambient metric.

**OUTPUT:**

• induced metric, as an instance of `DegenerateMetric`

**EXAMPLES:**

Section of the lightcone of the Minkowski space with a hyperplane passing through the origin:

```python
sage: M = Manifold(4, 'M', structure='Lorentzian')
sage: X.<t,x,y,z> = M.chart()
sage: S = Manifold(2, 'S', ambient=M, structure='degenerate_metric')
sage: X_S.<u,v> = S.chart()
sage: Phi = S.diff_map(M, {(X_S, X): [sqrt(u^2+v^2), u, v, 0]},
.....:   name='Phi', latex_name=r'\Phi');
sage: Phi_inv = M.diff_map(S, {(X, X_S): [x, y]}, name='Phi_inv',
.....:   latex_name=r'\Phi^{-1}');
sage: S.set_immersion(Phi, inverse=Phi_inv); S.declare_embedding()
sage: g = M.metric()
sage: g[0,0], g[1,1], g[2,2], g[3,3] = -1,1,1,1
sage: h = S.induced_metric(); h # long time
degenerate metric gamma on the 2-dimensional degenerate submanifold S embedded in 4-dimensional differentiable manifold M
```

**is_tangent(v)**

Determine whether a vector field on the ambient manifold along `self` is tangent to `self` or not.

**INPUT:**

• v – field on the ambient manifold along `self`

**OUTPUT:**

• True if v is everywhere tangent to `self` or False if not
EXAMPLES:

A degenerate hyperplane the 4-dimensional Minkowski space \( \mathbb{R}^4_1 \):

```python
sage: M = Manifold(4, 'M', structure='Lorentzian')
sage: X.<t,x,y,z> = M.chart()
sage: S = Manifold(3, 'S', ambient=M, structure='degenerate_metric')
sage: X_S.<u,v,w> = S.chart()
sage: Phi = S.diff_map(M, {(X_S, X): [u, u, v, w]},
                          name='Phi', latex_name=r'\Phi')
sage: Phi_inv = M.diff_map(S, {(X, X_S): [x,y, z]}, name='Phi_inv',
                          latex_name=r'\Phi^{-1}')
sage: S.set_immersion(Phi, inverse=Phi_inv); S.declare_embedding()
sage: g = M.metric()
sage: g[0,0], g[1,1], g[2,2], g[3,3] = -1,1,1,1
sage: v = M.vector_field(); v[1] = 1
sage: S.set_transverse(rigging=v)
sage: xi = M.vector_field(); xi[0] = 1; xi[1] = 1
sage: Sc = S.screen('Sc', (U,V), xi)  # long time
sage: S.is_tangent(xi.along(Phi))  # long time
True
sage: S.is_tangent(v.along(Phi))  # long time
False
```

`list_of_screens()`

Return the default screen distribution.

OUTPUT:

- an instance of `Screen`

EXAMPLES:

A degenerate hyperplane the 4-dimensional Minkowski space \( \mathbb{R}^4_1 \):

```python
sage: M = Manifold(4, 'M', structure='Lorentzian')
sage: X.<t,x,y,z> = M.chart()
sage: S = Manifold(3, 'S', ambient=M, structure='degenerate_metric')
sage: X_S.<u,v,w> = S.chart()
sage: Phi = S.diff_map(M, {(X_S, X): [u, u, v, w]},
                          name='Phi', latex_name=r'\Phi')
sage: Phi_inv = M.diff_map(S, {(X, X_S): [x,y, z]}, name='Phi_inv',
                          latex_name=r'\Phi^{-1}')
sage: S.set_immersion(Phi, inverse=Phi_inv); S.declare_embedding()
sage: g = M.metric()
sage: g[0,0], g[1,1], g[2,2], g[3,3] = -1,1,1,1
sage: S.set_transverse(rigging=x)
sage: xi = M.vector_field(); xi[0] = 1; xi[1] = 1
sage: Sc = S.screen('Sc', (U,V), xi)  # long time
sage: S.list_of_screens()  # long time
{'Sc': screen distribution Sc along the degenerate hypersurface S embedded in 4-dimensional differentiable manifold M mapped into the 4-dimensional Lorentzian manifold M}
```

`mean_curvature(screen=None)`

Mean curvature is the sum of principal curvatures. This method is implemented only for hypersurfaces.

INPUT:
• screen – (default: None) an instance of Screen. If None the default screen is used.

OUTPUT:

• the mean curvature, as a scalar field on the submanifold

EXAMPLES:

A degenerate hyperplane the 4-dimensional Minkowski space $\mathbb{R}^4_1$:

```python
sage: M = Manifold(4, 'M', structure="Lorentzian")
sage: X.<t,x,y,z> = M.chart()
sage: S = Manifold(3, 'S', ambient=M, structure='degenerate_metric')
sage: X_S.<u,v,w> = S.chart()
sage: Phi = S.diff_map(M, {(X_S, X): [u, u, v, w]},
                   name='Phi', latex_name=r'\Phi');
sage: Phi_inv = M.diff_map(S, {(X, X_S): [x,y, z]}, name='Phi_inv',
                    latex_name=r'\Phi^{-1}');
sage: S.set_immersion(Phi, inverse=Phi_inv); S.declare_embedding()
sage: g = M.metric()
sage: g[0,0], g[1,1], g[2,2], g[3,3] = -1,1,1,1
sage: S.set_transverse(rigging=x)
sage: xi = M.vector_field(); xi[0] = 1; xi[1] = 1
sage: Sc = S.screen('Sc', (U,V), xi); 
# long time
sage: m = S.mean_curvature(); m 
# long time
Scalar field on the degenerate hypersurface $S$ embedded in 4-dimensional differentiable manifold $M$
sage: m.display() 
# long time
S → $\mathbb{R}$
$(u, v, w) \mapsto 0$
```

principal_directions (screen=None)

Principal directions are eigenvectors of the shape operator. This method is implemented only for hypersurfaces.

INPUT:

• screen – (default: None) an instance of Screen. If None default screen is used.

OUTPUT:

• list of pairs (vector field, scalar field) representing the principal directions and the associated principal curvatures

EXAMPLES:

A degenerate hyperplane the 4-dimensional Minkowski space $\mathbb{R}^4_1$:

```python
sage: M = Manifold(4, 'M', structure="Lorentzian")
sage: X.<t,x,y,z> = M.chart()
sage: S = Manifold(3, 'S', ambient=M, structure='degenerate_metric')
sage: X_S.<u,v,w> = S.chart()
sage: Phi = S.diff_map(M, {(X_S, X): [u, u, v, w]},
                   name='Phi', latex_name=r'\Phi');
sage: Phi_inv = M.diff_map(S, {(X, X_S): [x,y, z]}, name='Phi_inv',
                    latex_name=r'\Phi^{-1}');
sage: S.set_immersion(Phi, inverse=Phi_inv); S.declare_embedding()
sage: g = M.metric()
sage: g[0,0], g[1,1], g[2,2], g[3,3] = -1,1,1,1
sage: S.set_transverse(rigging=x)
(continues on next page)
```

(continues on next page)
projection \((\text{tensor, screen}=\text{None})\)

For a given tensor \(T\) of type \((r,1)\) on the ambient manifold, this method returns the tensor \(T'\) of type \((r,1)\) such that for \(r\) vector fields \(v_1, \ldots, v_r\), \(T'(v_1, \ldots, v_r)\) is the projection of \(T(v_1, \ldots, v_r)\) on \(\text{self}\) along the bundle spanned by the transversal vector fields provided by \(\text{set_transverse()}\).

INPUT:

- \(\text{tensor}\) – a tensor of type \((r,1)\) on the ambient manifold

OUTPUT:

- a tensor of type \((r,1)\) on the ambient manifold along \(\text{self}\)

EXAMPLES:

A degenerate hyperplane the 4-dimensional Minkowski space \(\mathbb{R}^4_1\):

```python
sage: M = Manifold(4, 'M', structure='Lorentzian')
sage: X.<t,x,y,z> = M.chart()
sage: S = Manifold(3, 'S', ambient=M, structure='degenerate_metric')
sage: X_S.<u,v,w> = S.chart()
sage: Phi = S.diff_map(M, {(X_S, X): [u, u, v, w]}, name=Phi, latex_name=\r\Phi);
    ...
    latex_name=r'\Phi^{-1}');
sage: S.set_immersion(Phi, inverse=Phi_inv); S.declare_embedding()
sage: g = M.metric()
sage: g[0,0], g[1,1], g[2,2], g[3,3] = -1,1,1,1
sage: S.set_transverse(rigging=x)
sage: xi = M.vector_field(); xi[0] = 1; xi[1] = 1
sage: Sc = S.screen('Sc', (U,V), xi); # long time
sage: PD = S.adapted_frame() # long time
sage: PD[2][0].display(T) # long time
```

screen \((\text{name, screen, rad, latex_name}=\text{None})\)

For setting a screen distribution and vector fields of the radical distribution that will be used for computations

INPUT:

- \(\text{name}\) – string (default: \text{None}); name given to the screen
- \(\text{latex_name}\) – string (default: \text{None}); LaTeX symbol to denote the screen; if \text{None}, the LaTeX symbol is set to \text{name}
- \(\text{screen}\) – list or tuple of vector fields of the ambient manifold or chart function; of the ambient manifold in the latter case, the corresponding gradient vector field with respect to the ambient metric is calculated; the vectors must be linearly independent, tangent to the submanifold but not normal
- \(\text{rad}\) – list or tuple of vector fields of the ambient manifold or chart function; of the ambient manifold in the latter case, the corresponding gradient vector field with respect to the ambient metric is calculated; the vectors must be linearly independent, tangent and normal to the submanifold

EXAMPLES:
A degenerate hyperplane the 4-dimensional Minkowski space $\mathbb{R}^4_1$:

```python
sage: M = Manifold(4, 'M', structure="Lorentzian")
sage: X.<t,x,y,z> = M.chart()
sage: S = Manifold(3, 'S', ambient=M, structure='degenerate_metric')
sage: X_S.<u,v,w> = S.chart()
sage: Phi = S.diff_map(M, {(X_S, X): [u, u, v, w]},
                  name='Phi', latex_name=r'\Phi');
sage: Phi_inv = M.diff_map(S, {(X, X_S): [x,y, z]}, name =Phi_inv,
                  latex_name=r'\Phi^{-1}');
sage: S.set_immersion(Phi, inverse =Phi_inv); S .declare_embedding()
sage: g = M.metric()
sage: g[0,0], g[ 1,1], g[ 2,2], g[ 3,3] = -1,1,1,1
sage: S.set_transverse(rigging=x)
sage: xi = M.vector_field(); xi[ 0] = 1; xi[ 1] = 1
sage: Sc = S.screen('Sc', (U,V), xi); Sc  # long time
screen distribution Sc along the degenerate hypersurface S embedded in 4-dimensional differentiable manifold M mapped into the 4-dimensional Lorentzian manifold M
```

**screen_projection** (tensor, screen=None)

For a given tensor $T$ of type $(r,1)$ on the ambient manifold, this method returns the tensor $T'$ of type $(r,1)$ such that for $r$ vector fields $v_1,\ldots,v_r$, $T'(v_1,\ldots,v_r)$ is the projection of $T(v_1,\ldots,v_r)$ on the bundle spanned by $\text{screen}$ along the bundle spanned by the transversal plus the radical vector fields provided.

**INPUT:**
- tensor—a tensor of type $(r,1)$ on the ambient manifold

**OUTPUT:**
- a tensor of type $(r,1)$ on the ambient manifold

**EXAMPLES:**

A degenerate hyperplane the 4-dimensional Minkowski space $\mathbb{R}^4_1$:

```python
sage: M = Manifold(4, 'M', structure="Lorentzian")
sage: X.<t,x,y,z> = M.chart()
sage: S = Manifold(3, 'S', ambient=M, structure='degenerate_metric')
sage: X_S.<u,v,w> = S.chart()
sage: Phi = S.diff_map(M, {(X_S, X): [u, u, v, w]},
                  name='Phi', latex_name=r'\Phi');
sage: Phi_inv = M.diff_map(S, {(X, X_S): [x,y, z]}, name =Phi_inv,
                  latex_name=r'\Phi^{-1}');
sage: S.set_immersion(Phi, inverse =Phi_inv); S .declare_embedding()
sage: g = M.metric()
sage: g[0,0], g[ 1,1], g[ 2,2], g[ 3,3] = -1,1,1,1
sage: S.set_transverse(rigging=x)
sage: xi = M.vector_field(); xi[ 0] = 1; xi[ 1] = 1
sage: U1 = S.screen_projection(U);  # long time
screen distribution Sc along the degenerate hypersurface S embedded in 4-dimensional differentiable manifold M mapped into the 4-dimensional Lorentzian manifold M
```

**second_fundamental_form** (screen=None)

This method is implemented only for null hypersurfaces. The method returns a tensor $B$ of type $(0,2)$ instance of $\text{TangentTensor}$ such that for two vector fields $U,V$ on the ambient manifold along the null
hypothesizing, one has:

\[ \nabla_U V = D(U, V) + B(U, V) N \]

being \( \nabla \) the ambient connection, \( D \) the induced connection and \( N \) the chosen rigging.

**INPUT:**

- **screen** – (default: None) an instance of \( \text{Screen} \). If None, the default screen is used

**OUTPUT:**

- an instance of \( \text{TangentTensor} \)

**EXAMPLES:**

A degenerate hyperplane the 4-dimensional Minkowski space \( \mathbb{R}^4_1 \):

```python
sage: M = Manifold(4, 'M', structure='Lorentzian')
sage: X.<t,x,y,z> = M.chart()

sage: S = Manifold(3, 'S', ambient=M, structure='degenerate_metric')
sage: X_S.<u,v,w> = S.chart()

sage: Phi = S.diff_map(M, {(X_S, X): [u, u, v, w]},
                  ....: name='Phi', latex_name=r'\Phi');

sage: Phi_inv = M.diff_map(S, {(X, X_S): [x,y, z]}, name='Phi_inv',
                       ....: latex_name=r'\Phi^{-1}');

sage: S.set_immersion(Phi, inverse=Phi_inv); S.declare_embedding()

sage: g = M.metric()

sage: g[0,0], g[1,1], g[2,2], g[3,3] = -1,1,1,1

sage: S.set_transverse(rigging=x)

sage: xi = M.vector_field(); xi[0] = 1; xi[1] = 1


sage: Sc = S.screen('Sc', (U,V), xi);  # long time

sage: B = S.second_fundamental_form();  # long time

sage: B.display()  # long time

B = 0
```

**set_transverse** (rigging=None, normal=None)

For setting a transversal distribution of the degenerate submanifold.

According to the type of the submanifold among the 4 possible types, one must enter a list of normal transversal vector fields and/or a list of transversal and not normal vector fields spanning a transverse distribution.

**INPUT:**

- **rigging** – list or tuple (default: None); list of vector fields of the ambient manifold or chart function; of the ambient manifold in the latter case, the corresponding gradient vector field with respect to the ambient metric is calculated; the vectors must be linearly independent, transversal to the submanifold but not normal

- **normal** – list or tuple (default: None); list of vector fields of the ambient manifold or chart function; of the ambient manifold in the latter case, the corresponding gradient vector field with respect to the ambient metric is calculated; the vectors must be linearly independent, transversal and normal to the submanifold

**EXAMPLES:**

The lightcone of the 3-dimensional Minkowski space \( \mathbb{R}^3_1 \):

```python
sage: M = Manifold(3, 'M', structure='Lorentzian')
sage: X.<t,x,y> = M.chart()
```

(continues on next page)
sage: S = Manifold(2, 'S', ambient=M, structure='degenerate_metric')
sage: X_S.<u,v> = S.chart()
sage: Phi = S.diff_map(M, {(X_S, X): [sqrt(u^2+v^2), u, v]}, name='Phi', latex_name=r'\Phi')
sage: Phi_inv = M.diff_map(S, {(X, X_S): [x, y]}, name=Phi_inv, latex_name=r'\Phi^{-1}')
sage: S.set_immersion(Phi, inverse=Phi_inv); S.declare_embedding()
sage: g = M.metric()
sage: g[0,0], g[1,1], g[2,2] = -1,1,1
sage: S.set_transverse(rigging=t)

shape_operator (screen=None)

This method is implemented only for hypersurfaces. shape operator is the projection of the Weingarten map on the screen distribution along the radical distribution.

INPUT:

- screen – (default: None) an instance of Screen. If None the default screen is used.

OUTPUT:

- tensor of type (1, 1) instance of TangentTensor

EXAMPLES:

A degenerate hyperplane the 4-dimensional Minkowski space $\mathbb{R}^4_1$:

sage: M = Manifold(4, 'M', structure='Lorentzian')
sage: X.<t,x,y,z> = M.chart()
sage: S = Manifold(3, 'S', ambient=M, structure='degenerate_metric')
sage: X_S.<u,v,w> = S.chart()
sage: Phi = S.diff_map(M, {(X_S, X): [u, u, v, w]}, name='Phi', latex_name=r'\Phi');
sage: Phi_inv = M.diff_map(S, {(X, X_S): [x,y, z]}, name=Phi_inv, latex_name=r'\Phi^{-1}');
sage: S.set_immersion(Phi, inverse=Phi_inv); S.declare_embedding()
sage: g = M.metric()
sage: g[0,0], g[1,1], g[2,2], g[3,3] = -1,1,1,1
sage: v = M.vector_field(); v[1] = 1
sage: S.set_transverse(rigging=v)
sage: xi = M.vector_field(); xi[0] = 1; xi[1] = 1
sage: Sc = S.screen('Sc', (U,V), xi); # long time
sage: SO = S.shape_operator(); # long time
sage: SO.display();
A^* = 0

weingarten_map (screen=None)

This method is implemented only for hypersurfaces. Weigarten map is the 1-form $W$ defined for a vector field $U$ tangent to self by

$$W(U) = \nabla_U \xi$$

being $\nabla$ the Levi-Civita connection of the ambient manifold and $\xi$ the chosen vector field spanning the radical distribution.

INPUT:

- screen – (default: None) an instance of Screen. If None the default screen is used.
OUTPUT:
• tensor of type (1, 1) instance of \texttt{TangentTensor}

EXAMPLES:
A degenerate hyperplane the 4-dimensional Minkowski space $\mathbb{R}^4_1$:

```python
sage: M = Manifold(4, 'M', structure='Lorentzian')
sage: X.<t,x,y,z> = M.chart()
sage: S = Manifold(3, 'S', ambient=M, structure='degenerate_metric')
sage: X_S.<u,v,w> = S.chart()
sage: Phi = S.diff_map(M, {(X_S, X): [u, u, v, w]},
                    name='Phi', latex_name=r'\Phi');
    sage: Phi_inv = M.diff_map(S, {(X, X_S): [x,y, z]}, name='Phi_inv',
                    latex_name=r'\Phi^{-1}');
    sage: S.set_immersion(Phi, inverse=Phi_inv); S.declare_embedding()
```

```python
sage: g = M.metric()
sage: g[0,0], g[1,1], g[2,2], g[3,3] = -1,1,1,1
sage: v = M.vector_field(); v[1] = 1
sage: S.set_transverse(rigging=v)
```

```python
class sage.manifolds.differentiable.degenerate_submanifold.Screen (submanifold, name, screen, rad, latex_name=None)
```

Bases: \texttt{VectorFieldModule}

Let $H$ be a lightlike submanifold embedded in a pseudo-Riemannian manifold $(M, g)$ with $\Phi$ the embedding map. A screen distribution is a complementary $S(TH)$ of the radical distribution $\text{Rad}(TM) = TH \cap TH^\perp$ in $TH$. One then has

$$TH = S(TH) \oplus_{\text{orth}} \text{Rad}(TH)$$

INPUT:
• \texttt{submanifold} – a lightlike submanifold, as an instance of \texttt{DegenerateSubmanifold}

• \texttt{name} – name given to the screen distribution

• \texttt{screen} – vector fields of the ambient manifold which span the screen distribution

• \texttt{rad} – vector fields of the ambient manifold which span the radical distribution

• \texttt{latex_name} – (default: None) \LaTeX\ symbol to denote the screen distribution; if None, it is formed from name

EXAMPLES:
The horizon of the Schwarzschild black hole:

```python
sage: M = Manifold(4, 'M', structure='Lorentzian')
sage: X_M.<t, r, th, ph> = M.chart()
    ....: M.chart(r"t r:(0,oo) th:(0,pi):\theta ph:(0,2*pi):\phi")
```

(continues on next page)
m
sage: g = M.metric()
sage: g[0,0], g[0,1], g[1,1], g[2,2], g[3,3] = 
    -1+2*m/r, 2*m/r, 1+2*m/r, r^2, r^2*sin(th)^2
sage: H = Manifold(3, 'H', ambient=M, structure='degenerate_metric')
sage: X_H.<ht,hth,hph> = 
    H.chart(r"ht:(-oo,oo):t hth:(0,pi): \theta hph:(0,2*pi): \phi")

sage: Phi = H.diff_map(M, {(X_H, X_M): [ht, 2*m,hth, hph]}, name=Phi, latex_name =r'\Phi')
sage: Phi_inv = M.diff_map(H, {(X_M, X_H): [t,th, ph]}, name=Phi_inv, latex_name =r'\Phi^{-1}')

sage: H.set_immersion(Phi, inverse =Phi_inv); H .declare_embedding()

sage: xi = M.vector_field(-1, 0, 0, 0)
sage: v = M.vector_field(r, -r, 0, 0)
sage: e1 = M.vector_field(0, 0, 1, 0)
sage: e2 = M.vector_field(0, 0, 0, 1)

A screen distribution for the Schwarzschild black hole horizon:

sage: H.set_transverse(rigging=v)
sage: S = H.screen('S', [e1, e2], (xi)); S  # long time
screen distribution S along the degenerate hypersurface H embedded in 4-dimensional differentiable manifold M mapped into the 4-dimensional Lorentzian manifold M

The corresponding normal tangent null vector field and null transversal vector field:

sage: xi = S.normal_tangent_vector(); xi.display()  # long time

sage: N = S.rigging(); N.display()  # long time

Those vector fields are normalized by $g(\xi, N) = 1$:

sage: g.along(Phi)(xi, N) .display()  # long time
g(\xi, N): H → R
(ht, hth, hph) ↦ 1

normal_tangent_vector()

Return either a list $\text{Rad}$ of vector fields spanning the radical distribution or (in case of a hypersurface) a normal tangent null vector field spanning the radical distribution.

OUTPUT:

- either a list of vector fields or a single vector field in case of a hypersurface

EXAMPLES:

A degenerate hyperplane the 4-dimensional Minkowski space $\mathbb{R}^4_1$:
....: \text{latex_name}=r'\Phi^{-1}');
\text{sage: } g = M.\text{metric()}
\text{sage: } g[0,0], g[1,1], g[2,2], g[3,3] = -1,1,1,1
\text{sage: } v = M.\text{vector_field(); } v[1] = 1
\text{sage: } S.\text{set_transverse(rigging=v)}
\text{sage: } x_i = M.\text{vector_field(); } x_i[0] = 1; \ x_i[1] = 1
\text{sage: } U = M.\text{vector_field(); } U[2] = 1; \ V = M.\text{vector_field(); } V[3] = 1
\text{sage: } Sc = S.\text{screen('Sc', (U,V), xi); } \quad \# \text{long time}
\text{sage: } \text{Rad} = Sc.\text{normal_tangent_vector(); } \text{Rad.display()} \quad \# \text{long time}
\text{xi} = a/\partial t + a/\partial x

\textbf{rigging()}

Return either a list Rad of vector fields spanning the complementary of the normal distribution $TH^\perp$ in the transverse bundle or (when $H$ is a null hypersurface) the null transversal vector field defined in [DB1996].

\textbf{OUTPUT:}

- either a list made by vector fields or a vector field in case of hypersurface

\textbf{EXAMPLES:}

A degenerate hyperplane the 4-dimensional Minkowski space $\mathbb{R}^4$: 

\texttt{sage: } M = \text{Manifold(4, 'M', structure='Lorentzian')}
\texttt{sage: } X.<t,x,y,z> = M.\text{chart()}
\texttt{sage: } S = \text{Manifold(3, 'S', ambient=M, structure='degenerate_metric')}
\texttt{sage: } X_S.<u,v,w> = S.\text{chart()}
\texttt{sage: } Phi = S.\text{diff_map}(M, \{(X_S, X): [u, u, v, w]},
\texttt{.....: name='Phi', latex_name=r'\Phi');
\texttt{sage: } Phi_inv = M.\text{diff_map}(S, \{(X, X_S): [x, y, z]}, name='Phi_inv',
\texttt{.....: latex_name=r'\Phi^{-1}');
\texttt{sage: } S.\text{set_immersion(Phi, inverse=Phi_inv), S.\text{declare_embedding()}
\texttt{sage: } g = M.\text{metric()}
\texttt{sage: } g[0,0], g[1,1], g[2,2], g[3,3] = -1,1,1,1
\texttt{sage: } v = M.\text{vector_field(); } v[1] = 1
\texttt{sage: } S.\text{set_transverse(rigging=v)}
\texttt{sage: } x_i = M.\text{vector_field(); } x_i[0] = 1; \ x_i[1] = 1
\texttt{sage: } U = M.\text{vector_field(); } U[2] = 1; \ V = M.\text{vector_field(); } V[3] = 1
\texttt{sage: } Sc = S.\text{screen('Sc', (U,V), xi); } \quad \# \text{long time}
\texttt{sage: } \text{rig} = Sc.\text{rigging(); } \text{rig.display()} \quad \# \text{long time}
\texttt{N} = -1/2 \partial/\partial t + 1/2 \partial/\partial x
4.1 Poisson tensors

AUTHORS:
• Tobias Diez (2021): initial version

**Bases:** MultivectorField

A Poisson bivector field \( \varpi \) on a differentiable manifold.

That is, at each point \( m \in M \), \( \varpi_m \) is a bilinear map of the type:

\[
\varpi_m : T^*_m M \times T^*_m M \rightarrow \mathbb{R}
\]

where \( T^*_m M \) stands for the cotangent space to the manifold \( M \) at the point \( m \), such that \( \varpi_m \) is skew-symmetric and the Schouten-Nijenhuis bracket (cf. \( \text{bracket()} \)) of \( \varpi \) with itself vanishes.

**INPUT:**
• manifold – module \( \mathfrak{X}(M) \) of vector fields on the manifold \( M \), or the manifold \( M \) itself
• name – (default: \( \text{varpi} \)) name given to the Poisson tensor
• latex_name – (default: \( \text{\varpi} \)) LaTeX symbol to denote the Poisson tensor; if None, it is formed from name

**EXAMPLES:**
A Poisson tensor on the 2-sphere:

```python
sage: M.<x,y> = manifolds.Sphere(2, coordinates='stereographic')
sage: stereoN = M.stereographic_coordinates(pole=north)
sage: stereoS = M.stereographic_coordinates(pole=south)
```
varpi is initialized by providing its single nonvanishing component w.r.t. the vector frame associated to stereoN, which is the default frame on M:

```sage
varpi[1, 2] = 1
```

The components w.r.t. the vector frame associated to stereoS are obtained thanks to the method `add_comp_by_continuation()`:

```sage
varpi.add_comp_by_continuation(stereoS.frame(),
.....: stereoS.domain().intersection(stereoN.domain()))
```

The Schouten-Nijenhuis bracket of a Poisson tensor with itself vanishes (this is trivial here, since M is 2-dimensional):

```sage
varpi.bracket(varpi).display()
[varpi,varpi] = 0
```

**hamiltonian_vector_field** *(function)*

Return the Hamiltonian vector field \( X_f \) generated by the given function \( f : M \to \mathbb{R} \).

The Hamiltonian vector field is defined by

\[
X_f = -\varpi^\sharp(df),
\]

where \( \varpi^\sharp : T^*M \to TM \) is given by \( \beta(\varpi^\sharp(\alpha)) = \varpi(\alpha, \beta) \).

**INPUT:**

- function – the function generating the Hamiltonian vector field

**EXAMPLES:**

```sage
M.<q, p> = EuclideanSpace(2)
sage: poisson = M.poisson_tensor('varpi')
sage: poisson.set_comp()[1,2] = -1
sage: f = M.scalar_field(function('f')(q, p), name='f')
sage: Xf = poisson.hamiltonian_vector_field(f)
sage: Xf.display()
Xf = df/dp e_q - df/dq e_p
```

**poisson_bracket** *(f, g)*

Return the Poisson bracket

\[
\{f, g\} = \varpi(df, dg)
\]

of the given functions.

**INPUT:**
• \( f \) – first function
• \( g \) – second function

EXAMPLES:

```
sage: M.<q, p> = EuclideanSpace(2)
sage: poisson = M.poisson_tensor('varpi')
sage: poisson.set_comp()[1,2] = -1
sage: f = M.scalar_field(function('f')(q, p), name='f')
sage: g = M.scalar_field(function('g')(q, p), name='g')
sage: poisson.poisson_bracket(f, g).display()
poisson(f, g): E^2 → ℝ
   (q, p) ↦ d(f)/dp*d(g)/dq - d(f)/dq*d(g)/dp
```

**sharp** *(form)*

Return the image of the given differential form under the map \( \varpi^#: T^* M \to TM \) defined by

\[
\beta(\varpi^#(\alpha)) = \varpi(\alpha, \beta).
\]

for all \( \alpha, \beta \in T^*_m M \).

In indices, \( \alpha^i = \varpi^{ij} \alpha_j \).

**INPUT:**

• `form` – the differential form to calculate its sharp of

**EXAMPLES:**

```
sage: M.<q, p> = EuclideanSpace(2)
sage: poisson = M.poisson_tensor('varpi')
sage: poisson.set_comp()[1,2] = -1
sage: a = M.one_form(1, 0, name='a')
sage: poisson.sharp(a).display()
    a_sharp = e_p
```

4.1. Poisson tensors
class sage.manifolds.differentiable.poisson_tensor.PoissonTensorFieldParal (manifold: DifferentiableManifold | VectorFieldModule, name: str | None = None, latex_name: str | None = None)

Bases: PoissonTensorField, MultivectorFieldParal

A Poisson bivector field \( \varpi \) on a parallelizable manifold.

INPUT:

- manifold – module \( \mathfrak{X}(M) \) of vector fields on the manifold \( M \), or the manifold \( M \) itself
- name – (default: varpi) name given to the Poisson tensor
- latex_name – (default: \( \varpi \)) \LaTeX\ symbol to denote the Poisson tensor; if None, it is formed from name

EXAMPLES:

Standard Poisson tensor on \( \mathbb{R}^2 \):

\[
\begin{align*}
sage: & M.<q, p> = EuclideanSpace(2) \\
sage: & varpi = M.poisson_tensor(name='varpi', latex_name=r'\varpi') \\
sage: & varpi[1,2] = -1 \\
sage: & varpi \\
\text{2-vector field varpi on the Euclidean plane } \mathbb{E}^2 \\
\end{align*}
\]
\[
\begin{align*}
sage: & varpi.display() \\
& varpi = -e_q\wedge e_p
\end{align*}
\]
4.2 Symplectic structures

The class \texttt{SymplecticForm} implements symplectic structures on differentiable manifolds over \( \mathbb{R} \). The derived class \texttt{SymplecticFormParal} is devoted to symplectic forms on a parallelizable manifold.

AUTHORS:

• Tobias Diez (2021) : initial version

REFERENCES:

• [AM1990]
• [RS2012]

\texttt{class sage.manifolds.differentiable.symplectic_form.SymplecticForm(manifold: Differentiable-Manifold | VectorField-Module, name: str | None = None, latex_name: str | None = None)}

**Bases:** \texttt{DiffForm}

A symplectic form on a differentiable manifold.

An instance of this class is a closed nondegenerate differential 2-form \( \omega \) on a differentiable manifold \( M \) over \( \mathbb{R} \).

In particular, at each point \( m \in M \), \( \omega_m \) is a bilinear map of the type:

\[
\omega_m : T_m M \times T_m M \to \mathbb{R},
\]

where \( T_m M \) stands for the tangent space to the manifold \( M \) at the point \( m \), such that \( \omega_m \) is skew-symmetric:

\[
\forall u, v \in T_m M, \quad \omega_m(v, u) = -\omega_m(u, v)
\]

and nondegenerate:

\[
(\forall v \in T_m M, \quad \omega_m(u, v) = 0) \implies u = 0.
\]

**Note:** If \( M \) is parallelizable, the class \texttt{SymplecticFormParal} should be used instead.

**INPUT:**

• \texttt{manifold} – module \( \mathcal{X}(M) \) of vector fields on the manifold \( M \), or the manifold \( M \) itself
• \texttt{name} – (default: omega) name given to the symplectic form
• \texttt{latex_name} – (default: None) LaTeX symbol to denote the symplectic form; if None, it is formed from name

**EXAMPLES:**

A symplectic form on the 2-sphere:

\begin{verbatim}
sage: M.<x,y> = manifolds.Sphere(2, coordinates='stereographic')
sage: stereoN = M.stereographic_coordinates(pole='north')
sage: stereoS = M.stereographic_coordinates(pole='south')
sage: omega = M.symplectic_form(name='omega', latex_name=r'\omega')
sage: omega
Symplectic form omega on the 2-sphere S^2 of radius 1 smoothly embedded in the Euclidean space E^3
\end{verbatim}
omega is initialized by providing its single nonvanishing component w.r.t. the vector frame associated to stereoN, which is the default frame on M:

```python
sage: omega[1, 2] = 1/(1 + x^2 + y^2)^2
```

The components w.r.t. the vector frame associated to stereoS are obtained thanks to the method `add_comp_by_continuation()`:

```python
sage: omega.add_comp_by_continuation(stereoS.frame(),
    stereoS.domain().intersection(stereoN.domain()))

sage: omega.display()
omega = (x^2 + y^2 + 1)^(-2) dx∧dy
sage: omega.display(stereoS)
omega = -1/(xp^4 + yp^4 + 2*(xp^2 + 1)*yp^2 + 2*xp^2 + 1) dxp∧dyp
```

omega is an exact 2-form (this is trivial here, since M is 2-dimensional):

```python
sage: diff(omega).display()
domega = 0
```

**flat (vector_field)**

Return the image of the given differential form under the map $\omega^\flat : TM \rightarrow T^* M$ defined by

$$< \omega^\flat(X), Y >= \omega_m(X, Y)$$

for all $X, Y \in T_m M$.

In indices, $X_i = \omega_{ji} X^j$.

**INPUT:**

- vector_field – the vector field to calculate its flat of

**EXAMPLES:**

```python
sage: M = manifolds.StandardSymplecticSpace(2)
sage: omega = M.symplectic_form()
sage: X = M.vector_field_module().an_element()
sage: X.set_name('X')
sage: X.display()
X = 2 e_q + 2 e_p
sage: X.flat(omega).display()
X_flat = 2 dq - 2 dp
```

**hamiltonian_vector_field (function)**

The Hamiltonian vector field $X_f$ generated by a function $f : M \rightarrow \mathbb{R}$.

The Hamiltonian vector field is defined by

$$X_f.\omega + df = 0.$$ 

**INPUT:**

- function – the function generating the Hamiltonian vector field

**EXAMPLES:**
```python
sage: M = manifolds.StandardSymplecticSpace(2)
sage: omega = M.symplectic_form()
sage: f = M.scalar_field({ chart: function('f')(*chart[:]) for chart in M.
→atlas() }, name='f')
sage: f.display()
\( f: \mathbb{R}^2 \to \mathbb{R} \)
\begin{array}{ll}
  (q, p) & \mapsto f(q, p)
\end{array}
sage: Xf = omega.hamiltonian_vector_field(f)
sage: Xf.display()
\( Xf = \frac{\partial(f)}{\partial p} e_q - \frac{\partial(f)}{\partial q} e_p \)
```

**hodge_star** *(pform)*

Compute the Hodge dual of a differential form with respect to the symplectic form.

See `hodge_dual()` for the definition and more details.

**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 any form on the symplectic vector space \( \mathbb{R}^2 \):

```python
sage: M = manifolds.StandardSymplecticSpace(2)
sage: omega = M.symplectic_form()
sage: a = M.one_form(1, 0, name='a')
sage: omega.hodge_star(a).display()
\(*a = dq\)
sage: b = M.one_form(0, 1, name='b')
sage: omega.hodge_star(b).display()
\(*b = dp\)
sage: f = M.scalar_field(1, name='f')
sage: omega.hodge_star(f).display()
\(*f = -dq \wedge dp\)
sage: omega.hodge_star(omega).display()
\(*omega: \mathbb{R}^2 \to \mathbb{R} \)
\begin{array}{ll}
  (q, p) & \mapsto 1
\end{array}
```

**on_forms** *(first, second)*

Return the contraction of the two forms with respect to the symplectic form.

The symplectic form \( \omega \) gives rise to a bilinear form, also denoted by \( \omega \) on the space of 1-forms by

\[
\omega(\alpha, \beta) = \omega(\alpha^\sharp, \beta^\sharp),
\]

where \( \alpha^\sharp \) is the dual of \( \alpha \) with respect to \( \omega \), see `up()`. This bilinear form induces a bilinear form on the space of all forms determined by its value on decomposable elements as:

\[
\omega(\alpha_1 \wedge \ldots \wedge \alpha_p, \beta_1 \wedge \ldots \wedge \beta_p) = \det(\omega(\alpha_i, \beta_j)).
\]

**INPUT:**
- `first` — a \( p \)-form \( \alpha \)
- `second` — a \( p \)-form \( \beta \)
OUTPUT:

• the scalar field $\omega(\alpha, \beta)$

EXAMPLES:

sage: M = manifolds.StandardSymplecticSpace(2)  
sage: omega = M.symplectic_form()  
sage: a = M.one_form(1, 0, name='a')  
sage: b = M.one_form(0, 1, name='b')  
sage: omega.on_forms(a, b).display()  
\[ R^2 \to \mathbb{R} (q, p) \mapsto -1 \]

**poisson**(expansion_symbol=None, order=1)

Return the Poisson tensor associated with the symplectic form.

INPUT:

• expansion_symbol – (default: None) symbolic variable; if specified, the inverse will be expanded in power series with respect to this variable (around its zero value)

• order – integer (default: 1); the order of the expansion if expansion_symbol is not None; the order is defined as the degree of the polynomial representing the truncated power series in expansion_symbol; currently only first order inverse is supported

If expansion_symbol is set, then the zeroth order symplectic form must be invertible. Moreover, subsequent calls to this method will return a cached value, even when called with the default value (to enable computation of derived quantities). To reset, use `_del_derived()`.

OUTPUT:

• the Poisson tensor, as an instance of *PoissonTensorField*()

EXAMPLES:

Poisson tensor of 2-dimensional symplectic vector space:

sage: M = manifolds.StandardSymplecticSpace(2)  
sage: omega = M.symplectic_form()  
sage: poisson = omega.poisson(); poisson  
2-vector field poisson_omega on the Standard symplectic space $R^2$

sage: poisson.display()  
poisson_omega = -e_q\wedge e_p

**poisson_bracket**($f, g$)

Return the Poisson bracket $\{f, g\} = \omega(X_f, X_g)$

of the given functions.

INPUT:

• $f$ – function inserted in the first slot

• $g$ – function inserted in the second slot

EXAMPLES:

sage: M.<q, p> = EuclideanSpace(2)  
sage: poisson = M.poisson_tensor('varpi')  
sage: poisson.set_comp()[1,2] = -1  
sage: f = M.scalar_field({ chart: function('f')(*chart[:]) for chart in M.\rightarrow atlas() }, name='f')  
sage: g = M.scalar_field({ chart: function('g')(*chart[:]) for chart in M.\rightarrow atlas() }, name='g')
\begin{verbatim}
set atlas(), name='g')
sage: poisson.poisson_bracket(f, g).display()
poisson(f, g): E^2 \to \mathbb{R}
(q, p) \mapsto d(f)/dp*d(g)/dq - d(f)/dq*d(g)/dp
\end{verbatim}

**restrict** *(subdomain, dest_map=None)*

Return the restriction of the symplectic form to some subdomain.

If the restriction has not been defined yet, it is constructed here.

**INPUT:**

- **subdomain** – open subset $U$ of the symplectic form’s domain
- **dest_map** – (default: None) smooth destination map $\Phi : U \to V$, where $V$ is a subdomain of the symplectic form’s domain. If None, the restriction of the initial vector field module is used.

**OUTPUT:**

- the restricted symplectic form.

**EXAMPLES:**

```sage
sage: M = Manifold(6, 'M')
sage: omega = M.symplectic_form()
sage: U = M.open_subset('U')
sage: omega.restrict(U)
2-form omega on the Open subset U of the 6-dimensional differentiable manifold M
```

**sharp** *(form)*

Return the image of the given differential form under the map $\omega^\sharp : T^* M \to TM$ defined by

$$\omega(\omega^\sharp(\alpha), X) = \alpha(X)$$

for all $X \in T_mM$ and $\alpha \in T^*_mM$. The sharp map is inverse to the flat map.

In indices, $\alpha^i = \varpi^{ij}\alpha_j$, where $\varpi$ is the Poisson tensor associated with the symplectic form.

**INPUT:**

- **form** – the differential form to calculate its sharp of

**EXAMPLES:**

```sage
sage: M = manifolds.StandardSymplecticSpace(2)
sage: omega = M.symplectic_form()
sage: X = M.vector_field_module().an_element()
sage: alpha = omega.flat(X)
sage: alpha.set_name('alpha')
sage: alpha.display()
alpha = 2 dq - 2 dp
sage: omega.sharp(alpha).display()
alpha_sharp = 2 e_q + 2 e_p
```

**volume_form** *(contra=0)*

Liouville volume form $\frac{1}{n!} \omega^n$ associated with the symplectic form $\omega$, where $2n$ is the dimension of the manifold.

**INPUT:**
• contra – (default: 0) number of contravariant indices of the returned tensor

OUTPUT:
• if contra = 0: volume form associated with the symplectic form
• if contra = k, with 1 ≤ k ≤ n, the tensor field of type (k,n-k) formed from ε by raising the first k indices with the symplectic form (see method up())

EXAMPLES:
Volume form on $\mathbb{R}^4$:

```python
sage: M = manifolds.StandardSymplecticSpace(4)
sage: omega = M.symplectic_form()
sage: vol = omega.volume_form(); vol
4-form μ_omega on the Standard symplectic space R4
sage: vol.display()
μ_omega = dq1∧dp1∧dq2∧dp2
```

```python
static wrap(form, name=None, latex_name=None)
```

Define the symplectic form from a differential form.

INPUT:
• form – differential 2-form

EXAMPLES:
Volume form on the sphere as a symplectic form:

```python
sage: from sage.manifolds.differentiable.symplectic_form import SymplecticForm
sage: M = manifolds.Sphere(2, coordinates='stereographic')
sage: vol_form = M.induced_metric().volume_form()
    # long...
sage: omega = SymplecticForm.wrap(vol_form, 'omega', 'r\omega')
    # long...
sage: omega.display()
    # long...
sage: time omega = -4/(y1^4 + y2^4 + 2*(y1^2 + 1)*y2^2 + 2*y1^2 + 1) dy1∧dy2
```

```python
class sage.manifolds.differentiable.symplectic_form.SymplecticFormParal

Bases: SymplecticForm, DiffFormParal

A symplectic form on a parallelizable manifold.
```
Note: If $M$ is not parallelizable, the class \texttt{SymplecticForm} should be used instead.

INPUT:
- manifold -- module $\mathfrak{X}(M)$ of vector fields on the manifold $M$, or the manifold $M$ itself
- name -- (default: omega) name given to the symplectic form
- latex_name -- (default: None) \LaTeX{} symbol to denote the symplectic form; if None, it is formed from name

EXAMPLES:
Standard symplectic form on $\mathbb{R}^2$:

```python
sage: M.<q, p> = EuclideanSpace(name="R2", latex_name=r"\mathbb{R}^2")
sage: omega = M.symplectic_form(name=omega, latex_name=r'\omega')
```

\texttt{poisson} (expansion_symbol=None, order=1)

Return the Poisson tensor associated with the symplectic form.

INPUT:
- expansion_symbol -- (default: None) symbolic variable; if specified, the inverse will be expanded in power series with respect to this variable (around its zero value)
- order -- integer (default: 1); the order of the expansion if \texttt{expansion_symbol} is not None; the \texttt{order} is defined as the degree of the polynomial representing the truncated power series in \texttt{expansion_symbol}; currently only first order inverse is supported

If \texttt{expansion_symbol} is set, then the zeroth order symplectic form must be invertible. Moreover, subsequent calls to this method will return a cached value, even when called with the default value (to enable computation of derived quantities). To reset, use \_del_derived().

OUTPUT:
- the Poisson tensor, as an instance of \texttt{PoissonTensorFieldParal()}

EXAMPLES:
Poisson tensor of 2-dimensional symplectic vector space:

```python
sage: from sage.manifolds.differentiable.symplectic_form import... SymplecticFormParal
sage: M.<q, p> = EuclideanSpace(2, "R2", r"\mathbb{R}^2", symbols=r"q:p:p")
sage: omega = SymplecticFormParal(M, 'omega', r'\omega')
sage: omega[1,2] = -1
```

\texttt{restrict} (subdomain, dest_map=None)

Return the restriction of the symplectic form to some subdomain.

If the restriction has not been defined yet, it is constructed here.
INPUT:

- subdomain – open subset \( U \) of the symplectic form’s domain
- dest_map – (default: None) smooth destination map \( \Phi : U \to V \), where \( V \) is a subdomain of the symplectic form’s domain If None, the restriction of the initial vector field module is used.

OUTPUT:

- the restricted symplectic form.

EXAMPLES:

Restriction of the standard symplectic form on \( \mathbb{R}^2 \) to the upper half plane:

```markdown
sage: from sage.manifolds.differentiable.symplectic_form import SymplecticFormParal
sage: M = EuclideanSpace(2, "R2", r"\mathbb{R}^2", symbols=r"q:q p:p")
sage: X.<q, p> = M.chart()
sage: omega = SymplecticFormParal(M, 'omega', r'\omega')
sage: omega[1,2] = -1
sage: U = M.open_subset('U', coord_def={X: q>0})
sage: omegaU = omega.restrict(U); omegaU
Symplectic form omega on the Open subset U of the Euclidean plane R2
sage: omegaU.display()
omega = -dq∧dp
```

### 4.3 Symplectic vector spaces

AUTHORS:

- Tobias Diez (2021): initial version
class sage.manifolds.differentiable.examples.symplectic_space.StandardSymplecticSpace(dimension: int, name: str | None = None, latex_name: str | None = None, coordinates: str = 'Cartesian', symbols: str | None = None, symplectic_name: str | None = 'omega', symplectic_latex_name: str | None = None, start_index: int = 1, base_manifold: StandardSymplecticSpace | None = None, names: Tuple[str] | None = None)
Bases: EuclideanSpace

The vector space $\mathbb{R}^{2n}$ equipped with its standard symplectic form.

`symplectic_form()`

Return the symplectic form.

EXAMPLES:

Standard symplectic form on $\mathbb{R}^2$:

```
sage: M.<q, p> = manifolds.StandardSymplecticSpace(2, symplectic_name='omega')
sage: omega = M.symplectic_form()
sage: omega.display()
omega = -dq ∧ dp
```
This module defines helper functions which are used for simplifications and display of symbolic expressions.

**AUTHORS:**

- Michal Bejger (2015): class `ExpressionNice`
- Travis Scrimshaw (2016): review tweaks
- Marius Gerbershagen (2022): skip simplification of expressions with a single number or symbolic variable

**class** `sage.manifolds.utilities.ExpressionNice(ex)`

**Bases:** `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 - \frac{d(h)}{dz})^2 + x*d^2(f)/ dx dy
```

(continues on next page)
An example when function variables are themselves functions:

```python
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
```

Note that $D[1](g)(x, f(x, y))$ is rendered as $d(g)/d(f(x, y))$.

An example with multiple differentiations:

```python
sage: fun = f.diff(x,x,y,y,x)*x
sage: fun
x*diff(f(x, y), x, x, x, y, y)
```

Parentheses are added around powers of partial derivatives to avoid any confusion:

```python
sage: fun = f.diff(y)^2
sage: fun
diff(f(x, y), y)^2
```

The explicit mention of function arguments can be omitted for the sake of brevity:

```python
sage: fun = fun*f
sage: fun
f(x, y)*(d(f)/dy)^2
```

```python
sage: Manifold.options.omit_function_arguments=True
sage: fun
f*(d(f)/dy)^2
```

```python
sage: Manifold.options._reset()
sage: fun
f(x, y)*(d(f)/dy)^2
```

```python
sage: latex(Manifold.options._reset())
```

```python
f(x, y)*(d(f)/dy)^2
```
class sage.manifolds.utilities.SimplifyAbsTrig(ex)

Bases: ExpressionTreeWalker

Class for simplifying absolute values of cosines or sines (in the real domain), by walking the expression tree.

The end user interface is the function simplify_abs_trig().

INPUT:

• ex – a symbolic expression

EXAMPLES:

Let us consider the following symbolic expression with some assumption on the range of the variable \( x \):

```python
sage: assume(pi/2 < x, x < pi)
sage: a = abs(cos(x)) + abs(sin(x))
```

The method `simplify_full()` is ineffective on such an expression:

```python
sage: a.simplify_full()
abs(cos(x)) + abs(sin(x))
```

We construct a `SimplifyAbsTrig` object \( s \) from the symbolic expression \( a \):

```python
sage: from sage.manifolds.utilities import SimplifyAbsTrig
sage: s = SimplifyAbsTrig(a)
```

We use the `__call__` method to walk the expression tree and produce a correctly simplified expression, given that \( x \in (\pi/2, \pi) \):

```python
sage: s()
-cos(x) + sin(x)
```

Calling the simplifier \( s \) with an expression actually simplifies this expression:

```python
sage: s(a)  # same as s() since s is built from a
-cos(x) + sin(x)
sage: s(abs(cos(x/2)) + abs(sin(x/2)))  # pi/4 < x/2 < pi/2
-cos(1/2*x) + sin(1/2*x)
sage: s(abs(cos(2*x)) + abs(sin(2*x)))  # pi < 2 x < 2*pi
abs(cos(2*x)) - sin(2*x)
sage: s(abs(sin(2+abs(cos(x))))))  # nested abs(sin_or_cos(...))
sin(-cos(x) + 2)
```

See also:

`simplify_abs_trig()` for more examples with `SimplifyAbsTrig` at work.

composition(ex, operator)

This is the only method of the base class `ExpressionTreeWalker` that is reimplemented, since it manages the composition of \( \text{abs} \) with \( \text{cos} \) or \( \text{sin} \).

INPUT:

• ex – a symbolic expression

• operator – an operator

OUTPUT:
• a symbolic expression, equivalent to \( \text{ex} \) with \( \text{abs}(\cos(...)) \) and \( \text{abs}(\sin(...)) \) simplified, according to the range of their argument.

EXAMPLES:

```python
sage: from sage.manifolds.utilities import SimplifyAbsTrig
sage: assume(-pi/2 < x, x<0)
sage: a = abs(sin(x))
sage: s = SimplifyAbsTrig(a)
sage: a.operator()
abs
sage: s.composition(a, a .operator())
sin(-x)

sage: a = exp(function('f')(x))  # no abs(sin_or_cos(...))
sage: a.operator()
exp
sage: s.composition(a, a .operator())
e^f(x)

sage: forget()  # no longer any assumption on x
sage: a = abs(cos(sin(x)))  # simplifiable since -1 <= sin(x) <= 1
sage: s.composition(a, a .operator())
cos(sin(x))
sage: a = abs(sin(cos(x)))  # not simplifiable
sage: s.composition(a, a .operator())
abs(sin(cos(x)))
```

```python
class sage.manifolds.utilities.SimplifySqrtReal(ex)
Bases: ExpressionTreeWalker

Class for simplifying square roots in the real domain, by walking the expression tree.

The end user interface is the function \texttt{simplify_sqrt_real().} .

INPUT:

• \texttt{ex} – a symbolic expression

EXAMPLES:

Let us consider the square root of an exact square under some assumption:

```python
sage: assume(x<1)
sage: a = sqrt(x^2-2*x+1)
```

The method \texttt{simplify_full()} is ineffective on such an expression:

```python
sage: a.simplify_full()
sqrt(x^2 - 2*x + 1)
```

and the more aggressive method \texttt{canonicalize_radical()} yields a wrong result, given that \( x < 1 \):

```python
sage: a.canonicalize_radical()  # wrong output!
x - 1
```

We construct a \texttt{SimplifySqrtReal} object \texttt{s} from the symbolic expression \texttt{a}:

```python
sage: from sage.manifolds.utilities import SimplifySqrtReal
sage: s = SimplifySqrtReal(a)
```
We use the \_\_call\_\_ method to walk the expression tree and produce a correctly simplified expression:

```
\begin{verbatim}
sage: s()
-x + 1
\end{verbatim}
```

Calling the simplifier \texttt{s} with an expression actually simplifies this expression:

```
\begin{verbatim}
sage: s(a)  # same as s() since \texttt{s} is built from \texttt{a}
-x + 1
sage: s(sqrt(x^2))
abs(x)
sage: s(sqrt(1+sqrt(x^2-2*x+1)))  # nested sqrt's
sqrt(-x + 2)
\end{verbatim}
```

Another example where both \texttt{simplify\_full()} and \texttt{canonicalize\_radical()} fail:

```
\begin{verbatim}
sage: b = sqrt((x-1)/(x-2))*sqrt(1-x)
sage: b.simplify\_full()  # does not simplify
sqrt(-x + 1)*sqrt((x - 1)/(x - 2))
sage: b.canonicalize\_radical()  # wrong output, given that \texttt{x<1}
(I*x - I)/sqrt(x - 2)
sage: SimplifySqrtReal(b)()  # OK, given that \texttt{x<1}
-(x - 1)/sqrt(-x + 2)
\end{verbatim}
```

See also:

\texttt{simplify\_sqrt\_real()} for more examples with \texttt{SimplifySqrtReal} at work.

\textbf{arithmetic}(\texttt{ex}, \texttt{operator})

This is the only method of the base class \texttt{ExpressionTreeWalker} that is reimplemented, since square roots are considered as arithmetic operations with operator = \texttt{pow} and \texttt{ex.operands}[1] = 1/2 or -1/2.

\textbf{INPUT}:

- \texttt{ex} – a symbolic expression
- \texttt{operator} – an arithmetic operator

\textbf{OUTPUT}:

- a symbolic expression, equivalent to \texttt{ex} with square roots simplified

\textbf{EXAMPLES}:

```
\begin{verbatim}
sage: from sage.manifolds.utilities import SimplifySqrtReal
sage: a = sqrt(x^2+2*x+1)
sage: s = SimplifySqrtReal(a)
sage: a.operator()<built-in function pow>
sage: s.arithmetic(a, a.operator())
abs(x + 1)
\end{verbatim}
```

```
\begin{verbatim}
sage: a = x + 1  # no square root
sage: s.arithmetic(a, a.operator())
x + 1
\end{verbatim}
```

```
\begin{verbatim}
sage: a = x + 1 + sqrt(function('f')(x)^2)
sage: s.arithmetic(a, a.operator())
x + abs(f(x)) + 1
\end{verbatim}
```
sage.manifolds.utilities.exterior_derivative(form)

Exterior derivative of a differential form.

INPUT:

- form – a differential form; this must 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):

\begin{verbatim}
sage: from sage.manifolds.utilities import exterior_derivative/nsage: 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
\end{verbatim}

An alias is xder:

\begin{verbatim}
sage: from sage.manifolds.utilities import xder/nsage: df == xder(f)
True
\end{verbatim}

Exterior derivative of a 1-form:

\begin{verbatim}
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
\end{verbatim}

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
# needs sage.plot
g = sphere()
g.all
[Graphics3d Object]
from sage.manifolds.utilities import set_axes_labels
ga = set_axes_labels(g, 'X', 'Y', 'Z', color='red')
ga.all # the 3D frame has now axes labels
sage.manifolds.utilities.simplify_abs_trig(expr)
Simplify abs(sin(...)) and abs(cos(...)) in symbolic expressions.
**EXAMPLES:**
```sage
M = Manifold(3, 'M', structure='topological')
X.<x,y,z> = M.chart(x=(0,pi), y:=(0,pi), z:=(0,pi))
X.coord_range()
x: (-oo, +oo); y: (0, pi); z: (-1/3*pi, 0)
```

Since x spans all R, no simplification of abs(sin(x)) occurs, while abs(sin(y)) and abs(sin(3*z)) are correctly simplified, given that y ∈ (0, π) and z ∈ (−π/3, 0):
```sage
from sage.manifolds.utilities import simplify_abs_trig
simplify_abs_trig(abs(sin(x)) + abs(sin(y)) + abs(sin(3*z)))
abs(sin(x)) + sin(y) + sin(-3*z)
```

Note that neither simplify_trig() nor simplify_full() works in this case:
```sage
s = abs(sin(x)) + abs(sin(y)) + abs(sin(3*z))
s.simplify_trig()
abs(4*cos(-z)^2 - 1)*abs(sin(-z)) + abs(sin(x)) + abs(sin(y))
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(1/2*y))) # shall simplify
sin(1/2*y)
```

Simplification of $\text{abs}(\cos(...))$:

```
sage: forget()
sage: M = Manifold(3, 'M', structure='topological')
sage: X.<x,y,z> = M.chart()
```

```
x: (-oo, +oo); y: (0, 1/2*pi); z: (1/4*pi, 3/4*pi)
sage: simplify_abs_trig( abs(cos(x)) + abs(cos(y)) + abs(cos(2*z)) )
abs(cos(x)) + cos(y) - cos(2*z)
```

Additional tests:

```
sage: simplify_abs_trig(abs(cos(y-pi/2)))  # shall simplify
cos(-1/2*pi + y)
sage: simplify_abs_trig(abs(cos(y+pi/2)))  # shall simplify
-cos(1/2*pi + y)
sage: simplify_abs_trig(abs(cos(y-pi)))  # shall simplify
-cos(-pi + y)
sage: simplify_abs_trig(abs(cos(2*y)))  # must not simplify
abs(cos(2*y))
sage: simplify_abs_trig(abs(cos(y/2)) * abs(sin(z)))  # shall simplify
cos(1/2*y)*sin(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 $\mathbb{R}$, as implemented in ChartFunction.

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

Then neither $x$ nor $y$ is assumed to be real:
Accordingly, `simplify_chain_generic` does not simplify `sqrt(x^2)` to `abs(x)`:

```python
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:

```python
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)
(x - 1)/(x + 1)
sage: s = cos(2*x) - 2*cos(x)^2 + 1
0
```

`sage.manifolds.utilities.simplify_chain_generic_sympy(expr)`

Apply a chain of simplifications to a sympy expression.

This is the simplification chain used in calculus involving coordinate functions on manifolds over fields different from \( \mathbb{R} \), as implemented in `ChartFunction`.

The chain is formed by the following functions, called successively:

1. `combsimp`
2. `trigsimp`
3. `expand`
4. `simplify`

**EXAMPLES:**

We consider variables that are coordinates of a chart on a complex manifold:

```python
sage: forget()  # for doctest only
sage: M = Manifold(2, 'M', structure='topological', field='complex', calc_method='sympy')
sage: X.<x,y> = M.chart()
```

Then neither \( x \) nor \( y \) is assumed to be real:

```python
sage: assumptions()
[]
```

Accordingly, `simplify_chain_generic_sympy` does not simplify `sqrt(x^2)` to `abs(x)`:

```python
sage: from sage.manifolds.utilities import simplify_chain_generic_sympy
sage: s = (sqrt(x^2))._sympy_()
sage: simplify_chain_generic_sympy(s)
sqrt(x**2)
```
This contrasts with the behavior of `simplify_chain_real_sympy()`.

Other simplifications:

```python
sage: s = ((x+y)^2 - x^2 -2*x*y - y^2)._sympy_
sage: simplify_chain_generic_sympy(s)
0
sage: s = ((x^2 - 2*x + 1) / (x^2 -1))._sympy_
sage: simplify_chain_generic_sympy(s)
(x - 1)/(x + 1)
sage: s = (cos(2*x) - 2*cos(x)^2 + 1)._sympy_
sage: simplify_chain_generic_sympy(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 `ChartFunction`.

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:

```python
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart('x:(0,1) y')
```

The following assumptions then hold:

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

and we have:

```python
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:
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.manifolds.utilities.simplify_chain_real_sympy(expr)
Apply a chain of simplifications to a sympy expression, assuming the real domain.

This is the simplification chain used in calculus involving coordinate functions on real manifolds, as implemented in `ChartFunction`.

The chain is formed by the following functions, called successively:

1. `combsimp()`
2. `trigsimp()`
3. `simplify_sqrt_real()`
4. `simplify_abs_trig()`
5. `expand()`
6. `simplify()`

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', calc_method='sympy')
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_sympy
sage: s = (sqrt(y^2))._sympy_()
sage: simplify_chain_real_sympy(s)
Abs(y)
```

Furthermore, we have:

```
sage: s = (sqrt(x^2-2*x+1))._sympy_()
sage: simplify_chain_real_sympy(s)
1 - x
```

Other simplifications:

```
sage: s = (abs(sin(pi*x)))._sympy_()
sage: simplify_chain_real_sympy(s)  # correct output since x in (0,1)
sin(pi*x)
sage: s = (cos(y)^2 + sin(y)^2)._sympy_()
sage: simplify_chain_real_sympy(s)
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 `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)
```

(continues on next page)
Sage: \sqrt{x^2}.canonicalize_radical()
-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
-1

\textbf{Simplification of nested }sqrt\textbf{'s:}

Sage: forget()  # removes the assumption \(x<0\)
Sage: simplify_sqrt_real(\sqrt{1 + \sqrt{x^2}})
\sqrt{|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, \texttt{canonicalize_radical()} fails on the last one:

Sage: (\sqrt{x^2 + \sqrt{4*x^2} + 1}).canonicalize_radical()
x - 1

\texttt{sage.manifolds.utilities.xder(form)}

\textit{Exterior derivative of a differential form.}

\textbf{INPUT:}

- \texttt{form} – a differential form; this must an instance of either
  - \texttt{DiffScalarField} for a 0-form (scalar field)
  - \texttt{DiffFormParal} for a \(p\)-form \((p \geq 1)\) on a parallelizable manifold
  - \texttt{DiffForm} for a a \(p\)-form \((p \geq 1)\) on a non-parallelizable manifold

\textbf{OUTPUT:}

- the \((p+1)\)-form that is the exterior derivative of \texttt{form}

\textbf{EXAMPLES:}

Exterior derivative of a scalar field (0-form):

\begin{verbatim}
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+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()
\mathrm{dx} + 2*y \, dy + 3*z^2 \, dz
\end{verbatim}

An alias is \texttt{xder:}

\begin{verbatim}
Sage: from sage.manifolds.utilities import xder
Sage: df == xder(f)
True
\end{verbatim}

Exterior derivative of a 1-form:
```python
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()
da = 0
```

See also:

`sage.manifolds.differentiable.diff_form.DiffFormParal.exterior_derivative`
or `sage.manifolds.differentiable.diff_form.DiffForm.exterior_derivative` for more examples.
A catalog of manifolds to rapidly create various simple manifolds.

The current entries to the catalog are obtained by typing `manifolds.<tab>`, where `<tab>` indicates pressing the Tab key. They are:

- `EuclideanSpace`: Euclidean space
- `RealLine`: real line
- `OpenInterval`: open interval on the real line
- `Sphere`: sphere embedded in Euclidean space
- `Torus()`: torus embedded in Euclidean space
- `Minkowski()`: 4-dimensional Minkowski space
- `Kerr()`: Kerr spacetime
- `RealProjectiveSpace()`: n-dimensional real projective space

AUTHORS:

- Florentin Jaffredo (2018): initial version
- Trevor K. Karn (2022): projective space

```python
sage.manifolds.catalog.Kerr(m=1, a=0, coordinates='BL', names=None)
```

Generate a Kerr spacetime.

A Kerr spacetime is a 4 dimensional manifold describing a rotating black hole. Two coordinate systems are implemented: Boyer-Lindquist and Kerr (3+1 version).

The shortcut operator `.<>` can be used to specify the coordinates.

INPUT:

- `m` – (default: 1) mass of the black hole in natural units (c = 1, G = 1)
- `a` – (default: 0) angular momentum in natural units; if set to 0, the resulting spacetime corresponds to a Schwarzschild black hole
- `coordinates` – (default: "BL") either "BL" for Boyer-Lindquist coordinates or "Kerr" for Kerr coordinates (3+1 version)
- `names` – (default: None) name of the coordinates, automatically set by the shortcut operator

OUTPUT:

- Lorentzian manifold

EXAMPLES:
sage: m, a = var('m, a')
sage: K = manifolds.Kerr(m, a)
sage: K
4-dimensional Lorentzian manifold M
sage: K.atlas()
[Chart (M, (t, r, th, ph))]
sage: K.metric().display()
g = (2*m*r/(a^2*cos(th)^2 + r^2) - 1) dt\otimes dt + 2*a*m*r*sin(th)^2/(a^2*cos(th)^2 + r^2) dt\otimes dph + (a^2*cos(th)^2 + r^2)/(a^2 - 2*m*r + r^2) dr\otimes dr + (a^2*cos(th)^2 + r^2) dth\otimes dth + 2*a*m*r*sin(th)^2/(a^2*cos(th)^2 + r^2) dph\otimes dt + (2*a^2*m*r*sin(th)^2/(a^2*cos(th)^2 + r^2) + a^2 + r^2)*sin(th)^2 dph\otimes dph

sage: K.<t, r, th, ph> = manifolds.Kerr()
sage: K
4-dimensional Lorentzian manifold M
sage: K.metric().display()
g = (2/r - 1) dt\otimes dt + r^2/(r^2 - 2*r) dr\otimes dr + r^2 dth\otimes dth + r^2*sin(th)^2 dph\otimes dph
sage: K.default_chart().coord_range()
t: (-oo, +oo); r: (0, +oo); th: (0, pi); ph: [-pi, pi] (periodic)
sage: m, a = var('m, a')
sage: K.<t, r, th, ph> = manifolds.Kerr(m, a, coordinates="Kerr")
sage: K
4-dimensional Lorentzian manifold M
sage: K.atlas()
[Chart (M, (t, r, th, ph))]
sage: K.metric().display()
g = (2*m*r/(a^2*cos(th)^2 + r^2) - 1) dt\otimes dt + 2*m*r/(a^2*cos(th)^2 + r^2) dt\otimes dr - 2*a*m*r*sin(th)^2/(a^2*cos(th)^2 + r^2) dt\otimes dph + 2*m*r/(a^2*cos(th)^2 + r^2) dr\otimes dt + (2*m*r/(a^2*cos(th)^2 + r^2) + 1) dr\otimes dr - a^2*(2*m*r/(a^2*cos(th)^2 + r^2) + 1)*sin(th)^2 dr\otimes dph + (a^2*cos(th)^2 + r^2) dth\otimes dth + 2*a*m*r*sin(th)^2/(a^2*cos(th)^2 + r^2) dph\otimes dt + (a^2*m*r*sin(th)^2/(a^2*cos(th)^2 + r^2) + a^2 + r^2)*sin(th)^2 dph\otimes dph
sage: K.default_chart().coord_range()
t: (-oo, +oo); r: (0, +oo); th: (0, pi); ph: [-pi, pi] (periodic)
sage.manifolds.catalog.Minkowski (positive_spacelike=True, names=None)
Generate a Minkowski space of dimension 4.
By default the signature is set to (- + + +), but can be changed to (+ - - -) by setting the optional argument positive_spacelike to False. The shortcut operator .<,> can be used to specify the coordinates.

INPUT:

- positive_spacelike – (default: True) if False, then the spacelike vectors yield a negative sign (i.e., the signature is (+ - - -))
- names – (default: None) name of the coordinates, automatically set by the shortcut operator

OUTPUT:

- Lorentzian manifold of dimension 4 with (flat) Minkowskian metric
EXAMPLES:

```python
sage: M.<t, x, y, z> = manifolds.Minkowski()
sage: M.metric()[:]
[-1 0 0 0]
[ 0 1 0 0]
[ 0 0 1 0]
[ 0 0 0 1]
sage: M.<t, x, y, z> = manifolds.Minkowski(False)
sage: M.metric()[:]
[ 1 0 0 0]
[ 0 -1 0 0]
[ 0 0 -1 0]
[ 0 0 0 -1]
```

`sage.manifolds.catalog.RealProjectiveSpace(dim=2)`

Generate projective space of dimension dim over the reals.

This is the topological space of lines through the origin in $\mathbb{R}^{d+1}$. The standard atlas consists of $d + 2$ charts, which sends the set $U_i = \{x_1, x_2, \ldots, x_{d+1} \mid x_i \neq 0\}$ to $k^d$ by dividing by $x_i$ and omitting the $i$-th coordinate $x_i / x_i = 1$.

**INPUT:**

- dim – (default: 2) the dimension of projective space

**OUTPUT:**

- $\mathbb{P}^d$ – the projective space $\mathbb{R}P^d$ where $d = \text{dim}$.

**EXAMPLES:**

```python
sage: RP2 = manifolds.RealProjectiveSpace(); RP2
2-dimensional topological manifold RP2
sage: latex(RP2)
\mathbb{RP}^2
sage: C0, C1, C2 = RP2.top_charts()
sage: p = RP2.point((2,0), chart = C0)
sage: q = RP2.point((0,3), chart = C0)
sage: p in C0.domain()
True
sage: p in C1.domain()
True
sage: C1(p)
(1/2, 0)
sage: p in C2.domain()
False
sage: q in C0.domain()
True
sage: q in C1.domain()
False
sage: q in C2.domain()
True
sage: C2(q)
(1/3, 0)
sage: r = RP2.point((2,3))
sage: r in C0.domain() and r in C1.domain() and r in C2.domain()
```

(continues on next page)
True
sage: C0(r) (2, 3)
sage: C1(r) (1/2, 3/2)
sage: C2(r) (1/3, 2/3)

sage: p = RP2.point((2,3), chart = C1)
sage: p in C0.domain() and p in C1.domain() and p in C2.domain()
True
sage: C0(p) (1/2, 3/2)
sage: C2(p) (2/3, 1/3)

sage: RP1 = manifolds.RealProjectiveSpace(1); RP1
1-dimensional topological manifold RP1
sage: C0, C1 = RP1.top_charts()
sage: p, q = RP1.point((2,)), RP1.point((0,))
sage: p in C0.domain()
True
sage: q in C0.domain()
True
sage: q in C1.domain()
False
sage: C1(p) (1/2,)

sage: p, q = RP1.point((3,), chart = C1), RP1.point((0,), chart = C1)
sage: p in C0.domain()
True
sage: q in C0.domain()
False
sage: C0(p) (1/3,)

\begin{verbatim}
sage.manifolds.catalog.Torus\(R=2, r=1, names=None\)
\end{verbatim}

Generate a 2-dimensional torus embedded in Euclidean space.

The shortcut operator \(<,>\) can be used to specify the coordinates.

**INPUT:**

- \(R\) – (default: 2) distance form the center to the center of the tube
- \(r\) – (default: 1) radius of the tube
- \(names\) – (default: None) name of the coordinates, automatically set by the shortcut operator

**OUTPUT:**

- Riemannian manifold

**EXAMPLES:**
Manifolds, Release 10.3

```python
sage: T.<theta, phi> = manifolds.Torus(3, 1)
sage: T
2-dimensional Riemannian submanifold T embedded in the Euclidean space E^3
sage: T.atlas()
[Chart (T, (theta, phi))]
sage: T.embedding().display()
T → E^3
  (theta, phi) ↦ (X, Y, Z) = ((cos(theta) + 3)*cos(phi),
    (cos(theta) + 3)*sin(phi),
    sin(theta))
sage: T.metric().display()
gamma = dtheta⊗dtheta + (cos(theta)^2 + 6*cos(theta) + 9) dphi⊗dphi
```
INDICES AND TABLES

- Index
- Module Index
- Search Page
sage.manifolds.calculus_method, 162
sage.manifolds.catalog, 1011
sage.manifolds.chart, 85
sage.manifolds.chart_func, 130
sage.manifolds.continuous_map, 216
sage.manifolds.continuous_map_image, 243
sage.manifolds.differentiable.affine_connection, 765
sage.manifolds.differentiable.automorphismfield, 604
sage.manifolds.differentiable.automorphismfield_group, 598
sage.manifolds.differentiable.bundle_connection, 814
sage.manifolds.differentiable.characteristic_cohomology_class, 825
sage.manifolds.differentiable.chart, 388
sage.manifolds.differentiable.curve, 465
sage.manifolds.differentiable.de_rham_cohomology, 740
sage.manifolds.differentiable.degenerate, 959
sage.manifolds.differentiable.degenerate_submanifold, 965
sage.manifolds.differentiable.diff_form, 702
sage.manifolds.differentiable.diff_form_module, 690
sage.manifolds.differentiable.diff_map, 453
sage.manifolds.differentiable.differentiable_submanifold, 788
sage.manifolds.differentiable.examples.euclidean, 855
sage.manifolds.differentiable.examples.real_line, 403
sage.manifolds.differentiable.examples.sphere, 881
sage.manifolds.differentiable.examples.symplectic_space, 994
sage.manifolds.differentiable.integrated_curve, 481
sage.manifolds.differentiable.levi_civita_connection, 928
sage.manifolds.differentiable.manifold, 335
sage.manifolds.differentiable.manifold_homset, 440
sage.manifolds.differentiable.metric, 894
sage.manifolds.differentiable.mixed_form, 726
sage.manifolds.differentiable.mixed_form_algebra, 721
sage.manifolds.differentiable.multi_vector_module, 744
sage.manifolds.differentiable.multi_vectorfield, 750
sage.manifolds.differentiable.poisson_tensor, 983
sage.manifolds.differentiable.pseudo_riemannian, 845
sage.manifolds.differentiable.pseudo_riemannian_submanifold, 935
sage.manifolds.differentiable.scalarfield, 419
sage.manifolds.differentiable.scalarfield_algebra, 414
sage.manifolds.differentiable.symplectic_form, 987
sage.manifolds.differentiable.tangent_space, 514
sage.manifolds.differentiable.tangent_vector, 518
sage.manifolds.differentiable.tensorfield, 621
sage.manifolds.differentiable.tensorfield_module, 615
sage.manifolds.differentiable.tensorfield_paral, 667
sage.manifolds.differentiable.vec-
Non-alphabetical

__call__() (sage.manifolds.chart_func.ChartFunction method), 134

A

adapted_chart() (sage.manifolds.topological_submanifold.TopologicalSubmanifold method), 245
adapted_frame() (sage.manifolds.differentiable.degenerate_submanifold.DegenerateSubmanifold method), 969
add_coef() (sage.manifolds.differentiable.affine_connection.AffineConnection method), 770
add_comp() (sage.manifolds.differentiable.automorphismfield.AutomorphismField method), 606
add_comp() (sage.manifolds.differentiable.tensorfield_paral.TensorFieldParal method), 676
add_comp() (sage.manifolds.differentiable.tensorfield.TensorField method), 627
add_comp() (sage.manifolds.differentiable.tensorfield.TensorFieldParal method), 677
add_comp() (sage.manifolds.section.Section method), 305
add_comp() (sage.manifolds.section.TrivialSection method), 321
add_comp_by_continuation() (sage.manifolds.differentiable.mixed_form.MixedForm method), 729
add_comp_by_continuation() (sage.manifolds.differentiable.tensorfield.TensorField method), 628
add_comp_by_continuation() (sage.manifolds.section.Section method), 306
add_connection_form() (sage.manifolds.differentiable.bundle_connection.BundleConnection method), 817
add_coord() (sage.manifolds.point.ManifoldPoint method), 72
add_coordinates() (sage.manifolds.point.ManifoldPoint method), 73
add_expr() (sage.manifolds.continuous_map.ContinuousMap method), 221
add_expr() (sage.manifolds.scalarfield.ScalarField method), 190
add_expr_by_continuation() (sage.manifolds.scalarfield.ScalarField method), 191
add_expr_from_subdomain() (sage.manifolds.differentiable.tensorfield.TensorField method), 629
add_expr_from_subdomain() (sage.manifolds.section.Section method), 307
add_expression() (sage.manifolds.continuous_map.ContinuousMap method), 223
add_restrictions() (sage.manifolds.chart.Chart method), 89
add_restrictions() (sage.manifolds.chart.RealChart method), 110
additive_sequence() (in module sage.manifolds.differentiable.characteristic_cohomology_class), 841
affine_connection() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 344
AffineConnection (class in sage.manifolds.differentiable.affine_connection), 765
Algorithm_generic (class in sage.manifolds.differentiable.characteristic_cohomology_class), 829
along() (sage.manifolds.differentiable.tensorfield_paral.TensorFieldParal method), 677
along() (sage.manifolds.differentiable.tensorfield.TensorField method), 631
along() (sage.manifolds.differentiable.vectorframe.TensorFrame method), 590
alternating_contravariant_tensor() (sage.manifolds.differentiable.tensorfield_module.TensorFieldModule method), 543
alternating_form() (sage.manifolds.differentiable.tensorfield_module.TensorFieldModule method), 544
ambient() (sage.manifolds.subset.ManifoldSubset method), 39
ambient() (sage.manifolds.topological_submanifold.TopologicalSubmanifold method), 246
ambient_domain() (sage.manifolds.differentiable.vector_bundle.TensorBundle method), 797
ambient_domain() (sage.manifolds.differentiable.tensorfield_module.TensorFieldModule method), 544
```
table.vectorfield_module.VectorFieldFreeModule(), 534
ambient_domain() (sage.manifolds.differentiable.vectorfield_module.VectorFieldModule, 544)
ambient_domain() (sage.manifolds.differentiable.vectorfield_module.VectorFieldModule, 591)
ambient_extrinsic_curvature() (sage.manifolds.differentiable.pseudo_riemannian_submanifold.PseudoRiemannianSubmanifold, 940)
ambient_first_fundamental_form() (sage.manifolds.differentiable.pseudo_riemannian_submanifold.PseudoRiemannianSubmanifold, 941)
ambient_induced_metric() (sage.manifolds.differentiable.pseudo_riemannian_submanifold.PseudoRiemannianSubmanifold, 941)
ambient_metric() (sage.manifolds.differentiable.degenerate_submanifold.DegenerateSubmanifold, 969)
ambient_metric() (sage.manifolds.differentiable.pseudo_riemannian_submanifold.PseudoRiemannianSubmanifold, 942)
ambient_second_fundamental_form() (sage.manifolds.differentiable.pseudo_riemannian_submanifold.PseudoRiemannianSubmanifold, 942)
antisymmetrize() (sage.manifolds.differentiable.tensorfield.TensorField, 632)
apply_map() (sage.manifolds.differentiable.tensorfield.TensorField, 633)
arccos() (sage.manifolds.chart_func.ChartFunction, 135)
arccos() (sage.manifolds.scalarfield.ScalarField, 192)
arccos() (sage.manifolds.scalarfield.ScalarField, 192)
arccosh() (sage.manifolds.chart_func.ChartFunction, 136)
arccosh() (sage.manifolds.scalarfield.ScalarField, 192)
arccos() (sage.manifolds.scalarfield.ScalarField, 193)
arcsinh() (sage.manifolds.chart_func.ChartFunction, 137)
arcsinh() (sage.manifolds.scalarfield.ScalarField, 194)
arctan() (sage.manifolds.chart_func.ChartFunction, 137)
arctan() (sage.manifolds.scalarfield.ScalarField, 194)
arctanh() (sage.manifolds.chart_func.ChartFunction, 138)
arctanh() (sage.manifolds.scalarfield.ScalarField, 195)
arithmetic() (sage.manifolds.utilities.SimplifySqrtReal, 1001)
as_subset() (sage.manifolds.topological_submanifold.TopologicalSubmanifold, 247)
at() (sage.manifolds.differentiable.automorphismfield.AutomorphismFieldParal, 612)
at() (sage.manifolds.differentiable.tensorfield_paral.TensorFieldParal, 678)
at() (sage.manifolds.differentiable.vectorframe.TensorFrame, 634)
at() (sage.manifolds.differentiable.vectorframe.CoFrame, 583)
at() (sage.manifolds.differentiable.vectorframe.VectorFrame, 592)
at() (sage.manifolds.local_frame.LocalFrame, 282)
at() (sage.manifolds.local_frame.LocalCoFrame, 286)
at() (sage.manifolds.section.Section, 308)
at() (sage.manifolds.section.TrivialSection, 322)
atlas() (sage.manifolds.differentiable.vector_bundle.TensorBundle, 797)
atlas() (sage.manifolds.manifold.TopologicalManifold, 14)
atlas() (sage.manifolds.vector_bundle.TopologicalVectorBundle, 255)
automorphism() (sage.manifolds.differentiable.vectorfield_module.vector_bundle.TensorFieldModule, 544)
automorphism() (sage.manifolds.trivialization.TransitionMap, 272)
automorphism_field() (sage.manifolds.differentiable.manifold.DifferentiableManifold, 345)
automorphism_field_group() (sage.manifolds.differentiable.manifold.DifferentiableManifold, 346)
AutomorphismField (class in sage.manifolds.differentiable.automorphism_field, 604)
AutomorphismFieldGroup (class in sage.manifolds.differentiable.automorphism_field_group, 598)
AutomorphismFieldParal (class in sage.manifolds.differentiable.automorphism_field, 611)
AutomorphismFieldParalGroup (class in sage.manifolds.differentiable.automorphism_field_group, 601)
base_field() (sage.manifolds.manifold.TopologicalManifold, 15)
```
Index
display_exp() (sage.manifolds.differentiable.mixed_form.MixedForm method), 733
display_expansion() (sage.manifolds.differentiable.mixed_form.MixedForm method), 733
dist() (sage.manifolds.differentiable.examples.euclidean.EuclideanSpace method), 879
dist() (sage.manifolds.differentiable.examples.sphere.Sphere method), 886
div() (in module sage.manifolds_operators), 891
div() (sage.manifolds.differentiable.tensorfield.TensorField method), 647
divergence() (sage.manifolds.differentiable.tensorfield.TensorField method), 648

domain() (sage.manifolds.chart.Chart method), 91
domain() (sage.manifolds.differentiable.affine_connection.AffineConnection method), 778
domain() (sage.manifolds.differentiable.tensorfield.TensorField method), 650
domain() (sage.manifolds.differentiable.vectorfield_module.VectorFieldFreeModule method), 536
domain() (sage.manifolds.differentiable.vectorfield_module.VectorFieldModule method), 545
domain() (sage.manifolds.differentiable.vectorframe.VectorFrame method), 594
domain() (sage.manifolds.local_frame.LocalFrame method), 288
domain() (sage.manifolds.scalarfield.ScalarField method), 202
domain() (sage.manifolds.section_module.SectionFreeModule method), 297
domain() (sage.manifolds.section_module.SectionModule method), 300
domain() (sage.manifolds.section.Section method), 316
domain() (sage.manifolds.trivialization.Trivialization method), 277
dot() (sage.manifolds.differentiable.vectorfield.VectorField method), 559
dot_product() (sage.manifolds.differentiable.vectorfield.VectorField method), 560
down() (sage.manifolds.differentiable.tensorfield.TensorField method), 650
dual() (sage.manifolds.differentiable.vectorfield_module.VectorFieldModule method), 546
dual_exterior_power() (sage.manifolds.differentiable.vectorfield_module.VectorFieldModule method), 536
dual_exterior_power() (sage.manifolds.differentiable.vectorfield_module.VectorFieldFreeModule method), 546

dimension method), 323

deformation() (sage.manifolds.differentiable.tensorfield.TensorField method), 643
dominion() (sage.manifolds.differentiable.tensorfield.TensorField method), 644
dual() (sage.manifolds.differentiable.tensorfield.TensorField method), 647
div() (sage.manifolds.differentiable.tensorfield.TensorField method), 648

domain() (sage.manifolds.chart.Chart method), 91
domain() (sage.manifolds.differentiable.affine_connection.AffineConnection method), 778
domain() (sage.manifolds.differentiable.tensorfield.TensorField method), 650
domain() (sage.manifolds.differentiable.vectorfield_module.VectorFieldFreeModule method), 536
domain() (sage.manifolds.differentiable.vectorfield_module.VectorFieldModule method), 545
domain() (sage.manifolds.differentiable.vectorframe.VectorFrame method), 594
domain() (sage.manifolds.local_frame.LocalFrame method), 288
domain() (sage.manifolds.scalarfield.ScalarField method), 202
domain() (sage.manifolds.section_module.SectionFreeModule method), 297
domain() (sage.manifolds.section_module.SectionModule method), 300
domain() (sage.manifolds.section.Section method), 316
domain() (sage.manifolds.trivialization.Trivialization method), 277
dot() (sage.manifolds.differentiable.vectorfield.VectorField method), 559
dot_product() (sage.manifolds.differentiable.vectorfield.VectorField method), 560
down() (sage.manifolds.differentiable.tensorfield.TensorField method), 650
dual() (sage.manifolds.differentiable.vectorfield_module.VectorFieldModule method), 546
dual_exterior_power() (sage.manifolds.differentiable.vectorfield_module.VectorFieldModule method), 536
dual_exterior_power() (sage.manifolds.differentiable.vectorfield_module.VectorFieldFreeModule method), 546

dimension method), 323

deformation() (sage.manifolds.differentiable.tensorfield.TensorField method), 643
dominion() (sage.manifolds.differentiable.tensorfield.TensorField method), 644
dual() (sage.manifolds.differentiable.tensorfield.TensorField method), 647
div() (sage.manifolds.differentiable.tensorfield.TensorField method), 648

domain() (sage.manifolds.chart.Chart method), 91
domain() (sage.manifolds.differentiable.affine_connection.AffineConnection method), 778
domain() (sage.manifolds.differentiable.tensorfield.TensorField method), 650
domain() (sage.manifolds.differentiable.vectorfield_module.VectorFieldFreeModule method), 536
domain() (sage.manifolds.differentiable.vectorfield_module.VectorFieldModule method), 545
domain() (sage.manifolds.differentiable.vectorframe.VectorFrame method), 594
domain() (sage.manifolds.local_frame.LocalFrame method), 288
domain() (sage.manifolds.scalarfield.ScalarField method), 202
domain() (sage.manifolds.section_module.SectionFreeModule method), 297
domain() (sage.manifolds.section_module.SectionModule method), 300
domain() (sage.manifolds.section.Section method), 316
domain() (sage.manifolds.trivialization.Trivialization method), 277
dot() (sage.manifolds.differentiable.vectorfield.VectorField method), 559
dot_product() (sage.manifolds.differentiable.vectorfield.VectorField method), 560
down() (sage.manifolds.differentiable.tensorfield.TensorField method), 650
dual() (sage.manifolds.differentiable.vectorfield_module.VectorFieldModule method), 546
dual_exterior_power() (sage.manifolds.differentiable.vectorfield_module.VectorFieldModule method), 536
dual_exterior_power() (sage.manifolds.differentiable.vectorfield_module.VectorFieldFreeModule method), 546

Element (sage.manifolds.chart_func.ChartFunctionRing attribute), 156
Element (sage.manifolds.differentiable.automorphism_field_group.AutomorphismFieldGroup attribute), 600
Element (sage.manifolds.differentiable.automorphism_field_group.AutomorphismFieldParalGroup attribute), 604
Element (sage.manifolds.differentiable.characteristic_cohomology_class.CharacteristicCohomologyClassRing attribute), 832
Element (sage.manifolds.differentiable.de_rham_cohomology.DeRhamCohomologyRing attribute), 743
Element (sage.manifolds.differentiable.diff_form_module.DiffFormFreeModule attribute), 693
Element (sage.manifolds.differentiable.diff_form_module.DiffFormModule attribute), 697
Element (sage.manifolds.differentiable.manifold_homset.DifferentiableCurveSet attribute), 443
Element (sage.manifolds.differentiable.manifold_homset.DifferentiableManifoldHomset attribute), 445
Element (sage.manifolds.differentiable.manifold_homset.IntegratedAutoparallelCurveSet attribute), 447
Element (sage.manifolds.differentiable.manifold_homset.IntegratedCurveSet attribute), 450
Element (sage.manifolds.differentiable.manifold_homset.IntegratedGeodesicSet attribute), 453
Element (sage.manifolds.differentiable.mixed_form_algebra.MixedFormAlgebra attribute), 723
Element (sage.manifolds.differentiable.mixed_form_algebra.VectorGroup attribute), 747
Element (sage.manifolds.differentiable.mixed_form_algebra.MixedVectorGroup attribute), 749
Element (sage.manifolds.differentiable.scalarfield_algebra.DiffScalarFieldAlgebra attribute), 419
Element (sage.manifolds.differentiable.tangent_space.TangentSpace attribute), 517
Element (sage.manifolds.differentiable.tensorfield_module.TensorFieldFreeModule attribute), 618
Element (sage.manifolds.differentiable.tensorfield_module.TensorFieldModule attribute), 621
Element (sage.manifolds.differentiable.tensorfield_module.TensorFieldModule attribute), 534
Element (sage.manifolds.differentiable.tensorfield_module.TensorFieldModule attribute), 543
Element (sage.manifolds.manifold_homset.TopologicalManifoldHomset attribute), 215
Element (sage.manifolds.scalarfield_algebra.ScalarFieldAlgebra attribute), 171
Element (sage.manifolds.section_module.Section-
**Index**

**Manifolds, Release 10.3**

- `FreeModule method`, 537
- `general_linear_group()` (sage.manifolds.differentiable.vectorfield_module.VectorFieldModule method), 547
- `get()` (sage.manifolds.differentiable.characteristic_cohomology_class.Algorithm_generic method), 829
- `get()` (sage.manifolds.differentiable.characteristic_cohomology_class.EulerAlgorithm method), 836
- `get()` (sage.manifolds.differentiable.characteristic_cohomology_class.PontryaginEulerAlgorithm method), 840
- `get_chart()` (sage.manifolds.manifold.TopologicalManifold method), 23
- `get_form()` (sage.manifolds.differentiable.characteristic_cohomology_class.CharacteristicCohomologyClassRingElement method), 829
- `get_gen_pow()` (sage.manifolds.differentiable.characteristic_cohomology_class.Algorithm_generic method), 833
- `get_gen_pow()` (sage.manifolds.differentiable.characteristic_cohomology_class.PontryaginEulerAlgorithm method), 840
- `get_local()` (sage.manifolds.differentiable.characteristic_cohomology_class.Algorithm_generic method), 830
- `get_local()` (sage.manifolds.differentiable.characteristic_cohomology_class.ChernAlgorithm method), 835
- `get_local()` (sage.manifolds.differentiable.characteristic_cohomology_class.EulerAlgorithm method), 837
- `get_local()` (sage.manifolds.differentiable.characteristic_cohomology_class.PontryaginAlgorithm method), 839
- `get_local()` (sage.manifolds.differentiable.characteristic_cohomology_class.PontryaginEulerAlgorithm method), 840
- `get_subset()` (sage.manifolds.subset.ManifoldSubset method), 49
- `grad()` (in module sage.manifolds.operators), 892
- `gradient()` (sage.manifolds.differentiable.scalarfield.DiffScalarField method), 435
- `gradt()` (sage.manifolds.differentiable.pseudo_riemannian_submanifold.PseudoRiemannianSubmanifold method), 947

**H**

- `hamiltonian_vector_field()` (sage.manifolds.differentiable.poisson_tensor.PoissonTensorField method), 984
- `hamiltonian_vector_field()` (sage.manifolds.differentiable.symplectic_form.SymplecticForm method), 988
- `has_defined_points()` (sage.manifolds.subset.ManifoldSubset method), 49
- `has_orientation()` (sage.manifolds.manifold.TopologicalManifold method), 23
- `has_orientation()` (sage.manifolds.vector_bundle.TriplewiseVectorBundle method), 258
- `hodge_dual()` (sage.manifolds.differentiable.diff_form.DiffForm method), 708
- `hodge_dual()` (sage.manifolds.differentiable.scalarfield.DiffScalarField method), 436
- `hodge_star()` (sage.manifolds.differentiable.metric.PseudoRiemannianMetric method), 911
- `hodge_star()` (sage.manifolds.differentiable.symplectic_form.SymplecticForm method), 989
- `homeomorphism()` (sage.manifolds.manifold.TopologicalManifold method), 24
- `homology()` (sage.manifolds.differentiable.mixed_form_algebra.MixedFormAlgebra method), 724
- `homset` (sage.manifolds.structure.DegenerateStructure attribute), 66
- `homset` (sage.manifolds.structure.DifferentialStructure attribute), 67
- `homset` (sage.manifolds.structure.LorentzianStructure attribute), 67
- `homset` (sage.manifolds.structure.PseudoRiemannianStructure attribute), 68
- `homset` (sage.manifolds.structure.RealDifferentialStructure attribute), 68
- `homset` (sage.manifolds.structure.RealTopologicalStructure attribute), 69
- `homset` (sage.manifolds.structure.RiemannianStructure attribute), 69
- `homset` (sage.manifolds.structure.TopologicalStructure attribute), 69
- `identity_map()` (sage.manifolds.differentiable.vectorfield_module.VectorFieldModule method), 548
- `identity_map()` (sage.manifolds.manifold.TopologicalManifold method), 25
- `image()` (sage.manifolds.continuous_map.ContinuousMap method), 234
- `ImageManifoldSubset` (class in sage.manifolds.continuous_map.ContinuousMap method), 243
- `immersion()` (sage.manifolds.topological_submanifold.TriplewiseTopologicalSubmanifold method), 248
- `index_generator()` (sage.manifolds.manifold.TopologicalManifold method), 26
- `induced_metric()` (sage.manifolds.differentiable.degenerate_submanifold.DegenerateSubmanifold method), 972
is_tangent() (sage.manifolds.differentiable.degenerate_submanifold.DegenerateSubmanifold method), 972

is_trivial_one() (sage.manifolds.chart_func.ChartFunction method), 148

is_trivial_one() (sage.manifolds.scalarfield.ScalarField method), 204

is_trivial_zero() (sage.manifolds.calculus_method.CalculusMethod method), 163

is_trivial_zero() (sage.manifolds.chart_func.ChartFunction method), 148

is_trivial_zero() (sage.manifolds.scalarfield.ScalarField method), 205

is_unit() (sage.manifolds.chart_func.ChartFunction method), 149

is_unit() (sage.manifolds.scalarfield.ScalarField method), 206

jacobian() (sage.manifolds.chart_func.MultiCoordFunction method), 159

jacobian() (sage.manifolds.differentiable.chart.DiffCoordChange method), 397

jacobian_det() (sage.manifolds.chart_func.MultiCoordFunction method), 160

jacobian() (sage.manifolds.differentiable.chart.DiffCoordChange method), 397

jacobian_matrix() (sage.manifolds.differentiable.diff_map.DiffMap method), 461

Kerr() (in module sage.manifolds.catalog), 1011

laplacian() (in module sage.manifolds.operators), 893

laplacian() (sage.manifolds.differentiable.scalarfield.DiffScalarField method), 437

laplacian() (sage.manifolds.differentiable.tensorfield.TensorField method), 652

lapse() (sage.manifolds.differentiable.pseudo_riemannian_submanifold.PseudoRiemannianSubmanifold method), 948

LeviCivitaConnection (class in sage.manifolds.differentiable.levi_civita_connection), 928

lie_der() (sage.manifolds.differentiable.scalarfield.DiffScalarField method), 438

lie_der() (sage.manifolds.differentiable.tensorfield.paral.TensorFieldParal method), 683

lie_der() (sage.manifolds.differentiable.tensorfield.TensorField method), 653

lie_derivative() (sage.manifolds.differentiable.scalarfield.DiffScalarField method), 438

lie_derivative() (sage.manifolds.differentiable.tensorfield_paral.TensorFieldParal method), 685

lie_derivative() (sage.manifolds.differentiable.tensorfield.TensorField method), 655

lift() (sage.manifolds.differentiable.de_rham_cohomology.DeRhamCohomologyClass method), 741

lift() (sage.manifolds.subset.ManifoldSubset method), 52

lift_from_homology() (sage.manifolds.differentiable.degenerate_degenerate_submanifold.DegenerateSubmanifold method), 973

d_list_of_screens() (sage.manifolds.differentiable.degenerate_submanifold.DegenerateSubmanifold method), 973

d_list_of_subsets() (sage.manifolds.subset.ManifoldSubset method), 53

local_frame() (sage.manifolds.differentiable.vector_bundle.TopologicalVectorBundle method), 260

LocalCoFrame (class in sage.manifolds.local_frame), 281

LocalFrame (class in sage.manifolds.local_frame), 284

log() (sage.manifolds.chart_func.ChartFunction method), 149

log() (sage.manifolds.scalarfield.ScalarField method), 206

lorentzian_metric() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 362

LorentzianStructure (class in sage.manifolds.structure), 67

lower_bound() (sage.manifolds.differentiable.examples.real_line.OpenInterval method), 409

Manifold() (in module sage.manifolds.manifold), 8

manifold() (sage.manifolds.chart.Chart method), 92

manifold() (sage.manifolds.subset.ManifoldSubset method), 53

ManifoldObjectFiniteFamily (class in sage.manifolds.family), 326

ManifoldPoint (class in sage.manifolds.point), 71

ManifoldSubset (class in sage.manifolds.subset), 38

ManifoldSubsetClosure (class in sage.manifolds.subsets.closure), 328

1032 Index
ManifoldSubsetFiniteFamily (class in sage.manifolds.family), 327
ManifoldSubsetPullback (class in sage.manifolds.subsets.pullback), 329
matrix() (sage.manifolds.trivialization.TransitionMap method), 274
mean_curvature() (sage.manifolds.differentiable.degenerate_submanifold.DegenerateSubmanifold method), 973
mean_curvature() (sage.manifolds.differentiable.pseudo_riemannian_submanifold.PseudoRiemannianSubmanifold method), 949
metric() (sage.manifolds.differentiable.degenerate.DegenerateManifold method), 961
metric() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 363
metric() (sage.manifolds.differentiable.pseudo_riemannian_submanifold.PseudoRiemannianSubmanifold method), 949
metric() (sage.manifolds.differentiable.pseudo_riemannian.PseudoRiemannianManifold method), 851
metric() (sage.manifolds.differentiable.vector_field_module.VectorFieldFreeModule method), 538
metric() (sage.manifolds.differentiable.vector_field_module.VectorFieldModule method), 549
minimal_triangulation() (sage.manifolds.differentiable_examples.sphere.Sphere method), 886
Minkowski() (in module sage.manifolds.catalog), 1012
mixed_form() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 363
mixed_form_algebra() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 364
mixed_projection() (sage.manifolds.differentiable.pseudo_riemannian_submanifold.PseudoRiemannianSubmanifold method), 950
MixedForm (class in sage.manifolds.differentiable.mixed_form), 726
MixedFormAlgebra (class in sage.manifolds.differentiable.mixed_form_algebra), 721
module
sage.manifolds.calculus_method, 162
sage.manifolds.catalog, 1011
sage.manifolds.chart, 85
sage.manifolds.chart_func, 130
sage.manifolds.continuous_map, 216
sage.manifolds.continuous_map_image, 243
sage.manifolds.differentiable.affine_connection, 765
sage.manifolds.differentiable.automorphismfield, 604
sage.manifolds.differentiable.automorphismfield_group, 598
sage.manifolds.differentiable.bundle_connection, 814
sage.manifolds.differentiable.characteristic_cohomology_class, 825
sage.manifolds.differentiable.chart, 388
sage.manifolds.differentiable.curve, 465
sage.manifolds.differentiable.de_rham_cohomology, 740
sage.manifolds.differentiable.degenerate, 959
sage.manifolds.differentiable.degenerate_submanifold, 965
sage.manifolds.differentiable.diff_form, 702
sage.manifolds.differentiable.diff_form_module, 690
sage.manifolds.differentiable.diff_map, 453
sage.manifolds.differentiable.degenerate.submanifold, 788
sage.manifolds.differentiable.examples.euclidean, 855
sage.manifolds.differentiable.examples.real_line, 403
sage.manifolds.differentiable.examples.sphere, 881
sage.manifolds.differentiable.examples.symplectic_space, 994
sage.manifolds.differentiable.integrated_curve, 481
sage.manifolds.differentiable.levi_civita_connection, 928
sage.manifolds.differentiable.manifold, 335
sage.manifolds.differentiable.manifold_homset, 440
sage.manifolds.differentiable.metric, 894
sage.manifolds.differentiable.mixed_form, 726
sage.manifolds.differentiable.mixed_form_algebra, 721
sage.manifolds.differentiable.multivector_module, 744
sage.manifolds.differentiable.multivectorfield, 750
sage.manifolds.differentiable.poisson_tensor, 983
Index 1035

one() (sage.manifolds.chart_func.ChartFunctionRing method), 156
one() (sage.manifolds.differentiable.automorphism_field_group.AutomorphismFieldGroup method), 600
one() (sage.manifolds.differentiable.de_rham_cohomology.DeRhamCohomologyRing method), 743
one() (sage.manifolds.differentiable.manifold_homset.IntegratedCurveSet method), 450
one() (sage.manifolds.differentiable.manifold_homset.TopologicalManifoldHomset method), 215
one_form() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 367
one_function() (sage.manifolds.chart.Chart method), 93
one_scalar_field() (sage.manifolds.manifold.TopologicalManifold method), 27
open_cover_family() (sage.manifolds.manifold_subset.ManifoldSubset method), 54
open_covers() (sage.manifolds.subset.ManifoldSubset method), 55
open_interval() (sage.manifolds.differentiable.examples.real_line.OpenInterval method), 410
open_subset() (sage.manifolds.differentiable.degenerate.DegenerateManifold method), 962
open_subset() (sage.manifolds.differentiable.degenerate.ManifoldHomset method), 791
open_subset() (sage.manifolds.differentiable.DifferentiableManifold method), 368
open_subset() (sage.manifolds.differentiable.pseudo_riemannian_submanifold.PseudoRiemannianSubmanifold method), 953
open_subset() (sage.manifolds.differentiable.pseudo_riemannian_manifold.PseudoRiemannianManifold method), 853
open_subset() (sage.manifolds.manifold.TopologicalManifold method), 28
open_subset() (sage.manifolds.manifold_subset.ManifoldSubset method), 55
open_subset() (sage.manifolds.manifold_topological_submanifold.TopologicalSubmanifold method), 248
open_superset_family() (sage.manifolds.manifold_subset.ManifoldSubset method), 56
open_supersets() (sage.manifolds.manifold_subset.ManifoldSubset method), 57
OpenInterval (class in sage.manifolds.differentiable.examples.real_line), 403
options (sage.manifolds.manifold.TopologicalManifold attribute), 29
orientation() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 369
orientation() (sage.manifolds.differentiable.vector_bundle.TensorBundle method), 804
orientation() (sage.manifolds.manifold.TopologicalManifold method), 29
orientation() (sage.manifolds.vector_bundle.TopologicalVectorBundle method), 261

P

periods() (sage.manifolds.chart.Chart method), 94
plot() (sage.manifolds.chart.RealChart method), 113
plot() (sage.manifolds.differentiable.differentiable_curve.DifferentiableCurve method), 472
plot() (sage.manifolds.differentiable.tangent_vector.TangentVector method), 520
plot() (sage.manifolds.differentiable.vectorfield.VectorField method), 563
plot() (sage.manifolds.point.ManifoldPoint method), 77
plot() (sage.manifolds.topological_submanifold.TopologicalSubmanifold method), 249
plot_integrated() (sage.manifolds.differentiable.integrated_curve.IntegratedCurve method), 499
point() (sage.manifolds.subset.ManifoldSubset method), 57
poisson() (sage.manifolds.differentiable.symplectic_form.SymplecticForm method), 990
poisson() (sage.manifolds.differentiable.symplectic_form.SymplecticFormParal method), 993
poisson_bracket() (sage.manifolds.differentiable.poisson_tensor.PoissonTensorField method), 984
poisson_bracket() (sage.manifolds.differentiable.symplectic_form.SymplecticFormParal method), 990
poisson_tensor() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 370
poisson_tensor() (sage.manifolds.differentiable.vectorfield_module.VectorFieldFreeModule method), 538
poisson_tensor() (sage.manifolds.differentiable.vectorfield_module.VectorFieldModule method), 549
PoissonTensorField (class in sage.manifolds.differentiable.poisson_tensor), 983
PoissonTensorFieldParal (class in sage.manifolds.differentiable.poisson_tensor), 985
polar_coordinates() (sage.manifolds.differentiable.examples.euclidean.EuclideanPlane method), 873
polar_frame() (sage.manifolds.differentiable.examples.euclidean.EuclideanPlane method),
PontryaginAlgorithm (class in sage.manifolds.differentiable.characteristic_cohomology_class), 838
PontryaginEulerAlgorithm (class in sage.manifolds.differentiable.characteristic_cohomology_class), 839
preimage() (sage.manifolds.chart.Chart method), 94
preimage() (sage.manifolds.continuous_map.ContinuousMap method), 237
preimage() (sage.manifolds.scalarfield.ScalarField method), 206
principal_curvatures() (sage.manifolds.differentiable.pseudo_riemannian_submanifold.PseudoRiemannianSubmanifold method), 954
principal_directions() (sage.manifolds.differentiable.degenerate_submanifold.DegenerateSubmanifold method), 974
principal_directions() (sage.manifolds.differentiable.pseudo_riemannian_submanifold.PseudoRiemannianSubmanifold method), 955
project() (sage.manifolds.differentiable.pseudo_riemannian_submanifold.PseudoRiemannianSubmanifold method), 956
projection() (sage.manifolds.differentiable.degenerate_submanifold.DegenerateSubmanifold method), 975
projector() (sage.manifolds.differentiable.pseudo_riemannian_submanifold.PseudoRiemannianSubmanifold method), 956
PseudoRiemannianManifold (class in sage.manifolds.differentiable.pseudo_riemannian), 849
PseudoRiemannianMetric (class in sage.manifolds.differentiable.metric), 899
PseudoRiemannianMetricParal (class in sage.manifolds.differentiable.metric), 923
PseudoRiemannianStructure (class in sage.manifolds.differentiable.pseudo_riemannian), 68
PseudoRiemannianSubmanifold (class in sage.manifolds.differentiable.pseudo_riemannian_submanifold), 938
pullback() (sage.manifolds.chart.Chart method), 96
pullback() (sage.manifolds.continuous_map.ContinuousMap method), 238
pullback() (sage.manifolds.differentiable.diff_map.DiffMap method), 462
pullback() (sage.manifolds.differentiable.degenerate_submanifold.DegenerateSubmanifold method), 207
pushforward() (sage.manifolds.differentiable.diff_map.DiffMap method), 463
radius() (sage.manifolds.differentiable.examples.sphere.Sphere method), 887
rank() (sage.manifolds.vector_bundle.TopologicalVectorBundle method), 262
RealChart (class in sage.manifolds.chart), 106
RealDiffChart (class in sage.manifolds.differentiable.chart), 398
RealDifferentialStructure (class in sage.manifolds.differentiable.structure), 68
RealLine (class in sage.manifolds.differentiable.examples.real_line), 411
RealProjectiveSpace() (in module sage.manifolds.catalog), 1013
RealTopologicalStructure (class in sage.manifolds.differentiable.structure), 68
representative() (sage.manifolds.differentiable.characteristic_cohomology_class.CharacteristicCohomologyClassRingElement method), 833
reset() (sage.manifolds.differentiable.chart.RealChart method), 127
restrict() (sage.manifolds.chart.CoordChange method), 104
restrict() (sage.manifolds.chart.RealChart method), 127
restrict() (sage.manifolds.continuous_map.ContinuousMap method), 239
restrict() (sage.manifolds.differentiable.pseudo_riemannian.de_rham_cohomology.DeRhamCohomologyClass method), 742
restrict() (sage.manifolds.differentiable.chart.DiffChart method), 393
restrict() (sage.manifolds.differentiable.chart.RealDiffChart method), 402
restrict() (sage.manifolds.differentiable.affine_connection.AffineConnection method), 779
restrict() (sage.manifolds.differentiable.metric.AffineConnectionFieldParal method), 608
restrict() (sage.manifolds.differentiable.metric.AffineConnectionFieldParal method), 614
restrict() (sage.manifolds.differentiable.metric.PseudoRiemannianMetricParal method), 926
restrict() (sage.manifolds.differentiable.metric.PseudoRiemannianMetricParal method), 926
restrict() (sage.manifolds.differentiable.mixed_form.MixedForm method), 735
restrict() (sage.manifolds.differentiable.symplectic...)
<table>
<thead>
<tr>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sage.manifolds.differentiable.integrated_curve</code> module</td>
<td>481</td>
</tr>
<tr>
<td><code>sage.manifolds.differentiable.levi_civita_connection</code> module</td>
<td>928</td>
</tr>
<tr>
<td><code>sage.manifolds.differentiable.manifold</code> module</td>
<td>335</td>
</tr>
<tr>
<td><code>sage.manifolds.differentiable.manifold_homset</code> module</td>
<td>440</td>
</tr>
<tr>
<td><code>sage.manifolds.differentiable.metric</code> module</td>
<td>894</td>
</tr>
<tr>
<td><code>sage.manifolds.differentiable.mixed_form</code> module</td>
<td>726</td>
</tr>
<tr>
<td><code>sage.manifolds.differentiable.mixed_form_algebra</code> module</td>
<td>721</td>
</tr>
<tr>
<td><code>sage.manifolds.differentiable.multi_vector_module</code> module</td>
<td>744</td>
</tr>
<tr>
<td><code>sage.manifolds.differentiable.multi_vectorfield</code> module</td>
<td>750</td>
</tr>
<tr>
<td><code>sage.manifolds.differentiable.poisson_tensor</code> module</td>
<td>983</td>
</tr>
<tr>
<td><code>sage.manifolds.differentiable.pseudo_riemannian</code> module</td>
<td>845</td>
</tr>
<tr>
<td><code>sage.manifolds.differentiable.pseudo_riemannian_submanifold</code> module</td>
<td>935</td>
</tr>
<tr>
<td><code>sage.manifolds.differentiable.scalarfield</code> module</td>
<td>419</td>
</tr>
<tr>
<td><code>sage.manifolds.differentiable.scalarfield_algebra</code> module</td>
<td>414</td>
</tr>
<tr>
<td><code>sage.manifolds.differentiable.symplectic_form</code> module</td>
<td>987</td>
</tr>
<tr>
<td><code>sage.manifolds.differentiable.tangent_space</code> module</td>
<td>514</td>
</tr>
<tr>
<td><code>sage.manifolds.differentiable.tangent_vector</code> module</td>
<td>518</td>
</tr>
<tr>
<td><code>sage.manifolds.differentiable.tensorfield</code> module</td>
<td>621</td>
</tr>
<tr>
<td><code>sage.manifolds.differentiable.tensorfield_module</code> module</td>
<td>615</td>
</tr>
<tr>
<td><code>sage.manifolds.differentiable.tensorfield_paral</code> module</td>
<td>667</td>
</tr>
<tr>
<td><code>sage.manifolds.differentiable.vector_bundle</code> module</td>
<td>792</td>
</tr>
<tr>
<td><code>sage.manifolds.differentiable.vectorfield</code> module</td>
<td>552</td>
</tr>
<tr>
<td><code>sage.manifolds.differentiable.vectorfield_module</code> module</td>
<td>530</td>
</tr>
<tr>
<td><code>sage.manifolds.differentiable.vectorframe</code> module</td>
<td>579</td>
</tr>
<tr>
<td><code>sage.manifolds.family</code> module</td>
<td>326</td>
</tr>
<tr>
<td><code>sage.manifolds.local_frame</code> module</td>
<td>279</td>
</tr>
<tr>
<td><code>sage.manifolds.manifold</code> module</td>
<td>3</td>
</tr>
<tr>
<td><code>sage.manifolds.manifold_homset</code> module</td>
<td>213</td>
</tr>
<tr>
<td><code>sage.manifolds.operators</code> module</td>
<td>890</td>
</tr>
<tr>
<td><code>sage.manifolds.point</code> module</td>
<td>70</td>
</tr>
<tr>
<td><code>sage.manifolds.scalarfield</code> module</td>
<td>172</td>
</tr>
<tr>
<td><code>sage.manifolds.scalarfield_algebra</code> module</td>
<td>168</td>
</tr>
<tr>
<td><code>sage.manifolds.section</code> module</td>
<td>302</td>
</tr>
<tr>
<td><code>sage.manifolds.section_module</code> module</td>
<td>294</td>
</tr>
<tr>
<td><code>sage.manifolds.structure</code> module</td>
<td>66</td>
</tr>
<tr>
<td><code>sage.manifolds.subset</code> module</td>
<td>37</td>
</tr>
<tr>
<td><code>sage.manifolds.subsets.closure</code> module</td>
<td>328</td>
</tr>
<tr>
<td><code>sage.manifolds.subsets.pullback</code> module</td>
<td>329</td>
</tr>
<tr>
<td><code>sage.manifolds.topological_submanifold</code> module</td>
<td>243</td>
</tr>
<tr>
<td><code>sage.manifolds.trivialization</code> module</td>
<td>272</td>
</tr>
<tr>
<td><code>sage.manifolds.utilities</code> module</td>
<td>997</td>
</tr>
<tr>
<td><code>sage.manifolds.vector_bundle</code> module</td>
<td>252</td>
</tr>
</tbody>
</table>
set_comp() (sage.manifolds.section.Section method), 317
set_comp() (sage.manifolds.section.TrivialSection method), 325
set_connection_form() (sage.manifolds.differentiable.bundle_connection.BundleConnection method), 823
set_coord() (sage.manifolds.point.ManifoldPoint method), 82
set_coordinates() (sage.manifolds.point.ManifoldPoint method), 84
set_default_chart() (sage.manifolds.manifold.TopologicalManifold method), 33
set_default_frame() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 372
set_default_frame() (sage.manifolds.differentiable.vector_bundle.TensorBundle method), 808
set_default_frame() (sage.manifolds.section_module.SectionFreeModule method), 297
set_default_frame() (sage.manifolds.section_module.SectionModule method), 301
set_default_frame() (sage.manifolds.vector_bunddle.TopologicalVectorBundle method), 265
set_embedding() (sage.manifolds.topological_submanifold.TopologicalSubmanifold method), 250
set_expr() (sage.manifolds.chart_func.ChartFunction method), 151
set_expr() (sage.manifolds.continuous_map.ContinuousMap method), 240
set_expr() (sage.manifolds.scalarfield.ScalarField method), 209
set_expression() (sage.manifolds.continuous_map.ContinuousMap method), 241
set_immersion() (sage.manifolds.topological_submanifold.TopologicalSubmanifold method), 251
set_immutable() (sage.manifolds.chart_func.MultiCoordFunction method), 161
set_immutable() (sage.manifolds.differentiable.affine_connection.AffineConnection method), 784
set_immutable() (sage.manifolds.differentiable.bundle_connection.BundleConnection method), 824
set_immutable() (sage.manifolds.differentiable.mixed_form.MixedForm method), 737
set_immutable() (sage.manifolds.differentiable.tensorfield.TensorField method), 659
set_immutable() (sage.manifolds.scalarfield.ScalarField method), 209
set_immutable() (sage.manifolds.section.Section method), 318
set_inverse() (sage.manifolds.chart.CoordChange method), 104
set_name() (sage.manifolds.differentiable.characteristic_cohomology_class.CharacteristicCohomologyClassRingElement method), 834
set_name() (sage.manifolds.differentiable.mixed_form.MixedForm method), 737
set_name() (sage.manifolds.differentiable.tensorfield.TensorField method), 659
set_name() (sage.manifolds.differentiable.vectorframe.CoFrame method), 584
set_name() (sage.manifolds.differentiable.vectorframe.VectorFrame method), 596
set_name() (sage.manifolds.local_frame.LocalCoFrame method), 283
set_name() (sage.manifolds.local_frame.LocalFrame method), 290
set_name() (sage.manifolds.scalarfield.ScalarField method), 210
set_name() (sage.manifolds.section.Section method), 319
set_orientation() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 372
set_orientation() (sage.manifolds.differentiable.affine_connection.AffineConnection method), 784
set_orientation() (sage.manifolds.differentiable.mixed_form.MixedForm method), 737
set_orientation() (sage.manifolds.differentiable.tensorfield.tensorfield.TensorField method), 660
set_orientation() (sage.manifolds.ofields.tensorfield.TensorField method), 210
set_orientation() (sage.manifolds.section.Section method), 319
set_simplify_function() (sage.manifolds.calculus_method.CalculusMethod method), 164
set_transverse() (sage.manifolds.differentiable.degenerate_submanifold.DegenerateSubmanifold method), 977
shape_operator() (sage.manifolds.differentiable.degenerate_submanifold.DegenerateSubmanifold method), 978
shape_operator() (sage.manifolds.differentiable.pseudo_riemannian_submanifold.PseudoRiemannianSubmanifold method), 958
sphere() (sage.manifolds.differentiable.examples.euclidean.Euclidean3dimSpace method), 868
sphere() (sage.manifolds.differentiable.examples.euclidean.Euclidean.EuclideanSpace method), 880
spherical_coordinates() (sage.manifolds.differentiable.examples.euclidean.Euclidean3dimSpace method), 868
spherical_coordinates() (sage.manifolds.differentiable.examples.euclidean.Euclidean.EuclideanSpace method), 880
spherical_frame() (sage.manifolds.differentiable.examples.euclidean.Euclidean3dimSpace method), 869
sqrt() (sage.manifolds.chart_func.ChartFunction method), 154
sqrt() (sage.manifolds.scalarfield.ScalarField method), 211
sqrt_abs_det() (sage.manifolds.differentiable.metric.PseudoRiemannianMetric method), 150
StandardSymplecticSpace (class in sage.manifolds.differentiable.examples.symplectic_space), 994
start_index() (sage.manifolds.manifold.TopologicalManifold method), 35
stereographic_coordinates() (sage.manifolds.differentiable.examples.sphere.Sphere method), 888
structure_coeff() (sage.manifolds.differentiable.vectorframe.CoordFrame method), 587
structure_coeff() (sage.manifolds.differentiable.vectorframe.VectorFrame method), 597
subcategory() (sage.manifolds.structure.DegenerateStructure method), 67
subcategory() (sage.manifolds.structure.DifferentialStructure method), 67
subcategory() (sage.manifolds.structure.LorentzianStructure method), 67
subcategory() (sage.manifolds.structure.PseudoRiemannianStructure method), 68
subcategory() (sage.manifolds.structure.RealDifferentialStructure method), 68
subcategory() (sage.manifolds.structure.RealTopologicalStructure method), 69
subcategory() (sage.manifolds.structure.RiemannianStructure method), 69
subcategory() (sage.manifolds.structure.TopologicalStructure method), 70
subset() (sage.manifolds.subset.ManifoldSubset method), 58
subset_digraph() (sage.manifolds.subset.ManifoldSubset method), 59
subset_family() (sage.manifolds.subset.ManifoldSubset method), 61
subsets() (sage.manifolds.subset.ManifoldSubset method), 62

vector_bundle() (sage.manifolds.section_module.SectionFreeModule method), 298
vector_bundle() (sage.manifolds.section_module.SectionModule method), 301
vector_bundle() (sage.manifolds.trivialization.Trivialization method), 278
vector_field() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 382
vector_field_module() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 384
vector_field_module() (sage.manifolds.differentiable.mixed_form_algebra.MixedFormAlgebra method), 725
vector_frame() (sage.manifolds.differentiable.manifold.DifferentiableManifold method), 386
vector_frame() (sage.manifolds.differentiable.vector_bundle.TensorBundle method), 812
VectorBundleFiber (class in sage.manifolds.vector_bundle_fiber), 268
VectorBundleFiberElement (class in sage.manifolds.vector_bundle_fiber_element), 271
VectorField (class in sage.manifolds.differentiable.vectorfield), 552
VectorFieldDualFreeModule (class in sage.manifolds.differentiable.diff_form_module), 700
VectorFieldFreeModule (class in sage.manifolds.differentiable.vectorfield_module), 530
VectorFieldModule (class in sage.manifolds.differentiable.vectorfield_module), 541
VectorFieldParal (class in sage.manifolds.differentiable.vectorfield), 575
VectorFrame (class in sage.manifolds.differentiable.vectorfield), 587
volume_form() (sage.manifolds.differentiable.metric.PseudoRiemannianMetric method), 920
volume_form() (sage.manifolds.differentiable.pseudo_riemannian.PseudoRiemannianManifold method), 854
volume_form() (sage.manifolds.differentiable.symplectic_form.SymplecticForm method), 991
wedge() (sage.manifolds.differentiable.scalarfield.DiffScalarField method), 440
weingarten_map() (sage.manifolds.differentiable.degenerate_submanifold.DegenerateSubmanifold method), 978
weyl() (sage.manifolds.differentiable.metric.PseudoRiemannianMetric method), 922
wrap() (sage.manifolds.differentiable.symplectic_form.SymplecticForm static method), 992
X
xder() (in module sage.manifolds.utilities), 1009
Z
zero() (sage.manifolds.chart_func.ChartFunctionRing method), 157
zero() (sage.manifolds.differentiable.de_rham_cohomology.DeRhamCohomologyRing method), 743
zero() (sage.manifolds.differentiable.diff_form_module.DiffFormModule method), 699
zero() (sage.manifolds.differentiable.mixed_form_algebra.MixedFormAlgebra method), 725
zero() (sage.manifolds.differentiable.tensorfield_module.TensorFieldModule method), 621
zero() (sage.manifolds.differentiable.vectorfield_module.VectorFieldModule method), 552
zero() (sage.manifolds.scalarfield_algebra.ScalarFieldAlgebra method), 172
zero() (sage.manifolds.section_module.SectionModule method), 301
zero_function() (sage.manifolds.chart.Chart method), 101
zero_scalar_field() (sage.manifolds.manifold.TopologicalManifold method), 37
W
wedge() (sage.manifolds.differentiable.diff_form.DiffForm method), 712
wedge() (sage.manifolds.differentiable.diff_form.DiffFormParal method), 720
wedge() (sage.manifolds.differentiable.mixed_form.MixedForm method), 738
wedge() (sage.manifolds.differentiable.multivectorfield.MultivectorField method), 756
wedge() (sage.manifolds.differentiable.multivectorfield.MultivectorFieldParal method), 764