Sage Reference Manual: Manifolds

Release 7.3

The Sage Development Team

Aug 04, 2016

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.

CHAPTER

ONE

TOPOLOGICAL MANIFOLDS

1.1 Topological Manifolds

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Thanks to the operator `<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)
<type 'sage.symbolic.expression.Expression'>
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Given the definition of p, we have of course:

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

Similarly:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

and we have:

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

The following subsets and charts have been defined:

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

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

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

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

AUTHORS:

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

REFERENCES:

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

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

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

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

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

INPUT:

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

OUTPUT:

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

EXAMPLES:

A 3-dimensional real topological manifold:

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

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

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

A complex topological manifold:

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

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

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

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

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

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

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

A 3-dimensional real smooth manifold:

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

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

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

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

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

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

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

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

Uniqueness of manifold objects

Suppose we construct a manifold named M:

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

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

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

This results in a brand new object:

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

The object `M_old` is intact:

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

Both objects have the same display:

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

but they are different:

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

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

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

The charts are displayed in the same way:

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

but they are actually different:

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

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

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

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

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

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

Bases: *`sage.manifolds.subset.ManifoldSubset`*

Topological manifold over a topological field K.

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

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

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

INPUT:

- `n` – positive integer; dimension of the manifold
- `name` – string; name (symbol) given to the manifold
- `field` – field K on which the manifold is defined; allowed values are
  - `'real'` or an object of type `RealField` (e.g., `RR`) for a manifold over $\mathbf{R}$
  - `'complex'` or an object of type `ComplexField` (e.g., `CC`) for a manifold over $\mathbf{C}$
  - an object in the category of topological fields (see `Fields` and `TopologicalSpaces`) for other types of manifolds
- `structure` – manifold structure (see *`TopologicalStructure`* or *`RealTopologicalStructure`*)
- `ambient` – (default: `None`) if not `None`, must be a topological manifold; the created object is then an open subset of `ambient`
- `latex_name` – (default: `None`) string; LaTeX symbol to denote the manifold; if none are provided, it is set to `name`
- `start_index` – (default: 0) integer; lower value of the range of indices used for “indexed objects” on the manifold, e.g., coordinates in a chart
- `category` – (default: `None`) to specify the category; if `None`, `Manifolds(field)` is assumed (see the category `Manifolds`)

- `unique_tag` – (default: `None`) tag used to force the construction of a new object when all the other arguments have been used previously (without `unique_tag`, the `UniqueRepresentation` behavior inherited from *`ManifoldSubset`* would return the previously constructed object corresponding to these arguments)

EXAMPLES:

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

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

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

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

A complex manifold:

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

A manifold over $\mathbf{Q}$:

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

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

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

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

```
sage: isinstance(M, Parent)
True
sage: M.category()
Category of manifolds over Real Field with 53 bits of precision
sage: from sage.categories.manifolds import Manifolds
sage: M.category() is Manifolds(RR)
True
sage: M.category() is Manifolds(M.base_field())
```

```
True
sage: M in Manifolds(RR)
True
sage: N in Manifolds(Qp(5))
True
```

The corresponding Sage *elements* are points:

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

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

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

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

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

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

```
sage: TestSuite(M).run()
```

See also:

sage.manifolds.manifold

atlas ( )

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

EXAMPLES:

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

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

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

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

Let us introduce the chart of Cartesian coordinates:

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

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

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

Spherical (polar) coordinates on `U` :

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

See also:

`top_charts()`

`base_field` ( )

Return the field on which the manifold is defined.

OUTPUT:

•a topological field

EXAMPLES:

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

`base_field_type` ( )

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

OUTPUT:

•a string describing the field, with three possible values:

–`'real'` for the real field $\mathbf{R}$

–`'complex'` for the complex field $\mathbf{C}$

–`'neither_real_nor_complex'` for a field different from $\mathbf{R}$ and $\mathbf{C}$

EXAMPLES:

```
sage: M = Manifold(3, 'M', structure='topological')
sage: M.base_field_type()
```

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

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

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

A *chart* is a pair (U, φ), where U is the current manifold and $\varphi : U \to V \subset K^n$ 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 φ, 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, φ).

See `Chart` for a complete documentation.

INPUT:

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

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

1. The coordinate symbol (a letter or a few letters).
2. (optional, only for manifolds over $\mathbf{R}$) The interval I defining the coordinate range: if not provided, the coordinate is assumed to span all $\mathbf{R}$; otherwise I must be provided in the form `(a,b)` (or equivalently `]a,b[`) The bounds `a` and `b` can be `+/-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) The LaTeX spelling of the coordinate; if not provided the coordinate symbol given in the first field will be used.

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

OUTPUT:

- the created chart, as an instance of `Chart` or of the subclass `RealChart` for manifolds over $\mathbf{R}$.

EXAMPLES:

Chart on a 2-dimensional manifold:

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

```
y
sage: X[:]
(x, y)
```

The declared coordinates are not known at the global level:

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

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

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

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

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

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

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

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

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

`constant_scalar_field` (*value*, *name=None*, *latex_name=None*)

Define a constant scalar field on the manifold.

INPUT:

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

OUTPUT:

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

EXAMPLES:

A constant scalar field on the 2-sphere:

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

We have:

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

See also:

zero_scalar_field(), *one_scalar_field()*

continuous_map (*codomain*, *coord_functions=None*, *chart1=None*, *chart2=None*, *name=None*, *latex_name=None*)

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

INPUT:

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

  in both cases, if the dimension of the codomain is 1, a single coordinate expression can be passed instead of a tuple with a single element
- `chart1` – (default: `None`; used only in case (ii) above) chart on `self` defining the start coordinates involved in `coord_functions` for case (ii); if `None`, the coordinates are assumed to refer to the default chart of `self`
- `chart2` – (default: `None`; used only in case (ii) above) chart on `codomain` defining the target coordinates involved in `coord_functions` for case (ii); if `None`, the coordinates are assumed to refer to the default chart of `codomain`
- `name` – (default: `None`) name given to the continuous map

- `latex_name` – (default: `None`) LaTeX symbol to denote the continuous map; if `None` , the LaTeX symbol is set to `name`

OUTPUT:

- the continuous map as an instance of *ContinuousMap*

EXAMPLES:

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

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

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

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

The continuous map acting on a point:

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

TESTS:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: M.continuous_map(ZZ)
Traceback (most recent call last):
...
ValueError: Integer Ring is not a manifold
 over Real Field with 53 bits of precision
```

See also:

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

Todo

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

coord_change (*chart1*, *chart2*)
Return the change of coordinates (transition map) between two charts defined on the manifold.

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

INPUT:

- chart1 – chart 1
- chart2 – chart 2

OUTPUT:

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

EXAMPLES:

Change of coordinates on a 2-dimensional manifold:

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

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

OUTPUT:

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

EXAMPLES:

Various changes of coordinates on a 2-dimensional manifold:

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

default_chart ( )

Return the default chart defined on the manifold.

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

OUTPUT:

- instance of `Chart` representing the default chart

EXAMPLES:

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

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

dim ( )

Return the dimension of the manifold over its base field.

EXAMPLES:

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

A shortcut is `dim()`:

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

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

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

dimension ( )

Return the dimension of the manifold over its base field.

EXAMPLES:

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

A shortcut is `dim()` :

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

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

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

get_chart (*coordinates*, *domain=None*)

Get a chart from its coordinates.

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

INPUT:

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

OUTPUT:

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

EXAMPLES:

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

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

global_options (**args*, ***kwds*)

Deprecated: Use *options()* instead. See trac ticket #18555 for details.

homeomorphism (*codomain*, *coord_functions=None*, *chart1=None*, *chart2=None*, *name=None*, *latex_name=None*)

Define a homeomorphism between the current manifold and another one.

See *ContinuousMap* for a complete documentation.

INPUT:

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

  in both cases, if the dimension of the codomain is 1, a single coordinate expression can be passed instead of a tuple with a single element
- `chart1` – (default: `None` ; used only in case (ii) above) chart on `self` defining the start coordinates involved in `coord_functions` for case (ii); if `None` , the coordinates are assumed to refer to the default chart of `self`
- `chart2` – (default: `None` ; used only in case (ii) above) chart on `codomain` defining the target coordinates involved in `coord_functions` for case (ii); if `None` , the coordinates are assumed to refer to the default chart of `codomain`
- `name` – (default: `None`) name given to the homeomorphism
- `latex_name` – (default: `None`) LaTeX symbol to denote the homeomorphism; if `None` , the LaTeX symbol is set to `name`

OUTPUT:

- the homeomorphism, as an instance of *ContinuousMap*

EXAMPLES:

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

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

```
Phi: M --> N
   (x, y) |--> (X, Y) = (x/sqrt(-x^2 - y^2 + 1), y/sqrt(-x^2 - y^2 + 1))
```

The inverse homeomorphism:

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

See the documentation of `ContinuousMap` for more examples.

identity_map ( )

Identity map of `self`.

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

$$\begin{array}{llll} \mathrm{Id}_M : & M & \longrightarrow & M \\ & p & \longmapsto & p \end{array}$$

OUTPUT:

•the identity map as an instance of `ContinuousMap`

EXAMPLES:

Identity map of a complex manifold:

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

The identity map acting on a point:

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

See also:

See `ContinuousMap` for the complete documentation.

index_generator (*nb_indices*)

Generator of index series.

INPUT:

•`nb_indices` – number of indices in a series

OUTPUT:

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

EXAMPLES:

Indices on a 2-dimensional manifold:

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

Loops can be nested:

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

irange (*start=None*)

Single index generator.

INPUT:

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

OUTPUT:

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

EXAMPLES:

Index range on a 4-dimensional manifold:

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

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

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

In general, one has always:

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

is_manifestly_coordinate_domain ( )

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

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

EXAMPLES:

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

is_open ( )

Return if `self` is an open set.

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

TEST:

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

one_scalar_field ( )

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

OUTPUT:

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

EXAMPLES:

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

open_subset (*name*, *latex_name=None*, *coord_def={}*)

Create an open subset of the manifold.

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

INPUT:

- `name` – name given to the open subset
- `latex_name` – (default: `None`) LaTeX symbol to denote the subset; if none are provided, it is set to `name`

- `coord_def` – (default: {}) definition of the subset in terms of coordinates; `coord_def` must a be dictionary with keys charts on the manifold and values the symbolic expressions formed by the coordinates to define the subset

OUTPUT:

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

EXAMPLES:

Creating an open subset of a 2-dimensional manifold:

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

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

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

Creating an open subset of `A`:

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

We have then:

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

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

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

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

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

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

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

options (**get_value*, ***set_value*)

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

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

OPTIONS:

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

EXAMPLES:

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

For coordinate functions, the display is more "textbook" like:

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

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

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

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

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

We can switch it off by:

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

See `GlobalOptions` for more features of these options.

scalar_field (*coord_expression=None*, *chart=None*, *name=None*, *latex_name=None*)
Define a scalar field on the manifold.

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

INPUT:

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

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

OUTPUT:

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

EXAMPLES:

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

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

Equivalent definition with the chart specified:

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

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

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

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

See also:

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

`scalar_field_algebra` ( )

Return the algebra of scalar fields defined the manifold.

See `ScalarFieldAlgebra` for a complete documentation.

OUTPUT:

- instance of `ScalarFieldAlgebra` representing the algebra $C^0(U)$ of all scalar fields defined on U = `self`

EXAMPLES:

Scalar algebra of a 3-dimensional open subset:

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

The output is cached:

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

`set_default_chart` (*chart*)

Changing the default chart on `self` .

INPUT:

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

EXAMPLES:

Charts on a 2-dimensional manifold:

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

start_index ( )

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

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

OUTPUT:

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

EXAMPLES:

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

top_charts ( )

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

OUTPUT:

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

EXAMPLES:

Charts on a 2-dimensional manifold:

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

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

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

See also:

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

zero_scalar_field ( )

Return the zero scalar field defined on `self`.

OUTPUT:

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

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.zero_scalar_field() ; f
Scalar field zero on the 2-dimensional topological manifold M
sage: f.display()
zero: M --> R
   (x, y) |--> 0
```

```
sage: f.parent()
Algebra of scalar fields on the 2-dimensional topological manifold M
sage: f is M.scalar_field_algebra().zero()
True
```

1.2 Subsets of Topological Manifolds

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

AUTHORS:

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

REFERENCES:

- *[Lee11]* J.M. Lee : *Introduction to Topological Manifolds*, 2nd ed., Springer (New York) (2011)

EXAMPLES:

Two subsets on a manifold:

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

The intersection of the two subsets:

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

Their union:

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

Lists of subsets after the above operations:

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

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

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

Bases: `sage.structure.unique_representation.UniqueRepresentation`, `sage.structure.parent.Parent`

Subset of a topological manifold.

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

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

INPUT:

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

EXAMPLES:

A subset of a manifold:

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

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

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

The manifold is itself a subset:

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

Instances of `ManifoldSubset` are parents:

```
sage: isinstance(A, Parent)
True
sage: A.category()
Category of subobjects of sets
sage: p = A.an_element(); p
Point on the 2-dimensional topological manifold M
sage: p.parent()
Subset A of the 2-dimensional topological manifold M
sage: p in A
True
sage: p in M
True
```

`Element`
alias of `ManifoldPoint`

`ambient` ( )
Return the ambient manifold of `self`.

EXAMPLES:

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

An alias is `ambient`:

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

`declare_union` (*dom1*, *dom2*)
Declare that the current subset is the union of two subsets.

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

$$U = U_1 \cup U_2,$$

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

INPUT:

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

EXAMPLES:

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

get_subset (*name*)

Get a subset by its name.

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

INPUT:

- `name` – (string) name of the subset

OUTPUT:

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

EXAMPLES:

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

intersection (*other*, *name=None*, *latex_name=None*)

Return the intersection of the current subset with another subset.

INPUT:

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

OUTPUT:

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

EXAMPLES:

Intersection of two subsets:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: a = M.subset('A')
sage: b = M.subset('B')
```

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

Some checks:

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

is_open ( )

Return if `self` is an open set.

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

EXAMPLES:

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

is_subset (*other*)

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

EXAMPLES:

Subsets on a 2-dimensional manifold:

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

```
sage: b.is_subset(M)
True
sage: a.is_subset(b)
False
sage: c.is_subset(a)
False
```

lift (*p*)

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

INPUT:

- `p` – point of the subset

OUTPUT:

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

EXAMPLES:

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

list_of_subsets ()

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

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

OUTPUT:

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

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

EXAMPLES:

Subsets of a 2-dimensional manifold:

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

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

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

manifold ( )

Return the ambient manifold of `self`.

EXAMPLES:

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

An alias is `ambient`:

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

open_covers ( )

Return the list of open covers of the current subset.

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

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

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

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

EXAMPLES:

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

```
[[2-dimensional topological manifold M],
 [Open subset U of the 2-dimensional topological manifold M,
  Open subset V of the 2-dimensional topological manifold M],
 [Open subset A of the 2-dimensional topological manifold M,
  Open subset B of the 2-dimensional topological manifold M,
  Open subset V of the 2-dimensional topological manifold M]]
```

point (*coords=None*, *chart=None*, *name=None*, *latex_name=None*)

Define a point in `self` .

See *ManifoldPoint* for a complete documentation.

INPUT:

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

OUTPUT:

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

EXAMPLES:

Points on a 2-dimensional manifold:

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

retract (*p*)

Return the retract of `p` to `self` .

INPUT:

- `p` – point of the ambient manifold

OUTPUT:

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

EXAMPLES:

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

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

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

subset (*name*, *latex_name=None*, *is_open=False*)

Create a subset of the current subset.

INPUT:

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

OUTPUT:

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

EXAMPLES:

Creating a subset of a manifold:

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

Creating a subset of `A`:

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

We have then:

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

subsets ( )

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

OUTPUT:

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

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

EXAMPLES:

Subsets of a 2-dimensional manifold:

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

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

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

superset (*name*, *latex_name=None*, *is_open=False*)

Create a superset of the current subset.

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

INPUT:

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

OUTPUT:

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

EXAMPLES:

Creating some superset of a given subset:

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

The superset of the whole manifold is itself:

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

Two supersets of a given subset are a priori different:

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

union (*other*, *name=None*, *latex_name=None*)

Return the union of the current subset with another subset.

INPUT:

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

OUTPUT:

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

EXAMPLES:

Union of two subsets:

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

```
set([subset 'A_union_B' of the 2-dimensional manifold 'M',
    subset 'A' of the 2-dimensional manifold 'M',
    subset 'B' of the 2-dimensional manifold 'M'])
```

Some checks:

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

1.3 Manifold Structures

These classes encode the structure of a manifold.

AUTHORS:

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

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

Bases: `sage.misc.fast_methods.Singleton`

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

`chart`

alias of `DiffChart`

`homset`

alias of `DifferentiableManifoldHomset`

`scalar_field_algebra`

alias of `DiffScalarFieldAlgebra`

`subcategory` (*cat*)

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

EXAMPLE:

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

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

Bases: `sage.misc.fast_methods.Singleton`

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

chart
alias of `RealDiffChart`

homset
alias of `DifferentiableManifoldHomset`

scalar_field_algebra
alias of `DiffScalarFieldAlgebra`

subcategory (*cat*)
Return the subcategory of `cat` corresponding to the structure of `self`.

EXAMPLE:

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

class `sage.manifolds.structure.`**`RealTopologicalStructure`**
Bases: `sage.misc.fast_methods.Singleton`

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

chart
alias of `RealChart`

homset
alias of `TopologicalManifoldHomset`

scalar_field_algebra
alias of `ScalarFieldAlgebra`

subcategory (*cat*)
Return the subcategory of `cat` corresponding to the structure of `self`.

EXAMPLES:

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

class `sage.manifolds.structure.`**`TopologicalStructure`**
Bases: `sage.misc.fast_methods.Singleton`

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

chart
alias of `Chart`

homset
alias of `TopologicalManifoldHomset`

scalar_field_algebra
alias of `ScalarFieldAlgebra`

subcategory (*cat*)
Return the subcategory of `cat` corresponding to the structure of `self`.

EXAMPLES:

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

1.4 Points of Topological Manifolds

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

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

AUTHORS:

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

REFERENCES:

- [Lee11] J.M. Lee : *Introduction to Topological Manifolds*, 2nd ed., Springer (New York) (2011)
- [Lee13] J.M. Lee : *Introduction to Smooth Manifolds*, 2nd ed., Springer (New York, 2013)

EXAMPLES:

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

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

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

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

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

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

Points can be compared:

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

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

Bases: `sage.structure.element.Element`

Point of a topological manifold.

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

INPUT:

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

EXAMPLES:

A point on a 2-dimensional manifold:

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

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

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

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

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

```
Open subset U of the 2-dimensional topological manifold M
sage: q in U
True
sage: q in M
True
```

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

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

But it can be set to any value:

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

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

add_coord (*coords*, *chart=None*)

Adds some coordinates in the specified chart.

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

INPUT:

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

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

EXAMPLES:

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

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

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

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

A shortcut for `add_coordinates` is `add_coord`:

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

Let us introduce a second chart on the manifold:

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

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

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

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

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

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

Adds some coordinates in the specified chart.

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

INPUT:

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

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

EXAMPLES:

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

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

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

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

A shortcut for `add_coordinates` is `add_coord`:

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

Let us introduce a second chart on the manifold:

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

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

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

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

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

coord (*chart=None*, *old_chart=None*)

Return the point coordinates in the specified chart.

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

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

INPUT:

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

EXAMPLES:

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

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

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

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

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

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

A shortcut for coordinates is coord:

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

Computing the Cartesian coordinates from the spherical ones:

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

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

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

Coordinates of a point on a 2-dimensional manifold:

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

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

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

Coordinates of `P` in a new chart:

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

Coordinates of `P` in a third chart:

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

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

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

`coordinates` (*chart=None*, *old_chart=None*)

Return the point coordinates in the specified chart.

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

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

INPUT:

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

EXAMPLES:

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

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

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

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

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

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

A shortcut for `coordinates` is `coord`:

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

Computing the Cartesian coordinates from the spherical ones:

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

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

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

Coordinates of a point on a 2-dimensional manifold:

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

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

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

Coordinates of `P` in a new chart:

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

Coordinates of `P` in a third chart:

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

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

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

plot (*chart=None*, *ambient_coords=None*, *mapping=None*, *label=None*, *parameters=None*, *color='black'*, *label_offset=0.1*, *fontsize=10*, *label_color=None*, *size=10*, ***kwds*)

For real manifolds, plot `self` in a Cartesian graph based on the coordinates of some ambient chart.

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

INPUT:

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

- `fontsize` – (default: 10) size of the font used to print the label
- `label_offset` – (default: 0.1) determines the separation between the point and its label

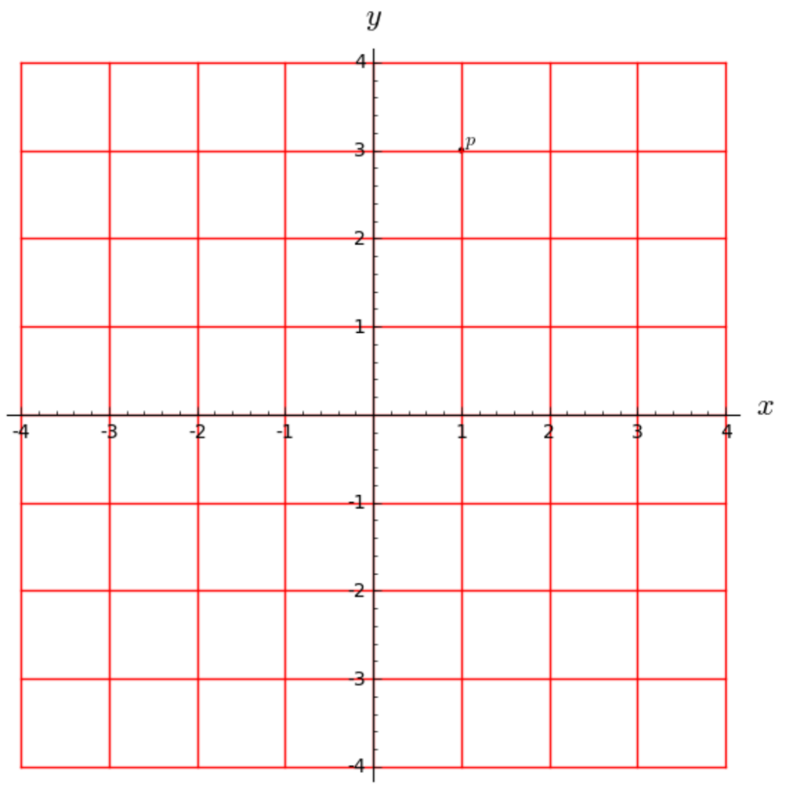
OUTPUT:

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

EXAMPLES:

Drawing a point on a 2-dimensional manifold:

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

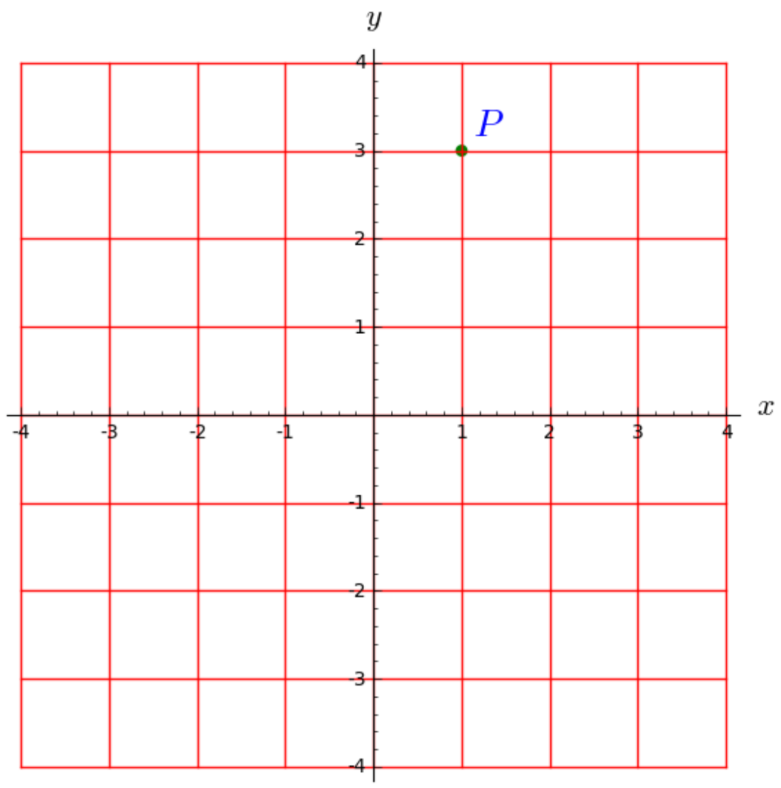


Actually, since `X` is the default chart of the open set in which `p` has been defined, it can be skipped in the arguments of `plot`:

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

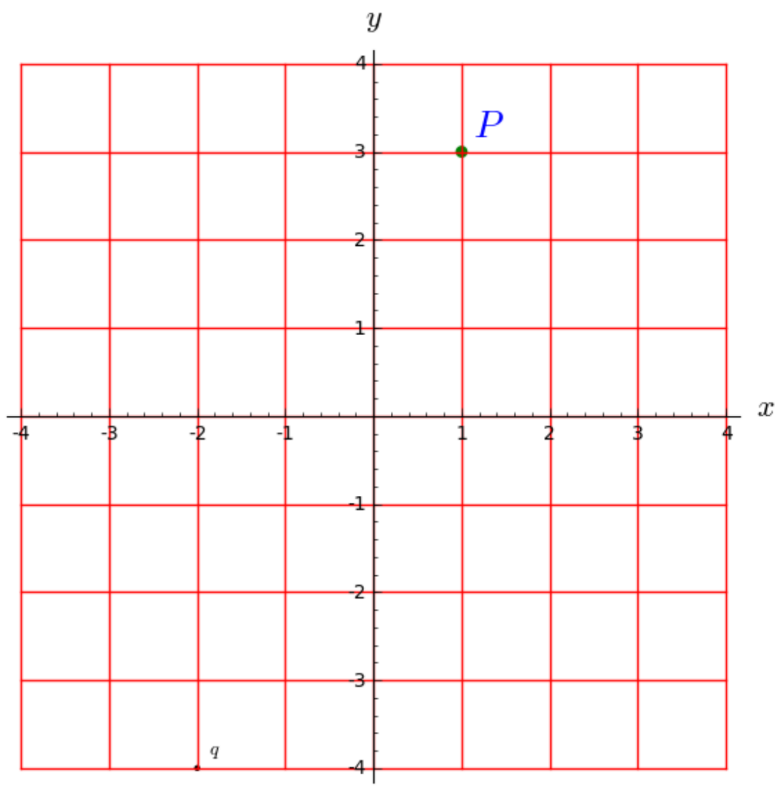
Call with some options:

```
sage: g = p.plot(chart=X, size=40, color='green', label='$P$',
....:             label_color='blue', fontsize=20, label_offset=0.3)
sage: g + gX
Graphics object consisting of 20 graphics primitives
```



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

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



The numerical value is used only for the plot:

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

Drawing a point on a 3-dimensional manifold:

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

Call with some options:

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

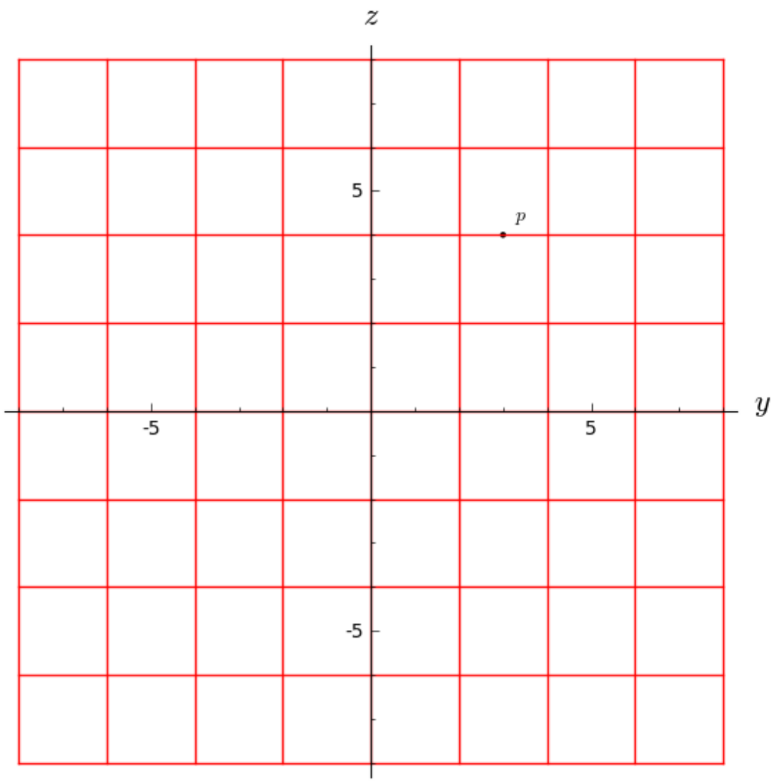
An example of plot via a mapping: plot of a point on a 2-sphere viewed in the 3-dimensional space `M`:

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

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

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

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



set_coord (*coords*, *chart=None*)

Sets the point coordinates in the specified chart.

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

INPUT:

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

EXAMPLES:

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

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

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

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

A shortcut for `set_coordinates` is `set_coord` :

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

Let us introduce a second chart on the manifold:

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

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

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

`set_coordinates` (*coords*, *chart=None*)

Sets the point coordinates in the specified chart.

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

INPUT:

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

EXAMPLES:

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

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

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

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

A shortcut for `set_coordinates` is `set_coord`:

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

Let us introduce a second chart on the manifold:

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

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

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

1.5 Coordinate Charts

1.5.1 Coordinate Charts

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

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

AUTHORS:

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

REFERENCES:

- Chap. 2 of *[Lee11]* J.M. Lee: *Introduction to Topological Manifolds*, 2nd ed., Springer (New York) (2011)
- Chap. 1 of *[Lee13]* J.M. Lee : *Introduction to Smooth Manifolds*, 2nd ed., Springer (New York) (2013)

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

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

Chart on a topological manifold.

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

The components $(x^1, \ldots, x^n)$ of φ, 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, φ).

INPUT:

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

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

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

If it contains any LaTeX expression, the string coordinates must be declared with the prefix 'r' (for "raw") to allow for a proper treatment of LaTeX's backslash character (see examples below). If no LaTeX spelling is to be set for any coordinate, the argument `coordinates` can be omitted when the shortcut operator `<,>` is used via Sage preparser (see examples below).

EXAMPLES:

A chart on a complex 2-dimensional topological manifold:

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

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

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

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

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

The coordinates are then immediately accessible:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Some partial sets of coordinates:

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

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

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

and to the atlas of the chart's domain:

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

Manifold subsets have a *default chart*, which, unless changed via the method *set_default_chart()* , is the first defined chart on the subset (or on 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 φ acting on a point is obtained by passing it as an input to the map:

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

See also:

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

add_restrictions (*restrictions*)

Add some restrictions on the coordinates.

INPUT:

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

A restriction can be any symbolic equality or inequality involving the coordinates, such as `x > y` or `x^2 + y^2 != 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]` .

EXAMPLES:

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

domain ( )

Return the open subset on which the chart is defined.

EXAMPLES:

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

function (*expression*)

Define a coordinate function to the base field.

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

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

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

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

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

INPUT:

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

OUTPUT:

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

EXAMPLES:

A symbolic coordinate function:

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

function_ring ( )

Return the ring of coordinate functions on `self`.

EXAMPLES:

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

manifold ( )

Return the manifold on which the chart is defined.

EXAMPLES:

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

multifunction (**expressions*)

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

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

$$\begin{array}{llll} f: & V \subset K^n & \longrightarrow & K^m \\ & (x^1, \ldots, x^n) & \longmapsto & (f_1(x^1, \ldots, x^n), \ldots, f_m(x^1, \ldots, x^n)), \end{array}$$

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

See `MultiCoordFunction` for a complete documentation.

INPUT:

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

OUTPUT:

- a `MultiCoordFunction` representing f

EXAMPLES:

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

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

TESTS:

```
sage: type(f)
<class 'sage.manifolds.coord_func.MultiCoordFunction'>
```

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

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

where V is the chart codomain.

See class `CoordFunctionSymb` for a complete documentation.

OUTPUT:

- a `CoordFunctionSymb` representing the one coordinate function f

EXAMPLES:

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

The result is cached:

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

One function on a p-adic manifold:

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

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

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

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

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

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

INPUT:

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

A restriction can be any symbolic equality or inequality involving the coordinates, such as `x > y` or `x^2 + y^2 != 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:

- chart (V, ψ) as a `Chart`

EXAMPLES:

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

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

transition_map (*other*, *transformations*, *intersection_name=None*, *restrictions1=None*, *restrictions2=None*)

Construct the transition map between the current chart, (U, φ) say, and another one, (V, ψ) 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, ψ) in terms of the coordinates $(x^1, \ldots, x^n)$ of (U, φ) on the open subset where the two charts intersect, i.e. on $U \cap V$.

INPUT:

- `other` – the chart (V, ψ)
- `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, ψ) 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 != 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:

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

EXAMPLES:

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

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

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

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

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

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

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

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

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

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

valid_coordinates (**coordinates*, ***kwds*)

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

INPUT:

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

OUTPUT:

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

EXAMPLES:

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

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

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

zero_function ( )

Return the zero function of the coordinates.

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

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

where V is the chart codomain.

See class *CoordFunctionSymb* for a complete documentation.

OUTPUT:

- a *CoordFunctionSymb* representing the zero coordinate function f

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: X.zero_function()
0
```

```
sage: X.zero_function().display()
(x, y) |--> 0
sage: type(X.zero_function())
<class 'sage.manifolds.coord_func_symb.CoordFunctionSymbRing_with_category.
↪element_class'>
```

The result is cached:

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

Zero function on a p-adic manifold:

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

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

Bases: `sage.structure.sage_object.SageObject`

Transition map between two charts of a topological manifold.

Giving two coordinate charts (U, φ) and (V, ψ) on a topological manifold M of dimension n over a topological field K, the *transition map from* (U, φ) *to* (V, ψ) 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, ψ) in terms of the coordinates $(x^1, \ldots, x^n)$ of (U, φ) on the open subset where the two charts intersect, i.e. on $U \cap V$.

INPUT:

- `chart1` – chart (U, φ)
- `chart2` – chart (V, ψ)
- `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(r'r:(0,+oo) ph:(0,2*pi):\phi')
sage: spher_to_cart = c_spher.transition_map(c_cart, [r*cos(ph), r*sin(ph)])
sage: spher_to_cart.display()
x = r*cos(ph)
y = r*sin(ph)
sage: latex(spher_to_cart.display())
\left\{\begin{array}{lcl} x & = & r \cos\left({\phi}\right) \\
 y & = & r \sin\left({\phi}\right) \end{array}\right.
```

A shortcut is `disp()`:

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

display ( )

Display of the coordinate transformation.

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

EXAMPLES:

From spherical coordinates to Cartesian ones in the plane:

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

A shortcut is `disp()`:

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

inverse ( )

Compute the inverse coordinate transformation.

OUTPUT:

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

EXAMPLES:

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

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: c_uv.<u,v> = M.chart()
sage: xy_to_uv = c_xy.transition_map(c_uv, ((x - sqrt(3)*y)/2, (sqrt(3)*x +␣
↪y)/2))
sage: M.coord_changes()
{(Chart (M, (x, y)),
  Chart (M, (u, v))): Change of coordinates from Chart (M, (x, y)) to Chart␣
↪(M, (u, v))}
sage: uv_to_xy = xy_to_uv.inverse(); uv_to_xy
Change of coordinates from Chart (M, (u, v)) to Chart (M, (x, y))
sage: uv_to_xy.display()
x = 1/2*sqrt(3)*v + 1/2*u
y = -1/2*sqrt(3)*u + 1/2*v
sage: M.coord_changes()   # random (dictionary output)
{(Chart (M, (u, v)),
  Chart (M, (x, y))): Change of coordinates from Chart (M, (u, v)) to Chart␣
↪(M, (x, y)),
 (Chart (M, (x, y)),
  Chart (M, (u, v))): Change of coordinates from Chart (M, (x, y)) to Chart␣
↪(M, (u, v))}
```

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

Restriction to subsets.

INPUT:

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

OUTPUT:

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

EXAMPLES:

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

The result is cached:

```
sage: X_to_Y.restrict(U) is X_to_Y_U
True
```

set_inverse (**transformations*, ***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` – keyword arguments: only `verbose=True` or `verbose=False` (default) are meaningful; it determines whether the provided transformations are checked to be indeed the inverse coordinate transformations

EXAMPLES:

From spherical coordinates to Cartesian ones in the plane:

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

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

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

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

Bases: *`sage.manifolds.chart.Chart`*

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

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

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

INPUT:

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

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

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

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

EXAMPLES:

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

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

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

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

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

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

The coordinates are then immediately accessible:

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

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

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

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

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

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

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

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

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

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

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

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

Coordinate are also accessible by their indices:

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

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

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

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

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

The coordinate ranges are used for simplifications:

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

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

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

and to the atlas of its domain:

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

Manifold subsets have a *default chart*, which, unless changed via the method `set_default_chart()` , is the first defined chart on the subset (or on 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 φ acting on a point is obtained by means of the call operator, i.e. the operator `()` :

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

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

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

```
sage: c_cartU.valid_coordinates(1,0,2)
False
sage: c_cart.valid_coordinates(1,0,2)
True
```

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

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

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

`add_restrictions` (*restrictions*)

Add some restrictions on the coordinates.

INPUT:

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

A restriction can be any symbolic equality or inequality involving the coordinates, such as `x > y` or `x^2 + y^2 != 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]`.

EXAMPLES:

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

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

The restrictions are transmitted to subcharts:

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

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

```
sage: U = M.open_subset('U')
sage: X_U = X.restrict(U)
sage: X_U.coord_range()
x: (-oo, +oo); y: (-oo, +oo)
```

```
sage: X_U.add_restrictions([x<0, y>1/2])
sage: X_U.coord_range()
x: (-oo, 0); y: (1/2, +oo)
```

coord_bounds (*i=None*)

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

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

INPUT:

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

OUTPUT:

- the coordinate bounds as the tuple `((xmin,min_included),(xmax,max_included))` where
  - `xmin` is the coordinate lower bound
  - `min_included` is a boolean, indicating whether the coordinate can take the value `xmin`, i.e. `xmin` is a strict lower bound iff `min_included` is `False`
  - `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()
(((-Infinity, False), (+Infinity, False)), ((0, True), (1, False)))
sage: c_xy.coord_bounds() == (c_xy.coord_bounds(0), c_xy.coord_bounds(1))
True
```

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

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

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

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

coord_range (*xx=None*)

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

INPUT:

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

EXAMPLES:

Ranges of coordinates on a 2-dimensional manifold:

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

The output is LaTeX-formatted for the notebook:

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

plot (*chart=None*, *ambient_coords=None*, *mapping=None*, *fixed_coords=None*, *ranges=None*, *max_range=8*, *nb_values=None*, *steps=None*, *parameters=None*, *color='red'*, *style='-'*, *label_axes=True*, *plot_points=75*, *thickness=1*, ***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`)

- `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`
- `nb_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 `nb_values` is a single integer, it represents the number of constant values for all coordinates; if `nb_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 `nb_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
- `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

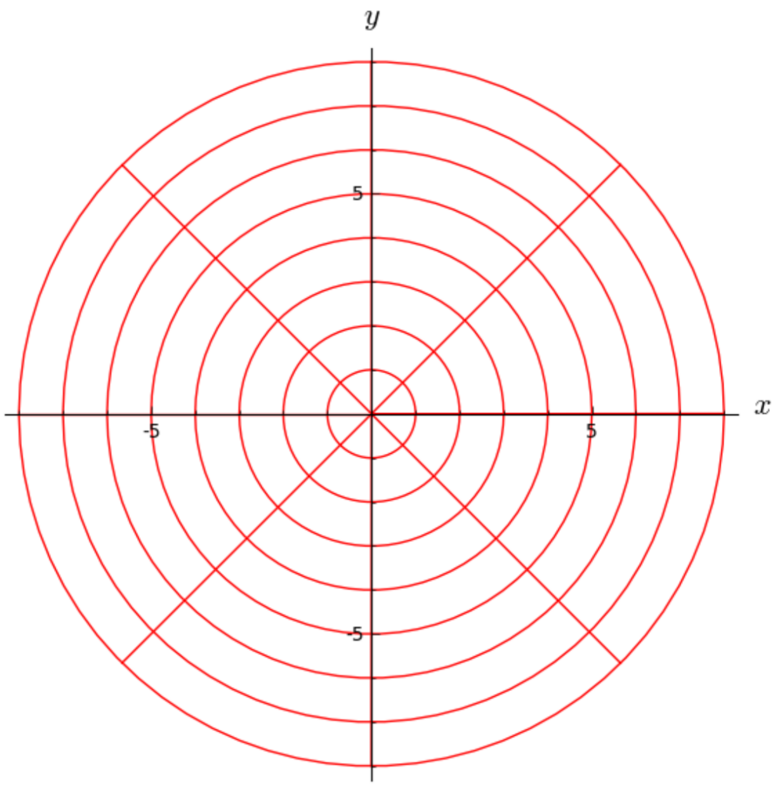
OUTPUT:

- 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:

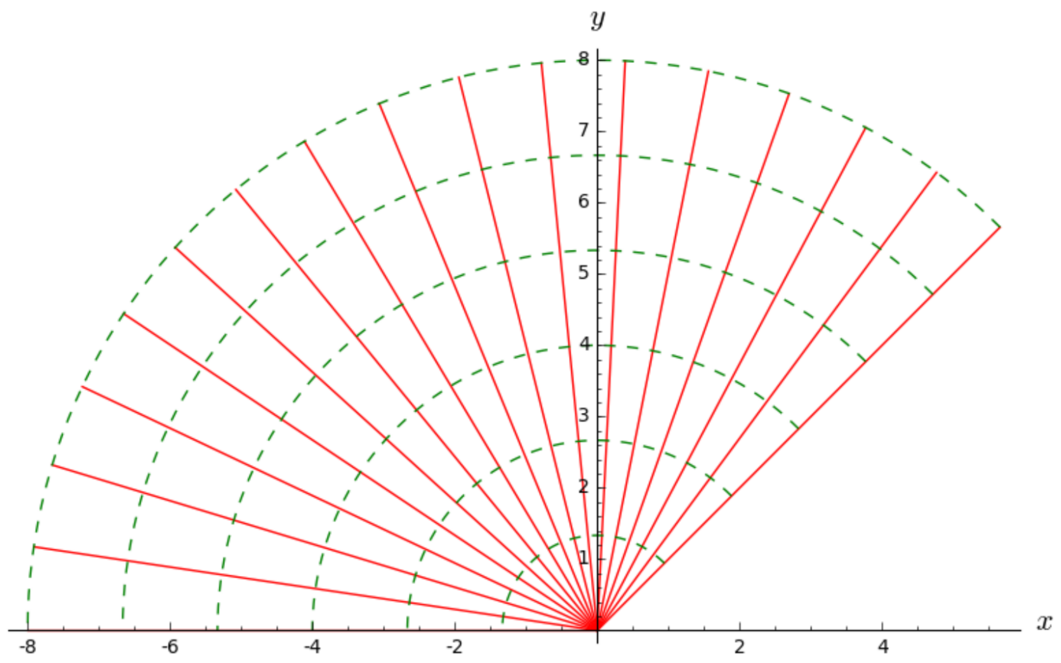
Grid of polar coordinates in terms of Cartesian coordinates in the Euclidean plane:

```
sage: R2 = Manifold(2, 'R^2', structure='topological') # the Euclidean plane
sage: c_cart.<x,y> = R2.chart() # Cartesian coordinates
sage: U = R2.open_subset('U', coord_def={c_cart: (y!=0, x<0)}) # the␣
↪complement of the segment y=0 and x>0
sage: c_pol.<r,ph> = U.chart(r'r:(0,+oo) ph:(0,2*pi):\phi') # polar␣
↪coordinates on U
sage: pol_to_cart = c_pol.transition_map(c_cart, [r*cos(ph), r*sin(ph)])
sage: g = c_pol.plot(c_cart)
sage: g
Graphics object consisting of 18 graphics primitives
```



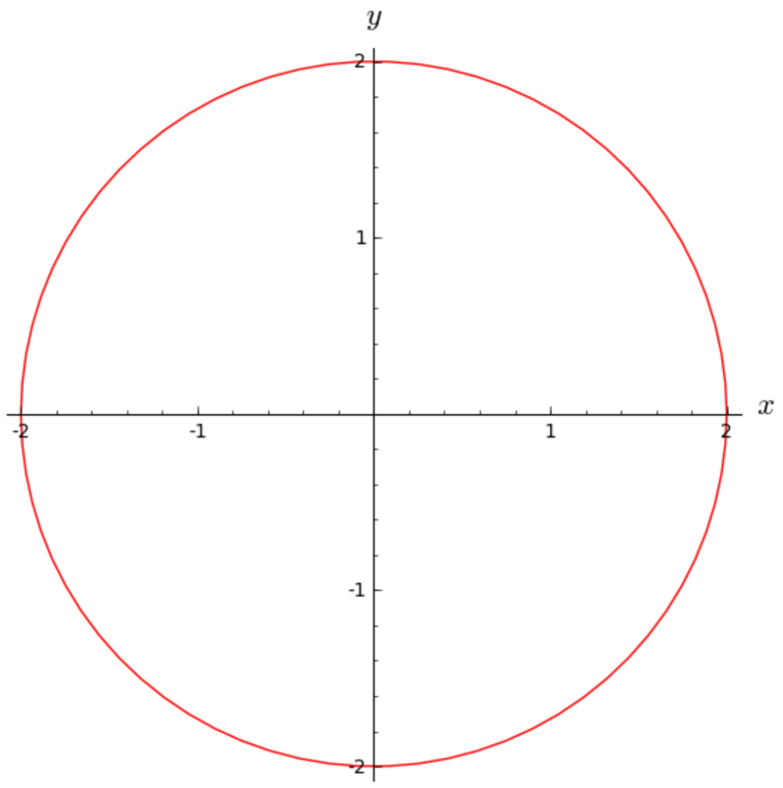
Call with non-default values:

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

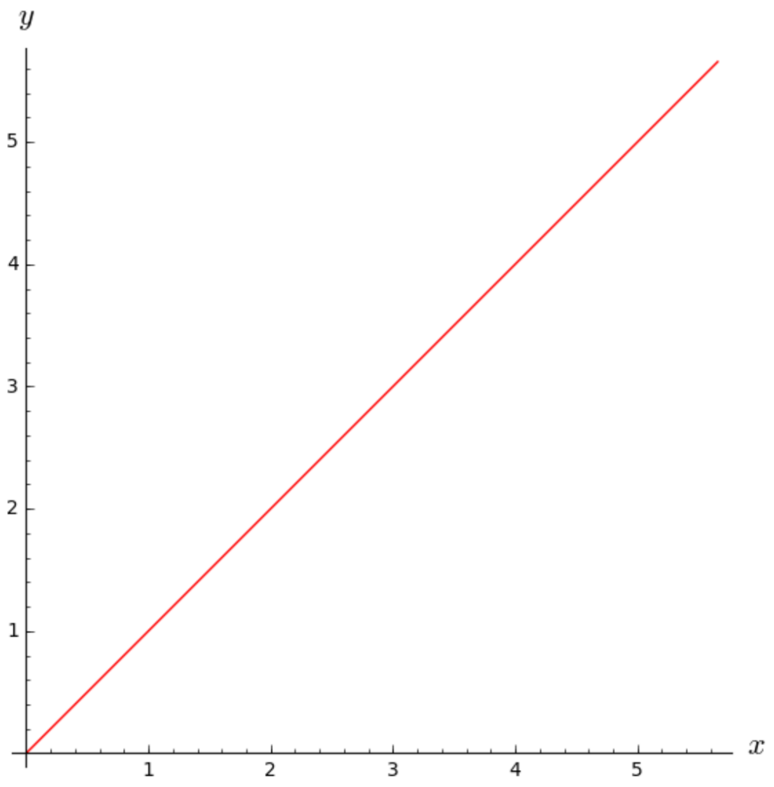


A single coordinate line can be drawn:

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

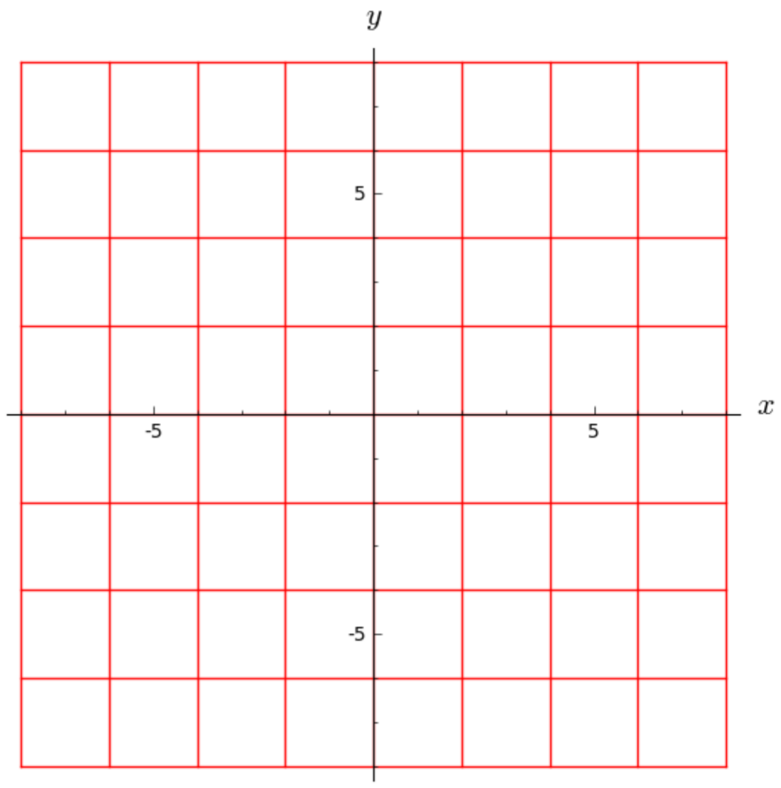


```
sage: g = c_pol.plot(c_cart, fixed_coords={ph: pi/4}) # draw a segment at␣
↪phi=pi/4
```



A chart can be plotted in terms of itself, resulting in a rectangular grid:

```
sage: g = c_cart.plot()  # equivalent to c_cart.plot(c_cart)
sage: g # a rectangular grid
Graphics object consisting of 18 graphics primitives
```



An example with the ambient chart given by the coordinate expression of some manifold map: 3D plot of the stereographic charts on the 2-sphere:

```
sage: S2 = Manifold(2, 'S^2', structure='topological') # the 2-sphere
sage: U = S2.open_subset('U') ; V = S2.open_subset('V') # complement of the␣
↪North and South pole, respectively
sage: S2.declare_union(U,V)
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                   intersection_name='W', restrictions1= x^2+y^2!=0,
....:                   restrictions2= u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: R3 = Manifold(3, 'R^3', structure='topological') # the Euclidean space␣
↪R^3
sage: c_cart.<X,Y,Z> = R3.chart()  # Cartesian coordinates on R^3
sage: Phi = S2.continuous_map(R3, {(c_xy, c_cart): [2*x/(1+x^2+y^2),
....:                           2*y/(1+x^2+y^2), (x^2+y^2-1)/(1+x^2+y^2)],
....:                           (c_uv, c_cart): [2*u/(1+u^2+v^2),
....:                           2*v/(1+u^2+v^2), (1-u^2-v^2)/(1+u^2+v^2)]},
....:                          name='Phi', latex_name=r'\Phi') # Embedding of␣
↪S^2 in R^3
sage: g = c_xy.plot(c_cart, mapping=Phi)
sage: g
Graphics3d Object
sage: type(g)
<class 'sage.plot.plot3d.base.Graphics3dGroup'>
```

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

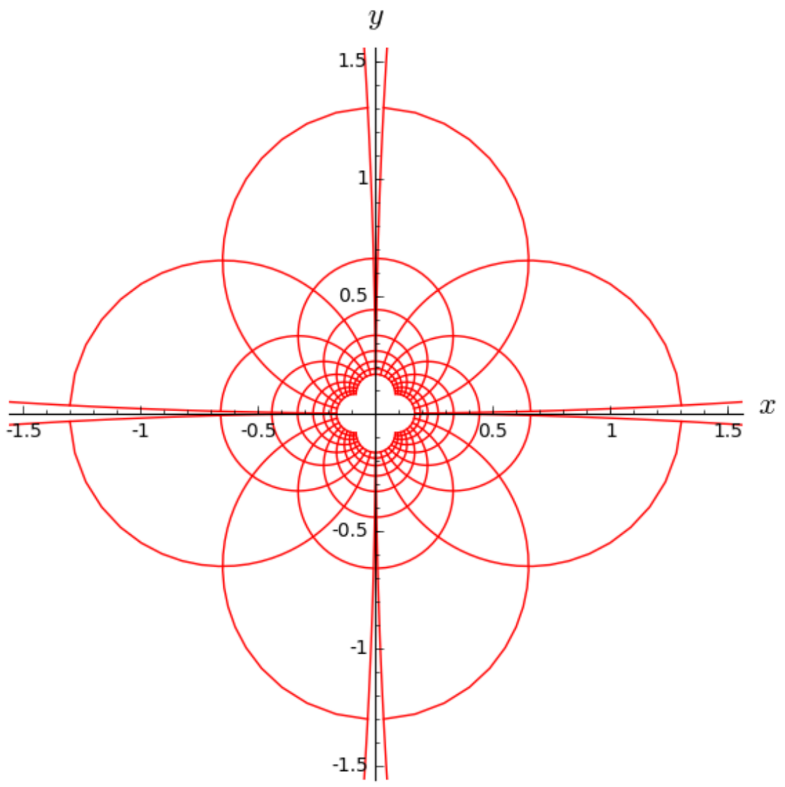
```
sage: g = c_xy.plot(c_cart, mapping=Phi, label_axes=False)
```

The North and South stereographic charts on the same plot:

```
sage: g2 = c_uv.plot(c_cart, mapping=Phi, color='green')
sage: g + g2
Graphics3d Object
```

South stereographic chart drawned in terms of the North one (we split the plot in four parts to avoid the singularity at $(u, v) = (0, 0)$):

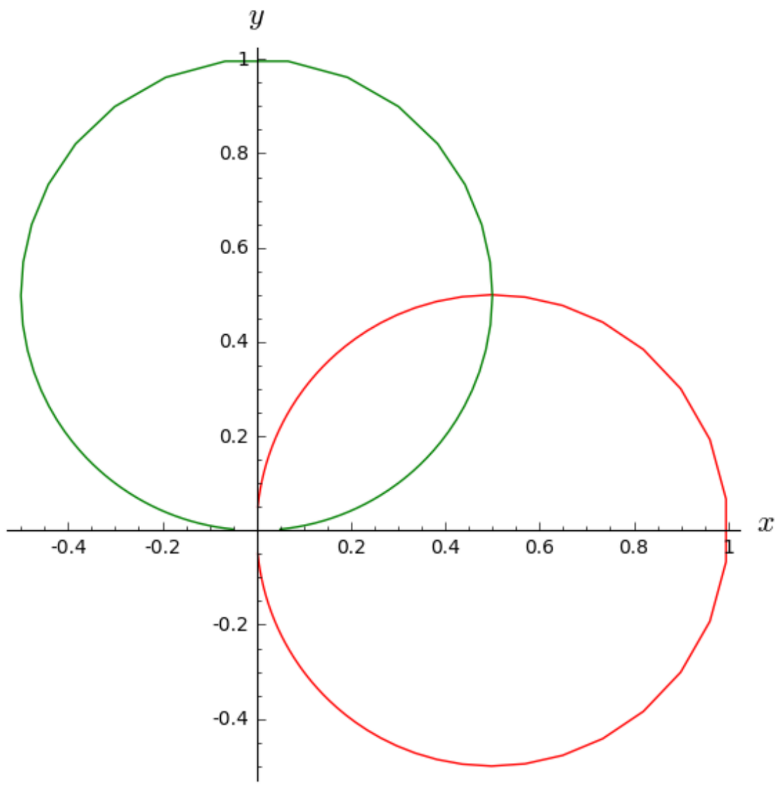
```
sage: W = U.intersection(V) # the subset common to both charts
sage: c_uvW = c_uv.restrict(W) # chart (W,(u,v))
sage: gSN1 = c_uvW.plot(c_xy, ranges={u:[-6.,-0.02], v:[-6.,-0.02]})  # long␣
↪time
sage: gSN2 = c_uvW.plot(c_xy, ranges={u:[-6.,-0.02], v:[0.02,6.]})  # long␣
↪time
sage: gSN3 = c_uvW.plot(c_xy, ranges={u:[0.02,6.], v:[-6.,-0.02]})  # long␣
↪time
sage: gSN4 = c_uvW.plot(c_xy, ranges={u:[0.02,6.], v:[0.02,6.]})  # long time
sage: show(gSN1+gSN2+gSN3+gSN4, xmin=-1.5, xmax=1.5, ymin=-1.5, ymax=1.5)  #␣
↪long time
```



The coordinate line $u = 1$ (red) and the coordinate line $v = 1$ (green) on the same plot:

```
sage: gu1 = c_uvW.plot(c_xy, fixed_coords={u: 1}, max_range=20, plot_
↪points=300)  # long time
sage: gv1 = c_uvW.plot(c_xy, fixed_coords={v: 1}, max_range=20, plot_
↪points=300, color='green')  # long time
```

```
sage: gu1 + gv1  # long time
Graphics object consisting of 2 graphics primitives
```



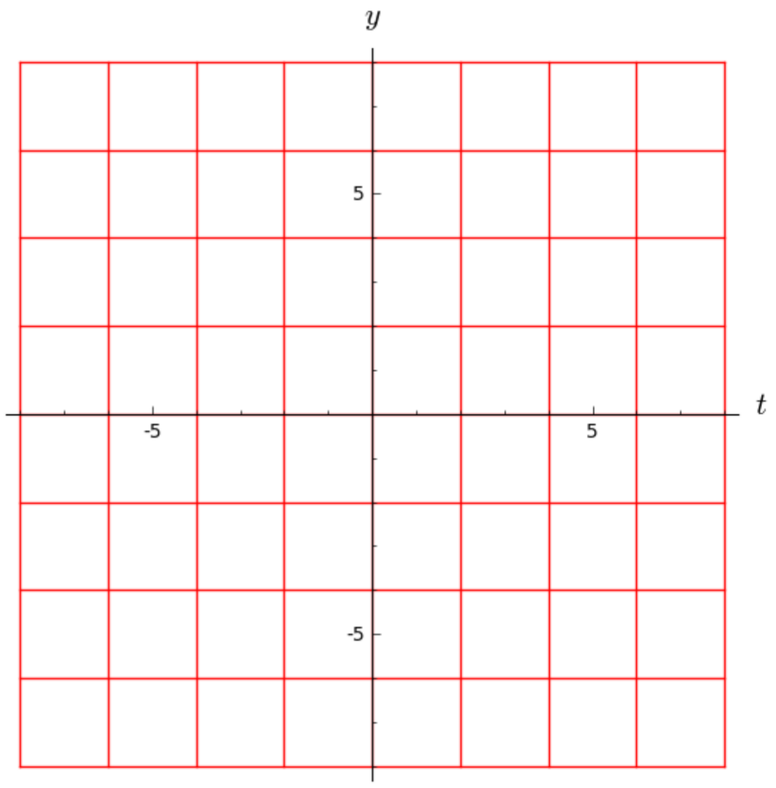
Note that we have set `max_range=20` to have a wider range for the coordinates u and v, i.e. to have $[-20, 20]$ instead of the default $[-8, 8]$.

A 3-dimensional chart plotted in terms of itself results in a 3D rectangular grid:

```
sage: g = c_cart.plot() # equivalent to c_cart.plot(c_cart)  # long time
sage: g  # a 3D mesh cube  # long time
Graphics3d Object
```

A 4-dimensional chart plotted in terms of itself (the plot is performed for at most 3 coordinates, which must be specified via the argument `ambient_coords`):

```
sage: M = Manifold(4, 'M', structure='topological')
sage: X.<t,x,y,z> = M.chart()
sage: g = X.plot(ambient_coords=(t,x,y)) # the coordinate z is not depicted
↪# long time
sage: g  # a 3D mesh cube  # long time
Graphics3d Object
sage: g = X.plot(ambient_coords=(t,y)) # the coordinates x and z are not␣
↪depicted
sage: g  # a 2D mesh square
Graphics object consisting of 18 graphics primitives
```



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

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

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

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

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

INPUT:

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

A restriction can be any symbolic equality or inequality involving the coordinates, such as `x > y` or `x^2 + y^2 != 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:

- the chart (V, ψ) as a *RealChart*

EXAMPLES:

Cartesian coordinates on the unit open disc in $\mathbf{R}^2$ as a subchart of the global Cartesian coordinates:

```
sage: M = Manifold(2, 'R^2', structure='topological')
sage: c_cart.<x,y> = M.chart() # Cartesian coordinates on R^2
sage: D = M.open_subset('D') # the unit open disc
sage: c_cart_D = c_cart.restrict(D, x^2+y^2<1)
sage: p = M.point((1/2, 0))
sage: p in D
True
sage: q = M.point((1, 2))
sage: q in D
False
```

Cartesian coordinates on the annulus $1 < \sqrt{x^2+y^2} < 2$:

```
sage: A = M.open_subset('A')
sage: c_cart_A = c_cart.restrict(A, [x^2+y^2>1, x^2+y^2<4])
sage: p in A, q in A
(False, False)
sage: a = M.point((3/2,0))
sage: a in A
True
```

valid_coordinates (**coordinates*, ***kwds*)

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

INPUT:

- `*coordinates` – coordinate values
- `**kwds` – options:
  - `tolerance=0` , to set the absolute tolerance in the test of coordinate ranges
  - `parameters=None` , to set some numerical values to parameters

OUTPUT:

- `True` if the coordinate values are admissible in the chart range and `False` otherwise

EXAMPLES:

Cartesian coordinates on a square interior:

```
sage: forget()  # for doctest only
sage: M = Manifold(2, 'M', structure='topological')  # the square interior
sage: X.<x,y> = M.chart('x:(-2,2) y:(-2,2)')
sage: X.valid_coordinates(0,1)
True
sage: X.valid_coordinates(-3/2,5/4)
True
sage: X.valid_coordinates(0,3)
False
```

The unit open disk inside the square:

```
sage: D = M.open_subset('D', coord_def={X: x^2+y^2<1})
sage: XD = X.restrict(D)
sage: XD.valid_coordinates(0,1)
False
sage: XD.valid_coordinates(-3/2,5/4)
False
```

```
sage: XD.valid_coordinates(-1/2,1/2)
True
sage: XD.valid_coordinates(0,0)
True
```

1.5.2 Coordinate Functions

In the context of a topological manifold M over a topological field K, a *coordinate function* is a function from a chart codomain to K. In other words, a coordinate function is a K-valued function of the coordinates associated to some chart.

More precisely, let (U, φ) be a chart on M, i.e. U is an open subset of M and $\varphi : U \to V \subset K^n$ is a homeomorphism from U to an open subset V of K^n. A *coordinate function associated to the chart* (U, φ) is a function

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

Coordinate functions are implemented by derived classes of the abstract base class `CoordFunction`.

The class `MultiCoordFunction` implements K^m-valued functions of the coordinates of a chart, with m a positive integer.

AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2013-2015) : initial version
- Travis Scrimshaw (2016) : make `CoordFunction` inheritate from `AlgebraElement`

class `sage.manifolds.coord_func.`**`CoordFunction`**(*parent*)

Bases: `sage.structure.element.AlgebraElement`

Abstract base class for coordinate functions.

If (U, φ) is a chart on a topological manifold M of dimension n over a topological field K, a *coordinate function* associated to (U, φ) is a map $f : V \subset K^n \to K$, where V is the codomain of φ. In other words, f is a K-valued function of the coordinates associated to the chart (U, φ).

The class `CoordFunction` is an abstract one. Specific coordinate functions must be implemented by derived classes, like `CoordFunctionSymb` for symbolic coordinate functions.

INPUT:

- `parent` – the algebra of coordinate functions on a given chart

`arccos`()

Arc cosine of `self`.

OUTPUT:

- coordinate function $\arccos(f)$, where f is the current coordinate function

TESTS:

This method must be implemented by derived classes; it is not implemented here:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: from sage.manifolds.coord_func import CoordFunction
sage: f = CoordFunction(X.function_ring())
sage: f.arccos()
Traceback (most recent call last):
```

```
...
NotImplementedError: <abstract method arccos at 0x...>
```

arccosh ( )

Inverse hyperbolic cosine of `self` .

OUTPUT:

- coordinate function $\operatorname{arcosh}(f)$, where f is the current coordinate function

TESTS:

This method must be implemented by derived classes; it is not implemented here:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: from sage.manifolds.coord_func import CoordFunction
sage: f = CoordFunction(X.function_ring())
sage: f.arccosh()
Traceback (most recent call last):
...
NotImplementedError: <abstract method arccosh at 0x...>
```

arcsin ( )

Arc sine of `self` .

OUTPUT:

- coordinate function $\arcsin(f)$, where f is the current coordinate function

TESTS:

This method must be implemented by derived classes; it is not implemented here:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: from sage.manifolds.coord_func import CoordFunction
sage: f = CoordFunction(X.function_ring())
sage: f.arcsin()
Traceback (most recent call last):
...
NotImplementedError: <abstract method arcsin at 0x...>
```

arcsinh ( )

Inverse hyperbolic sine of `self` .

OUTPUT:

- coordinate function $\operatorname{arsinh}(f)$, where f is the current coordinate function

TESTS:

This method must be implemented by derived classes; it is not implemented here:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: from sage.manifolds.coord_func import CoordFunction
sage: f = CoordFunction(X.function_ring())
sage: f.arcsinh()
Traceback (most recent call last):
...
NotImplementedError: <abstract method arcsinh at 0x...>
```

arctan ( )

Arc tangent of `self` .

OUTPUT:

•coordinate function $\arctan(f)$, where f is the current coordinate function

TESTS:

This method must be implemented by derived classes; it is not implemented here:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: from sage.manifolds.coord_func import CoordFunction
sage: f = CoordFunction(X.function_ring())
sage: f.arctan()
Traceback (most recent call last):
...
NotImplementedError: <abstract method arctan at 0x...>
```

arctanh ( )

Inverse hyperbolic tangent of `self` .

OUTPUT:

•coordinate function $\mathrm{artanh}(f)$, where f is the current coordinate function

TESTS:

This method must be implemented by derived classes; it is not implemented here:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: from sage.manifolds.coord_func import CoordFunction
sage: f = CoordFunction(X.function_ring())
sage: f.arctanh()
Traceback (most recent call last):
...
NotImplementedError: <abstract method arctanh at 0x...>
```

chart ( )

Return the chart with respect to which `self` is defined.

OUTPUT:

•a *`Chart`*

EXAMPLE:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(1+x+y^2)
sage: f.chart()
Chart (M, (x, y))
sage: f.chart() is X
True
```

copy ( )

Return an exact copy of the object.

OUTPUT:

•an instance of *`CoordFunction`*

TESTS:

This method must be implemented by derived classes; it is not implemented here:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: from sage.manifolds.coord_func import CoordFunction
sage: f = CoordFunction(X.function_ring())
sage: f.copy()
Traceback (most recent call last):
...
NotImplementedError: <abstract method copy at 0x...>
```

cos ( )

Cosine of `self`.

OUTPUT:

- coordinate function $\cos(f)$, where f is the current coordinate function

TESTS:

This method must be implemented by derived classes; it is not implemented here:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: from sage.manifolds.coord_func import CoordFunction
sage: f = CoordFunction(X.function_ring())
sage: f.cos()
Traceback (most recent call last):
...
NotImplementedError: <abstract method cos at 0x...>
```

cosh ( )

Hyperbolic cosine of `self`.

OUTPUT:

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

TESTS:

This method must be implemented by derived classes; it is not implemented here:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: from sage.manifolds.coord_func import CoordFunction
sage: f = CoordFunction(X.function_ring())
sage: f.cosh()
Traceback (most recent call last):
...
NotImplementedError: <abstract method cosh at 0x...>
```

diff (*coord*)

Return the partial derivative of `self` with respect to a coordinate.

INPUT:

- `coord` – either the coordinate x^i with respect to which the derivative of the coordinate function f is to be taken, or the index i labelling this coordinate (with the index convention defined on the chart domain via the parameter `start_index`)

OUTPUT:

•instance of `CoordFunction` representing the partial derivative $\frac{\partial f}{\partial x^i}$

TESTS:

This method must be implemented by derived classes; it is not implemented here:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: from sage.manifolds.coord_func import CoordFunction
sage: f = CoordFunction(X.function_ring())
sage: f.diff(x)
Traceback (most recent call last):
...
NotImplementedError: <abstract method diff at 0x...>
```

disp ( )

Display the function in arrow notation.

TESTS:

This method must be implemented by derived classes; it is not implemented here:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: from sage.manifolds.coord_func import CoordFunction
sage: f = CoordFunction(X.function_ring())
sage: f.display()
Traceback (most recent call last):
...
NotImplementedError: <abstract method display at 0x...>
```

display ( )

Display the function in arrow notation.

TESTS:

This method must be implemented by derived classes; it is not implemented here:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: from sage.manifolds.coord_func import CoordFunction
sage: f = CoordFunction(X.function_ring())
sage: f.display()
Traceback (most recent call last):
...
NotImplementedError: <abstract method display at 0x...>
```

exp ( )

Exponential of `self`.

OUTPUT:

•coordinate function $\exp(f)$, where f is the current coordinate function.

TESTS:

This method must be implemented by derived classes; it is not implemented here:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: from sage.manifolds.coord_func import CoordFunction
sage: f = CoordFunction(X.function_ring())
```

```
sage: f.exp()
Traceback (most recent call last):
...
NotImplementedError: <abstract method exp at 0x...>
```

expr ( )

Return some data that, along with the chart, is sufficient to reconstruct the object.

For a symbolic coordinate function, this returns the symbol expression representing the function (see *sage.manifolds.coord_func_symb.CoordFunctionSymb.expr()*)

TESTS:

This method must be implemented by derived classes; it is not implemented here:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: from sage.manifolds.coord_func import CoordFunction
sage: f = CoordFunction(X.function_ring())
sage: f.expr()
Traceback (most recent call last):
...
NotImplementedError: <abstract method expr at 0x...>
```

is_zero ( )

Return `True` if the function is zero and `False` otherwise.

TESTS:

This method must be implemented by derived classes; it is not implemented here:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: from sage.manifolds.coord_func import CoordFunction
sage: f = CoordFunction(X.function_ring())
sage: f.is_zero()
Traceback (most recent call last):
...
NotImplementedError: <abstract method is_zero at 0x...>
```

log (*base=None*)

Logarithm of `self` .

INPUT:

- `base` – (default: `None`) base of the logarithm; if None, the natural logarithm (i.e. logarithm to base e) is returned

OUTPUT:

- coordinate function $\log_a(f)$, where f is the current coordinate function and a is the base

TESTS:

This method must be implemented by derived classes; it is not implemented here:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: from sage.manifolds.coord_func import CoordFunction
sage: f = CoordFunction(X.function_ring())
sage: f.log()
```

```
Traceback (most recent call last):
...
NotImplementedError: <abstract method log at 0x...>
```

scalar_field (*name=None*, *latex_name=None*)

Construct the scalar field that has `self` as coordinate expression.

The domain of the scalar field is the open subset covered by the chart on which `self` is defined.

INPUT:

- `name` – (default: `None`) name given to the scalar field
- `latex_name` – (default: `None`) LaTeX symbol to denote the scalar field; if `None` , the LaTeX symbol is set to `name`

OUTPUT:

- a *ScalarField*

EXAMPLES:

Construction of a scalar field on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: fc = c_xy.function(x+2*y^3)
sage: f = fc.scalar_field() ; f
Scalar field on the 2-dimensional topological manifold M
sage: f.display()
M --> R
(x, y) |--> 2*y^3 + x
sage: f.coord_function(c_xy) is fc
True
```

sin ( )

Sine of `self` .

OUTPUT:

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

TESTS:

This method must be implemented by derived classes; it is not implemented here:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: from sage.manifolds.coord_func import CoordFunction
sage: f = CoordFunction(X.function_ring())
sage: f.sin()
Traceback (most recent call last):
...
NotImplementedError: <abstract method sin at 0x...>
```

sinh ( )

Hyperbolic sine of `self` .

OUTPUT:

- coordinate function $\sinh(f)$, where f is the current coordinate function

TESTS:

This method must be implemented by derived classes; it is not implemented here:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: from sage.manifolds.coord_func import CoordFunction
sage: f = CoordFunction(X.function_ring())
sage: f.sinh()
Traceback (most recent call last):
...
NotImplementedError: <abstract method sinh at 0x...>
```

sqrt ( )

Square root of `self` .

OUTPUT:

- coordinate function $\sqrt{f}$, where f is the current coordinate function

TESTS:

This method must be implemented by derived classes; it is not implemented here:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: from sage.manifolds.coord_func import CoordFunction
sage: f = CoordFunction(X.function_ring())
sage: f.sqrt()
Traceback (most recent call last):
...
NotImplementedError: <abstract method sqrt at 0x...>
```

tan ( )

Tangent of `self` .

OUTPUT:

- coordinate function $\tan(f)$, where f is the current coordinate function

TESTS:

This method must be implemented by derived classes; it is not implemented here:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: from sage.manifolds.coord_func import CoordFunction
sage: f = CoordFunction(X.function_ring())
sage: f.tan()
Traceback (most recent call last):
...
NotImplementedError: <abstract method tan at 0x...>
```

tanh ( )

Hyperbolic tangent of `self` .

OUTPUT:

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

TESTS:

This method must be implemented by derived classes; it is not implemented here:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: from sage.manifolds.coord_func import CoordFunction
sage: f = CoordFunction(X.function_ring())
sage: f.tanh()
Traceback (most recent call last):
...
NotImplementedError: <abstract method tanh at 0x...>
```

class sage.manifolds.coord_func.**MultiCoordFunction** (*chart*, *expressions*)

Bases: sage.structure.sage_object.SageObject

Coordinate function to some Cartesian power of the base field.

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

$$\begin{array}{llcl} f: & V \subset K^n & \longrightarrow & K^m \\ & (x^1, \ldots, x^n) & \longmapsto & (f_1(x^1, \ldots, x^n), \ldots, f_m(x^1, \ldots, x^n)), \end{array}$$

where V is the codomain of φ. In other words, f is a K^m-valued function of the coordinates associated to the chart (U, φ). Each component f_i $(1 \leq i \leq m)$ is a coordinate function and is therefore stored as a CoordFunction.

INPUT:

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

EXAMPLES:

A function $f : V \subset \mathbf{R}^2 \longrightarrow \mathbf{R}^3$:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.multifunction(x-y, x*y, cos(x)*exp(y)); f
Coordinate functions (x - y, x*y, cos(x)*e^y) on the Chart (M, (x, y))
sage: type(f)
<class 'sage.manifolds.coord_func.MultiCoordFunction'>
sage: f(x,y)
(x - y, x*y, cos(x)*e^y)
sage: latex(f)
\left(x - y, x y, \cos\left(x\right) e^{y}\right)
```

Each real-valued function f_i $(1 \leq i \leq m)$ composing f can be accessed via the square-bracket operator, by providing $i - 1$ as an argument:

```
sage: f[0]
x - y
sage: f[1]
x*y
sage: f[2]
cos(x)*e^y
```

We can give a more verbose explanation of each function:

```
sage: f[0].display()
(x, y) |--> x - y
```

Each `f[i-1]` is an instance of *`CoordFunction`*:

```
sage: isinstance(f[0], sage.manifolds.coord_func.CoordFunction)
True
```

In the present case, `f[i-1]` is an instance of the subclass *`CoordFunctionSymb`*:

```
sage: type(f[0])
<class 'sage.manifolds.coord_func_symb.CoordFunctionSymbRing_with_category.
↪element_class'>
```

A class *`MultiCoordFunction`* can represent a real-valued function (case $m = 1$), although one should rather employ the class *`CoordFunction`* for this purpose:

```
sage: g = X.multifunction(x*y^2)
sage: g(x,y)
(x*y^2,)
```

Evaluating the functions at specified coordinates:

```
sage: f(1,2)
(-1, 2, cos(1)*e^2)
sage: var('a b')
(a, b)
sage: f(a,b)
(a - b, a*b, cos(a)*e^b)
sage: g(1,2)
(4,)
```

`chart` ( )

Return the chart with respect to which `self` is defined.

OUTPUT:

- a *`Chart`*

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.multifunction(x-y, x*y, cos(x)*exp(y))
sage: f.chart()
Chart (M, (x, y))
sage: f.chart() is X
True
```

`expr` ( )

Return a tuple of data, the item no.‘i‘ begin sufficient to reconstruct the coordinate function no. i.

In other words, if `f` is a multi-coordinate function, then `f.chart().multifunction(*(f.expr()))` results in a multi-coordinate function identical to `f`.

For a symbolic multi-coordinate function, *`expr()`* returns the tuple of the symbolic expressions of the coordinate functions composing the object.

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.multifunction(x-y, x*y, cos(x)*exp(y))
sage: f.expr()
(x - y, x*y, cos(x)*e^y)
sage: type(f.expr()[0])
<type 'sage.symbolic.expression.Expression'>
```

One shall always have:

```
sage: f.chart().multifunction(*(f.expr())) == f
True
```

jacobian ()

Return the Jacobian matrix of the system of coordinate functions.

`jacobian()` is a 2-dimensional array of size $m \times n$, where m is the number of functions and n the number of coordinates, the generic element being $J_{ij} = \frac{\partial f_i}{\partial x^j}$ with $1 \leq i \leq m$ (row index) and $1 \leq j \leq n$ (column index).

OUTPUT:

- Jacobian matrix as a 2-dimensional array `J` of coordinate functions with `J[i-1][j-1]` being $J_{ij} = \frac{\partial f_i}{\partial x^j}$ for $1 \leq i \leq m$ and $1 \leq j \leq n$

EXAMPLES:

Jacobian of a set of 3 functions of 2 coordinates:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.multifunction(x-y, x*y, y^3*cos(x))
sage: f.jacobian()
[           1           -1]
[           y            x]
[ -y^3*sin(x)  3*y^2*cos(x)]
```

Each element of the result is a *coordinate function*:

```
sage: type(f.jacobian()[2,0])
<class 'sage.manifolds.coord_func_symb.CoordFunctionSymbRing_with_category.
↪element_class'>
sage: f.jacobian()[2,0].display()
(x, y) |--> -y^3*sin(x)
```

Test of the computation:

```
sage: [[f.jacobian()[i,j] == f[i].diff(j) for j in range(2)] for i in␣
↪range(3)]
[[True, True], [True, True], [True, True]]
```

Test with `start_index = 1`:

```
sage: M = Manifold(2, 'M', structure='topological', start_index=1)
sage: X.<x,y> = M.chart()
sage: f = X.multifunction(x-y, x*y, y^3*cos(x))
sage: f.jacobian()
[           1           -1]
[           y            x]
[ -y^3*sin(x)  3*y^2*cos(x)]
sage: [[f.jacobian()[i,j] == f[i].diff(j+1) for j in range(2)]  # note the j+1
....:                                          for i in range(3)]
[[True, True], [True, True], [True, True]]
```

jacobian_det ( )

Return the Jacobian determinant of the system of functions.

The number m of coordinate functions must equal the number n of coordinates.

OUTPUT:

- a `CoordFunction` representing the determinant

EXAMPLES:

Jacobian determinant of a set of 2 functions of 2 coordinates:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.multifunction(x-y, x*y)
sage: f.jacobian_det()
x + y
```

The output of `jacobian_det()` is an instance of `CoordFunction` and can therefore be called on specific values of the coordinates, e.g. $(x, y) = (1, 2)$:

```
sage: type(f.jacobian_det())
<class 'sage.manifolds.coord_func_symb.CoordFunctionSymbRing_with_category.
↪element_class'>
sage: f.jacobian_det().display()
(x, y) |--> x + y
sage: f.jacobian_det()(1,2)
3
```

The result is cached:

```
sage: f.jacobian_det() is f.jacobian_det()
True
```

We verify the determinant of the Jacobian:

```
sage: f.jacobian_det() == det(matrix([[f[i].diff(j).expr() for j in range(2)]
....:                                   for i in range(2)]))
True
```

Jacobian determinant of a set of 3 functions of 3 coordinates:

```
sage: M = Manifold(3, 'M', structure='topological')
sage: X.<x,y,z> = M.chart()
sage: f = X.multifunction(x*y+z^2, z^2*x+y^2*z, (x*y*z)^3)
sage: f.jacobian_det().display()
(x, y, z) |--> 6*x^3*y^5*z^3 - 3*x^4*y^3*z^4 - 12*x^2*y^4*z^5 + 6*x^3*y^2*z^6
```

We verify the determinant of the Jacobian:

```
sage: f.jacobian_det() == det(matrix([[f[i].diff(j).expr() for j in range(3)]
....:                                   for i in range(3)]))
True
```

1.5.3 Symbolic Coordinate Functions

In the context of a topological manifold M over a topological field K, a *coordinate function* is a function from a chart codomain to K. In other words, a coordinate function is a K-valued function of the coordinates associated to some chart.

More precisely, let (U, φ) be a chart on M, i.e. U is an open subset of M and $\varphi : U \to V \subset K^n$ is a homeomorphism from U to an open subset V of K^n. A *coordinate function associated to the chart* (U, φ) is a function

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

This module implements symbolic coordinate functions via the class `CoordFunctionSymb` .

AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2013-2015) : initial version
- Travis Scrimshaw (2016) : make coordinate functions elements of `CoordFunctionSymbRing` .

class `sage.manifolds.coord_func_symb.` **`CoordFunctionSymb`** (*parent*, *expression*)

Bases: *`sage.manifolds.coord_func.CoordFunction`*

Coordinate function with symbolic representation.

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

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

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

INPUT:

- `parent` – the algebra of coordinate functions on the chart (U, φ)
- `expression` – a symbolic expression representing $f(x^1, \ldots, x^n)$, where $(x^1, \ldots, x^n)$ are the coordinates of the chart (U, φ)

EXAMPLES:

A symbolic coordinate function associated with a 2-dimensional chart:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x^2+3*y+1)
sage: type(f)
<class 'sage.manifolds.coord_func_symb.CoordFunctionSymbRing_with_category.
↪element_class'>
sage: f.display()
(x, y) |--> x^2 + 3*y + 1
sage: f(x,y)
x^2 + 3*y + 1
```

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

```
sage: f
x^2 + 3*y + 1
sage: latex(f)
x^{2} + 3 \, y + 1
```

A similar output is obtained by means of the method *expr()* :

```
sage: f.expr()
x^2 + 3*y + 1
```

The value of the function at specified coordinates is obtained by means of the standard parentheses notation:

```
sage: f(2,-1)
2
sage: var('a b')
(a, b)
sage: f(a,b)
a^2 + 3*b + 1
```

An unspecified coordinate function:

```
sage: g = X.function(function('G')(x, y))
sage: g
G(x, y)
sage: g.display()
(x, y) |--> G(x, y)
sage: g.expr()
G(x, y)
sage: g(2,3)
G(2, 3)
```

Coordinate functions can be compared to other values:

```
sage: f = X.function(x^2+3*y+1)
sage: f == 2
False
sage: f == x^2 + 3*y + 1
True
sage: g = X.function(x*y)
sage: f == g
False
sage: h = X.function(x^2+3*y+1)
sage: f == h
True
```

Differences between `CoordFunctionSymb` and callable symbolic expressions

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

```
sage: f0(x,y) = x^2 + 3*y + 1
sage: type(f0)
<type 'sage.symbolic.expression.Expression'>
sage: f0
(x, y) |--> x^2 + 3*y + 1
sage: f0(x,y)
x^2 + 3*y + 1
```

To get an output similar to that of `f0` for the coordinate function `f`, we must use the method *display()* :

```
sage: f
x^2 + 3*y + 1
sage: f.display()
```

```
(x, y) |--> x^2 + 3*y + 1
sage: f(x,y)
x^2 + 3*y + 1
```

More importantly, instances of *CoordFunctionSymb* differ from callable symbolic expression by the automatic simplifications in all operations. For instance, adding the two callable symbolic expressions:

```
sage: f0(x,y,z) = cos(x)^2 ; g0(x,y,z) = sin(x)^2
```

results in:

```
sage: f0 + g0
(x, y, z) |--> cos(x)^2 + sin(x)^2
```

To get 1, one has to call `simplify_trig()` :

```
sage: (f0 + g0).simplify_trig()
(x, y, z) |--> 1
```

On the contrary, the sum of the corresponding *CoordFunctionSymb* instances is automatically simplified (see *simplify_chain_real()* and *simplify_chain_generic()* for details):

```
sage: f = X.function(cos(x)^2) ; g = X.function(sin(x)^2)
sage: f + g
1
```

Another difference regards the display of partial derivatives: for callable symbolic functions, it relies on Pynac notation `D[0]` , `D[1]` , etc.:

```
sage: g = function('g')(x, y)
sage: f0(x,y) = diff(g, x) + diff(g, y)
sage: f0
(x, y) |--> D[0](g)(x, y) + D[1](g)(x, y)
```

while for coordinate functions, the display is more "textbook" like:

```
sage: f = X.function(diff(g, x) + diff(g, y))
sage: f
d(g)/dx + d(g)/dy
```

The difference is even more dramatic on LaTeX outputs:

```
sage: latex(f0)
\left( x, y \right) \ {\mapsto} \ D[0]\left(g\right)\left(x, y\right)
 + D[1]\left(g\right)\left(x, y\right)
sage: latex(f)
\frac{\partial\,g}{\partial x} + \frac{\partial\,g}{\partial y}
```

Note that this regards only the display of coordinate functions: internally, the Pynac notation is still used, as we can check by asking for the symbolic expression stored in `f` :

```
sage: f.expr()
D[0](g)(x, y) + D[1](g)(x, y)
```

One can switch to Pynac notation by changing the options:

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

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

```
sage: f = X.function(function('u')(x, y) * function('v')(x, y))
sage: f
u(x, y)*v(x, y)
sage: f0(x,y) = function('u')(x, y) * function('v')(x, y)
sage: f0
(x, y) |--> u(x, y)*v(x, y)
```

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

```
sage: Manifold.options.omit_function_arguments=True
sage: f
u*v
```

Note that neither the callable symbolic expression `f0` nor the internal expression of `f` is affected by the above command:

```
sage: f0
(x, y) |--> u(x, y)*v(x, y)
sage: f.expr()
u(x, y)*v(x, y)
```

We revert to the default behavior by:

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

arccos ( )

Arc cosine of `self`.

OUTPUT:

- coordinate function $\arccos(f)$, where f is the current coordinate function

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x*y)
sage: f.arccos()
arccos(x*y)
sage: arccos(f)  # equivalent to f.arccos()
arccos(x*y)
sage: acos(f)  # equivalent to f.arccos()
arccos(x*y)
sage: arccos(f).display()
(x, y) |--> arccos(x*y)
```

```
sage: arccos(X.zero_function()).display()
(x, y) |--> 1/2*pi
```

arccosh ( )

Inverse hyperbolic cosine of `self` .

OUTPUT:

- coordinate function $\operatorname{arcosh}(f)$, where f is the current coordinate function

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x*y)
sage: f.arccosh()
arccosh(x*y)
sage: arccosh(f)  # equivalent to f.arccosh()
arccosh(x*y)
sage: acosh(f)  # equivalent to f.arccosh()
arccosh(x*y)
sage: arccosh(f).display()
(x, y) |--> arccosh(x*y)
sage: arccosh(X.function(1)) == X.zero_function()
True
```

arcsin ( )

Arc sine of `self` .

OUTPUT:

- coordinate function $\arcsin(f)$, where f is the current coordinate function

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x*y)
sage: f.arcsin()
arcsin(x*y)
sage: arcsin(f)  # equivalent to f.arcsin()
arcsin(x*y)
sage: asin(f)  # equivalent to f.arcsin()
arcsin(x*y)
sage: arcsin(f).display()
(x, y) |--> arcsin(x*y)
sage: arcsin(X.zero_function()) == X.zero_function()
True
```

arcsinh ( )

Inverse hyperbolic sine of `self` .

OUTPUT:

- coordinate function $\operatorname{arsinh}(f)$, where f is the current coordinate function

EXAMPLES:

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

```
sage: f.arcsinh()
arcsinh(x*y)
sage: arcsinh(f)  # equivalent to f.arcsinh()
arcsinh(x*y)
sage: asinh(f)  # equivalent to f.arcsinh()
arcsinh(x*y)
sage: arcsinh(f).display()
(x, y) |--> arcsinh(x*y)
sage: arcsinh(X.zero_function()) == X.zero_function()
True
```

arctan ( )

Arc tangent of `self`.

OUTPUT:

•coordinate function $\arctan(f)$, where f is the current coordinate function

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x*y)
sage: f.arctan()
arctan(x*y)
sage: arctan(f)  # equivalent to f.arctan()
arctan(x*y)
sage: atan(f)  # equivalent to f.arctan()
arctan(x*y)
sage: arctan(f).display()
(x, y) |--> arctan(x*y)
sage: arctan(X.zero_function()) == X.zero_function()
True
```

arctanh ( )

Inverse hyperbolic tangent of `self`.

OUTPUT:

•coordinate function $\operatorname{artanh}(f)$, where f is the current coordinate function

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x*y)
sage: f.arctanh()
arctanh(x*y)
sage: arctanh(f)  # equivalent to f.arctanh()
arctanh(x*y)
sage: atanh(f)  # equivalent to f.arctanh()
arctanh(x*y)
sage: arctanh(f).display()
(x, y) |--> arctanh(x*y)
sage: arctanh(X.zero_function()) == X.zero_function()
True
```

collect (*s*)

Collect the coefficients of s in the expression of `self` into a group.

INPUT:

- `s` – the symbol whose coefficients will be collected

OUTPUT:

- `self` with the coefficients of `s` grouped in its expression

EXAMPLES:

Action on a 2-dimensional coordinate function:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x^2*y + x*y + (x*y)^2)
sage: f.display()
(x, y) |--> x^2*y^2 + x^2*y + x*y
sage: f.collect(y)
x^2*y^2 + (x^2 + x)*y
```

The method `collect()` has changed the expression of `f`:

```
sage: f.display()
(x, y) |--> x^2*y^2 + (x^2 + x)*y
```

`collect_common_factors` ( )

Collect common factors in the expression of `self`.

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

OUTPUT:

- `self` with the common factors collected in its expression

EXAMPLES:

Action on a 2-dimensional coordinate function:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x/(x^2*y + x*y))
sage: f.display()
(x, y) |--> x/(x^2*y + x*y)
sage: f.collect_common_factors()
1/((x + 1)*y)
```

The method `collect_common_factors()` has changed the expression of `f`:

```
sage: f.display()
(x, y) |--> 1/((x + 1)*y)
```

`copy` ( )

Return an exact copy of the object.

OUTPUT:

- a *`CoordFunctionSymb`*

EXAMPLES:

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

By construction, `g` is identical to `f` :

```
sage: type(g) == type(f)
True
sage: g == f
True
```

but it is not the same object:

```
sage: g is f
False
```

cos ( )

Cosine of `self` .

OUTPUT:

- coordinate function $\cos(f)$, where f is the current coordinate function

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x*y)
sage: f.cos()
cos(x*y)
sage: cos(f)  # equivalent to f.cos()
cos(x*y)
sage: cos(f).display()
(x, y) |--> cos(x*y)
sage: cos(X.zero_function()).display()
(x, y) |--> 1
```

cosh ( )

Hyperbolic cosine of `self` .

OUTPUT:

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

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x*y)
sage: f.cosh()
cosh(x*y)
sage: cosh(f)  # equivalent to f.cosh()
cosh(x*y)
sage: cosh(f).display()
(x, y) |--> cosh(x*y)
sage: cosh(X.zero_function()).display()
(x, y) |--> 1
```

diff (*coord*)
Partial derivative with respect to a coordinate.

INPUT:

- `coord` – either the coordinate x^i with respect to which the derivative of the coordinate function f is to be taken, or the index i labelling this coordinate (with the index convention defined on the chart domain via the parameter `start_index`)

OUTPUT:

- a *CoordFunctionSymb* representing the partial derivative $\frac{\partial f}{\partial x^i}$

EXAMPLES:

Partial derivatives of a 2-dimensional coordinate function:

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

Each partial derivatives is itself a coordinate function:

```
sage: type(f.diff(x))
<class 'sage.manifolds.coord_func_symb.CoordFunctionSymbRing_with_category.
↪element_class'>
```

An index can be used instead of the coordinate symbol:

```
sage: f.diff(0)
2*x
sage: f.diff(1)
3
```

The index range depends on the convention used on the chart's domain:

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

disp ( )
Display `self` in arrow notation.

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

EXAMPLES:

Coordinate function on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(cos(x*y/2))
sage: f.display()
(x, y) |--> cos(1/2*x*y)
sage: latex(f.display())
\left(x, y\right) \mapsto \cos\left(\frac{1}{2} \, x y\right)
```

A shortcut is `disp()` :

```
sage: f.disp()
(x, y) |--> cos(1/2*x*y)
```

Display of the zero function:

```
sage: X.zero_function().display()
(x, y) |--> 0
```

display ( )

Display `self` in arrow notation.

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

EXAMPLES:

Coordinate function on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(cos(x*y/2))
sage: f.display()
(x, y) |--> cos(1/2*x*y)
sage: latex(f.display())
\left(x, y\right) \mapsto \cos\left(\frac{1}{2} \, x y\right)
```

A shortcut is `disp()` :

```
sage: f.disp()
(x, y) |--> cos(1/2*x*y)
```

Display of the zero function:

```
sage: X.zero_function().display()
(x, y) |--> 0
```

exp ( )

Exponential of `self` .

OUTPUT:

- coordinate function $\exp(f)$, where f is the current coordinate function

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x+y)
sage: f.exp()
e^(x + y)
sage: exp(f) # equivalent to f.exp()
```

```
e^(x + y)
sage: exp(f).display()
(x, y) |--> e^(x + y)
sage: exp(X.zero_function())
1
```

expand ( )

Expand the coordinate expression of `self` .

OUTPUT:

•`self` with its expression expanded

EXAMPLES:

Expanding a 2-dimensional coordinate function:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function((x - y)^2)
sage: f.display()
(x, y) |--> (x - y)^2
sage: f.expand()
x^2 - 2*x*y + y^2
```

The method `expand()` has changed the expression of `f` :

```
sage: f.display()
(x, y) |--> x^2 - 2*x*y + y^2
```

expr ( )

Return the symbolic expression representing the image of `self` .

OUTPUT:

•`symbolic expression` involving the chart coordinates

EXAMPLES:

Coordinate function of a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x^2+3*y+1)
sage: f.expr()
x^2 + 3*y + 1
sage: type(f.expr())
<type 'sage.symbolic.expression.Expression'>
```

For a symbolic coordinate function, one shall always have:

```
sage: bool( f.expr() == f(*(f.chart()[:])) )
True
```

The method *`expr()`* is useful for accessing to all the symbolic expression functionalities in Sage; for instance:

```
sage: var('a')
a
sage: f = X.function(a*x*y); f.display()
```

```
(x, y) |--> a*x*y
sage: f.expr()
a*x*y
sage: f.expr().subs(a=2)
2*x*y
```

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

```
sage: f(x,3)
3*a*x
sage: bool( f(x,3) == f.expr().subs(y=3) )
True
```

factor ( )

Factorize the coordinate expression of `self` .

OUTPUT:

- `self` with its expression factorized

EXAMPLES:

Factorization of a 2-dimensional coordinate function:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x^2 + 2*x*y + y^2)
sage: f.display()
(x, y) |--> x^2 + 2*x*y + y^2
sage: f.factor()
(x + y)^2
```

The method `factor()` has changed the expression of `f` :

```
sage: f.display()
(x, y) |--> (x + y)^2
```

is_zero ( )

Return `True` if the function is zero and `False` otherwise.

EXAMPLES:

Coordinate functions associated to a 2-dimensional chart:

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

log (*base=None*)

Logarithm of `self` .

INPUT:

•`base` – (default: `None`) base of the logarithm; if `None` , the natural logarithm (i.e. logarithm to base e) is returned

OUTPUT:

•coordinate function $\log_a(f)$, where f is the current coordinate function and a is the base

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x+y)
sage: f.log()
log(x + y)
sage: log(f) # equivalent to f.log()
log(x + y)
sage: log(f).display()
(x, y) |--> log(x + y)
sage: f.log(2)
log(x + y)/log(2)
sage: log(f, 2)
log(x + y)/log(2)
```

simplify ( )

Simplify the coordinate expression of `self` .

For details about the employed chain of simplifications, see *`simplify_chain_real()`* for coordinate functions on real manifolds and *`simplify_chain_generic()`* for the generic case.

OUTPUT:

•`self` with its expression simplified

EXAMPLES:

Simplification of a 2-dimension coordinate function:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(cos(x)^2+sin(x)^2 + sqrt(x^2))
sage: f.display()
(x, y) |--> cos(x)^2 + sin(x)^2 + sqrt(x^2)
sage: f.simplify()
abs(x) + 1
```

The method `simplify()` has changed the expression of `f` :

```
sage: f.display()
(x, y) |--> abs(x) + 1
```

Another example:

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

Examples taking into account the declared range of a coordinate:

```
sage: M =  Manifold(2, 'M_1', structure='topological')
sage: X.<x,y> = M.chart('x:(0,+oo) y')
sage: f = X.function(sqrt(x^2))
sage: f
x
sage: f.simplify()
x
```

```
sage: forget()  # to clear the previous assumption on x
sage: M =  Manifold(2, 'M_2', structure='topological')
sage: X.<x,y> = M.chart('x:(-oo,0) y')
sage: f = X.function(sqrt(x^2))
sage: f
sqrt(x^2)
sage: f.simplify()
-x
```

sin ( )

Sine of `self` .

OUTPUT:

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

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x*y)
sage: f.sin()
sin(x*y)
sage: sin(f)  # equivalent to f.sin()
sin(x*y)
sage: sin(f).display()
(x, y) |--> sin(x*y)
sage: sin(X.zero_function()) == X.zero_function()
True
```

sinh ( )

Hyperbolic sine of `self` .

OUTPUT:

- coordinate function $\sinh(f)$, where f is the current coordinate function

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x*y)
sage: f.sinh()
sinh(x*y)
sage: sinh(f)  # equivalent to f.sinh()
sinh(x*y)
sage: sinh(f).display()
(x, y) |--> sinh(x*y)
sage: sinh(X.zero_function()) == X.zero_function()
True
```

sqrt ()

Square root of `self`.

OUTPUT:

- coordinate function $\sqrt{f}$, where f is the current coordinate function

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x+y)
sage: f.sqrt()
sqrt(x + y)
sage: sqrt(f)  # equivalent to f.sqrt()
sqrt(x + y)
sage: sqrt(f).display()
(x, y) |--> sqrt(x + y)
sage: sqrt(X.zero_function()).display()
(x, y) |--> 0
```

tan ()

Tangent of `self`.

OUTPUT:

- coordinate function $\tan(f)$, where f is the current coordinate function

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x*y)
sage: f.tan()
sin(x*y)/cos(x*y)
sage: tan(f)  # equivalent to f.tan()
sin(x*y)/cos(x*y)
sage: tan(f).display()
(x, y) |--> sin(x*y)/cos(x*y)
sage: tan(X.zero_function()) == X.zero_function()
True
```

tanh ()

Hyperbolic tangent of `self`.

OUTPUT:

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

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = X.function(x*y)
sage: f.tanh()
sinh(x*y)/cosh(x*y)
sage: tanh(f)  # equivalent to f.tanh()
sinh(x*y)/cosh(x*y)
sage: tanh(f).display()
(x, y) |--> sinh(x*y)/cosh(x*y)
sage: tanh(X.zero_function()) == X.zero_function()
True
```

class `sage.manifolds.coord_func_symb.`**`CoordFunctionSymbRing`**`(`*chart*`)`

Bases: `sage.structure.parent.Parent`, `sage.structure.unique_representation.UniqueRepresent`

Ring of all symbolic coordinate functions on a chart.

INPUT:

- `chart` – a coordinate chart, as an instance of class `Chart`

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: FR = X.function_ring(); FR
Ring of coordinate functions on Chart (M, (x, y))
sage: type(FR)
<class 'sage.manifolds.coord_func_symb.CoordFunctionSymbRing_with_category'>
sage: FR.category()
Category of commutative algebras over Symbolic Ring
```

`Element`

alias of `CoordFunctionSymb`

`from_base_ring``(`*r*`)`

Return the canonical embedding of `r` into `self`.

INPUT:

- `r` – an element of `self.base_ring()`

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: FR = X.function_ring()
sage: f = FR.from_base_ring(x*y)
sage: f.display()
(x, y) |--> x*y
```

`is_field``()`

Return `False` as `self` is not an integral domain.

EXAMPLES:

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

`is_integral_domain``()`

Return `False` as `self` is not an integral domain.

EXAMPLES:

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

```
sage: FR = X.function_ring()
sage: FR.is_integral_domain()
False
sage: FR.is_field()
False
```

one ( )

Return the constant function 1 in `self` .

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: FR = X.function_ring()
sage: FR.one()
1

sage: M = Manifold(2, 'M', structure='topological', field=Qp(5))
sage: X.<x,y> = M.chart()
sage: X.function_ring().one()
1 + O(5^20)
```

zero ( )

Return the constant function 0 in `self` .

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: FR = X.function_ring()
sage: FR.zero()
0

sage: M = Manifold(2, 'M', structure='topological', field=Qp(5))
sage: X.<x,y> = M.chart()
sage: X.function_ring().zero()
0
```

1.6 Scalar Fields

1.6.1 Algebra of Scalar Fields

The class *ScalarFieldAlgebra* implements the commutative algebra $C^0(M)$ of scalar fields on a topological manifold M over a topological field K. By *scalar field*, it is meant a continuous function $M \to K$. The set $C^0(M)$ is an algebra over K, whose ring product is the pointwise multiplication of K-valued functions, which is clearly commutative.

AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2014-2015): initial version
- Travis Scrimshaw (2016): review tweaks

REFERENCES:

- *[Lee11]* J.M. Lee : *Introduction to Topological Manifolds*, 2nd ed., Springer (New York) (2011)

- *[KN63]* S. Kobayashi & K. Nomizu : *Foundations of Differential Geometry*, vol. 1, Interscience Publishers (New York) (1963)

class `sage.manifolds.scalarfield_algebra.`**`ScalarFieldAlgebra`**`(` *domain* `)`

Bases: `sage.structure.unique_representation.UniqueRepresentation`, `sage.structure.parent.Parent`

Commutative algebra of scalar fields on a topological manifold.

If M is a topological manifold over a topological field K, the commutative algebra of scalar fields on M is the set $C^0(M)$ of all continuous maps $M \to K$. The set $C^0(M)$ is an algebra over K, whose ring product is the pointwise multiplication of K-valued functions, which is clearly commutative.

If $K = \mathbf{R}$ or $K = \mathbf{C}$, the field K over which the algebra $C^0(M)$ is constructed is represented by the `Symbolic Ring SR`, since there is no exact representation of $\mathbf{R}$ nor $\mathbf{C}$.

INPUT:

- `domain` – the topological manifold M on which the scalar fields are defined

EXAMPLES:

Algebras of scalar fields on the sphere S^2 and on some open subsets of it:

```
sage: M = Manifold(2, 'M', structure='topological') # the 2-dimensional sphere S^2
sage: U = M.open_subset('U')   # complement of the North pole
sage: c_xy.<x,y> = U.chart()   # stereographic coordinates from the North pole
sage: V = M.open_subset('V')   # complement of the South pole
sage: c_uv.<u,v> = V.chart()   # stereographic coordinates from the South pole
sage: M.declare_union(U,V)     # S^2 is the union of U and V
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                                intersection_name='W',
....:                                restrictions1= x^2+y^2!=0,
....:                                restrictions2= u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: CM = M.scalar_field_algebra(); CM
Algebra of scalar fields on the 2-dimensional topological manifold M
sage: W = U.intersection(V)  # S^2 minus the two poles
sage: CW = W.scalar_field_algebra(); CW
Algebra of scalar fields on the Open subset W of the
 2-dimensional topological manifold M
```

$C^0(M)$ and $C^0(W)$ belong to the category of commutative algebras over $\mathbf{R}$ (represented here by `SymbolicRing`):

```
sage: CM.category()
Category of commutative algebras over Symbolic Ring
sage: CM.base_ring()
Symbolic Ring
sage: CW.category()
Category of commutative algebras over Symbolic Ring
sage: CW.base_ring()
Symbolic Ring
```

The elements of $C^0(M)$ are scalar fields on M:

```
sage: CM.an_element()
Scalar field on the 2-dimensional topological manifold M
sage: CM.an_element().display()   # this sample element is a constant field
M --> R
```

```
on U: (x, y) |--> 2
on V: (u, v) |--> 2
```

Those of $C^0(W)$ are scalar fields on W:

```
sage: CW.an_element()
Scalar field on the Open subset W of the 2-dimensional topological
 manifold M
sage: CW.an_element().display()  # this sample element is a constant field
W --> R
(x, y) |--> 2
(u, v) |--> 2
```

The zero element:

```
sage: CM.zero()
Scalar field zero on the 2-dimensional topological manifold M
sage: CM.zero().display()
zero: M --> R
on U: (x, y) |--> 0
on V: (u, v) |--> 0
```

```
sage: CW.zero()
Scalar field zero on the Open subset W of the 2-dimensional
 topological manifold M
sage: CW.zero().display()
zero: W --> R
   (x, y) |--> 0
   (u, v) |--> 0
```

The unit element:

```
sage: CM.one()
Scalar field 1 on the 2-dimensional topological manifold M
sage: CM.one().display()
1: M --> R
on U: (x, y) |--> 1
on V: (u, v) |--> 1
```

```
sage: CW.one()
Scalar field 1 on the Open subset W of the 2-dimensional topological
 manifold M
sage: CW.one().display()
1: W --> R
  (x, y) |--> 1
  (u, v) |--> 1
```

A generic element can be constructed by using a dictionary of the coordinate expressions defining the scalar field:

```
sage: f = CM({c_xy: atan(x^2+y^2), c_uv: pi/2 - atan(u^2+v^2)}); f
Scalar field on the 2-dimensional topological manifold M
sage: f.display()
M --> R
on U: (x, y) |--> arctan(x^2 + y^2)
on V: (u, v) |--> 1/2*pi - arctan(u^2 + v^2)
```

```
sage: f.parent()
Algebra of scalar fields on the 2-dimensional topological manifold M
```

Specific elements can also be constructed in this way:

```
sage: CM(0) == CM.zero()
True
sage: CM(1) == CM.one()
True
```

Note that the zero scalar field is cached:

```
sage: CM(0) is CM.zero()
True
```

Elements can also be constructed by means of the method `scalar_field()` acting on the domain (this allows one to set the name of the scalar field at the construction):

```
sage: f1 = M.scalar_field({c_xy: atan(x^2+y^2), c_uv: pi/2 - atan(u^2+v^2)},
....:                     name='f')
sage: f1.parent()
Algebra of scalar fields on the 2-dimensional topological manifold M
sage: f1 == f
True
sage: M.scalar_field(0, chart='all') == CM.zero()
True
```

The algebra $C^0(M)$ coerces to $C^0(W)$ since W is an open subset of M:

```
sage: CW.has_coerce_map_from(CM)
True
```

The reverse is of course false:

```
sage: CM.has_coerce_map_from(CW)
False
```

The coercion map is nothing but the restriction to W of scalar fields on M:

```
sage: fW = CW(f) ; fW
Scalar field on the Open subset W of the
 2-dimensional topological manifold M
sage: fW.display()
W --> R
   (x, y) |--> arctan(x^2 + y^2)
   (u, v) |--> 1/2*pi - arctan(u^2 + v^2)
```

```
sage: CW(CM.one()) == CW.one()
True
```

The coercion map allows for the addition of elements of $C^0(W)$ with elements of $C^0(M)$, the result being an element of $C^0(W)$:

```
sage: s = fW + f
sage: s.parent()
Algebra of scalar fields on the Open subset W of the
 2-dimensional topological manifold M
```

```
sage: s.display()
W --> R
   (x, y) |--> 2*arctan(x^2 + y^2)
   (u, v) |--> pi - 2*arctan(u^2 + v^2)
```

Another coercion is that from the Symbolic Ring. Since the Symbolic Ring is the base ring for the algebra `CM`, the coercion of a symbolic expression `s` is performed by the operation `s*CM.one()`, which invokes the (reflected) multiplication operator. If the symbolic expression does not involve any chart coordinate, the outcome is a constant scalar field:

```
sage: h = CM(pi*sqrt(2)) ; h
Scalar field on the 2-dimensional topological manifold M
sage: h.display()
M --> R
on U: (x, y) |--> sqrt(2)*pi
on V: (u, v) |--> sqrt(2)*pi
sage: a = var('a')
sage: h = CM(a); h.display()
M --> R
on U: (x, y) |--> a
on V: (u, v) |--> a
```

If the symbolic expression involves some coordinate of one of the manifold's charts, the outcome is initialized only on the chart domain:

```
sage: h = CM(a+x); h.display()
M --> R
on U: (x, y) |--> a + x
sage: h = CM(a+u); h.display()
M --> R
on V: (u, v) |--> a + u
```

If the symbolic expression involves coordinates of different charts, the scalar field is created as a Python object, but is not initialized, in order to avoid any ambiguity:

```
sage: h = CM(x+u); h.display()
M --> R
```

TESTS:

Ring laws:

```
sage: h = CM(pi*sqrt(2))
sage: s = f + h ; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M --> R
on U: (x, y) |--> sqrt(2)*pi + arctan(x^2 + y^2)
on V: (u, v) |--> 1/2*pi*(2*sqrt(2) + 1) - arctan(u^2 + v^2)
```

```
sage: s = f - h ; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M --> R
on U: (x, y) |--> -sqrt(2)*pi + arctan(x^2 + y^2)
on V: (u, v) |--> -1/2*pi*(2*sqrt(2) - 1) - arctan(u^2 + v^2)
```

```
sage: s = f*h ; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M --> R
on U: (x, y) |--> sqrt(2)*pi*arctan(x^2 + y^2)
on V: (u, v) |--> 1/2*sqrt(2)*(pi^2 - 2*pi*arctan(u^2 + v^2))
```

```
sage: s = f/h ; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M --> R
on U: (x, y) |--> 1/2*sqrt(2)*arctan(x^2 + y^2)/pi
on V: (u, v) |--> 1/4*(sqrt(2)*pi - 2*sqrt(2)*arctan(u^2 + v^2))/pi
```

```
sage: f*(h+f) == f*h + f*f
True
```

Ring laws with coercion:

```
sage: f - fW == CW.zero()
True
sage: f/fW == CW.one()
True
sage: s = f*fW ; s
Scalar field on the Open subset W of the 2-dimensional topological
 manifold M
sage: s.display()
W --> R
(x, y) |--> arctan(x^2 + y^2)^2
(u, v) |--> 1/4*pi^2 - pi*arctan(u^2 + v^2) + arctan(u^2 + v^2)^2
sage: s/f == fW
True
```

Multiplication by a real number:

```
sage: s = 2*f ; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M --> R
on U: (x, y) |--> 2*arctan(x^2 + y^2)
on V: (u, v) |--> pi - 2*arctan(u^2 + v^2)
```

```
sage: 0*f == CM.zero()
True
sage: 1*f == f
True
sage: 2*(f/2) == f
True
sage: (f+2*f)/3 == f
True
sage: 1/3*(f+2*f) == f
True
```

The Sage test suite for algebras is passed:

```
sage: TestSuite(CM).run()
```

It is passed also for $C^0(W)$:

```
sage: TestSuite(CW).run()
```

Element
alias of ScalarField

one ( )
Return the unit element of the algebra.

This is nothing but the constant scalar field 1 on the manifold, where 1 is the unit element of the base field.

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: CM = M.scalar_field_algebra()
sage: h = CM.one(); h
Scalar field 1 on the 2-dimensional topological manifold M
sage: h.display()
1: M --> R
   (x, y) |--> 1
```

The result is cached:

```
sage: CM.one() is h
True
```

zero ( )
Return the zero element of the algebra.

This is nothing but the constant scalar field 0 on the manifold, where 0 is the zero element of the base field.

EXAMPLE:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: CM = M.scalar_field_algebra()
sage: z = CM.zero(); z
Scalar field zero on the 2-dimensional topological manifold M
sage: z.display()
zero: M --> R
   (x, y) |--> 0
```

The result is cached:

```
sage: CM.zero() is z
True
```

1.6.2 Scalar Fields

Given a topological manifold M over a topological field K (in most applications, $K = \mathbf{R}$ or $K = \mathbf{C}$), a *scalar field* on M is a continuous map

$$f : M \longrightarrow K$$

Scalar fields are implemented by the class *ScalarField*.

AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2013-2015): initial version
- Travis Scrimshaw (2016): review tweaks

REFERENCES:

- [Lee11] J.M. Lee : *Introduction to Topological Manifolds*, 2nd ed., Springer (New York) (2011)
- [KN63] S. Kobayashi & K. Nomizu : *Foundations of Differential Geometry*, vol. 1, Interscience Publishers (New York) (1963)

class `sage.manifolds.scalarfield.`**`ScalarField`**`(`*parent*, *coord_expression=None*, *chart=None*, *name=None*, *latex_name=None*`)`

Bases: `sage.structure.element.CommutativeAlgebraElement`

Scalar field on a topological manifold.

Given a topological manifold M over a topological field K (in most applications, $K = \mathbf{R}$ or $K = \mathbf{C}$), a *scalar field on* M is a continuous map

$$f : M \longrightarrow K.$$

A scalar field on M is an element of the commutative algebra $C^0(M)$ (see *ScalarFieldAlgebra*).

INPUT:

- `parent` – the algebra of scalar fields containing the scalar field (must be an instance of class *ScalarFieldAlgebra*)
- `coord_expression` – (default: `None`) coordinate expression(s) of the scalar field; this can be either
  - a dictionary of coordinate expressions in various charts on the domain, with the charts as keys;
  - a single coordinate expression; if the argument `chart` is `'all'` , this expression is set to all the charts defined on the open set; otherwise, the expression is set in the specific chart provided by the argument `chart`
- `chart` – (default: `None`) chart defining the coordinates used in `coord_expression` when the latter is a single coordinate expression; if none is provided (default), the default chart of the open set is assumed. If `chart=='all'` , `coord_expression` is assumed to be independent of the chart (constant scalar field).
- `name` – (default: `None`) string; name (symbol) given to the scalar field
- `latex_name` – (default: `None`) string; LaTeX symbol to denote the scalar field; if none is provided, the LaTeX symbol is set to `name`

If `coord_expression` is `None` or incomplete, coordinate expressions can be added after the creation of the object, by means of the methods *add_expr()* , *add_expr_by_continuation()* and *set_expr()* .

EXAMPLES:

A scalar field on the 2-sphere:

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

For scalar fields defined by a single coordinate expression, the latter can be passed instead of the dictionary over the charts:

```
sage: g = U.scalar_field(x*y, chart=c_xy, name='g') ; g
Scalar field g on the Open subset U of the 2-dimensional topological
 manifold M
```

The above is indeed equivalent to:

```
sage: g = U.scalar_field({c_xy: x*y}, name='g') ; g
Scalar field g on the Open subset U of the 2-dimensional topological
 manifold M
```

Since `c_xy` is the default chart of `U`, the argument `chart` can be skipped:

```
sage: g = U.scalar_field(x*y, name='g') ; g
Scalar field g on the Open subset U of the 2-dimensional topological
 manifold M
```

The scalar field g is defined on U and has an expression in terms of the coordinates (u, v) on $W = U \cap V$:

```
sage: g.display()
g: U --> R
   (x, y) |--> x*y
on W: (u, v) |--> u*v/(u^4 + 2*u^2*v^2 + v^4)
```

Scalar fields on M can also be declared with a single chart:

```
sage: f = M.scalar_field(1/(1+x^2+y^2), chart=c_xy, name='f') ; f
Scalar field f on the 2-dimensional topological manifold M
```

Their definition must then be completed by providing the expressions on other charts, via the method *add_expr()*, to get a global cover of the manifold:

```
sage: f.add_expr((u^2+v^2)/(1+u^2+v^2), chart=c_uv)
sage: f.display()
f: M --> R
on U: (x, y) |--> 1/(x^2 + y^2 + 1)
on V: (u, v) |--> (u^2 + v^2)/(u^2 + v^2 + 1)
```

We can even first declare the scalar field without any coordinate expression and provide them subsequently:

```
sage: f = M.scalar_field(name='f')
sage: f.add_expr(1/(1+x^2+y^2), chart=c_xy)
sage: f.add_expr((u^2+v^2)/(1+u^2+v^2), chart=c_uv)
sage: f.display()
f: M --> R
```

```
on U: (x, y) |--> 1/(x^2 + y^2 + 1)
on V: (u, v) |--> (u^2 + v^2)/(u^2 + v^2 + 1)
```

We may also use the method `add_expr_by_continuation()` to complete the coordinate definition using the analytic continuation from domains in which charts overlap:

```
sage: f = M.scalar_field(1/(1+x^2+y^2), chart=c_xy, name='f') ; f
Scalar field f on the 2-dimensional topological manifold M
sage: f.add_expr_by_continuation(c_uv, U.intersection(V))
sage: f.display()
f: M --> R
on U: (x, y) |--> 1/(x^2 + y^2 + 1)
on V: (u, v) |--> (u^2 + v^2)/(u^2 + v^2 + 1)
```

A scalar field can also be defined by some unspecified function of the coordinates:

```
sage: h = U.scalar_field(function('H')(x, y), name='h') ; h
Scalar field h on the Open subset U of the 2-dimensional topological
 manifold M
sage: h.display()
h: U --> R
   (x, y) |--> H(x, y)
on W: (u, v) |--> H(u/(u^2 + v^2), v/(u^2 + v^2))
```

We may use the argument `latex_name` to specify the LaTeX symbol denoting the scalar field if the latter is different from `name` :

```
sage: latex(f)
f
sage: f = M.scalar_field({c_xy: 1/(1+x^2+y^2), c_uv: (u^2+v^2)/(1+u^2+v^2)},
....:                    name='f', latex_name=r'\mathcal{F}')
sage: latex(f)
\mathcal{F}
```

The coordinate expression in a given chart is obtained via the method `expr()` , which returns a symbolic expression:

```
sage: f.expr(c_uv)
(u^2 + v^2)/(u^2 + v^2 + 1)
sage: type(f.expr(c_uv))
<type 'sage.symbolic.expression.Expression'>
```

The method `coord_function()` returns instead a function of the chart coordinates, i.e. an instance of `CoordFunction`:

```
sage: f.coord_function(c_uv)
(u^2 + v^2)/(u^2 + v^2 + 1)
sage: type(f.coord_function(c_uv))
<class 'sage.manifolds.coord_func_symb.CoordFunctionSymbRing_with_category.
↪element_class'>
sage: f.coord_function(c_uv).display()
(u, v) |--> (u^2 + v^2)/(u^2 + v^2 + 1)
```

The value returned by the method `expr()` is actually the coordinate expression of the chart function:

```
sage: f.expr(c_uv) is f.coord_function(c_uv).expr()
True
```

A constant scalar field is declared by setting the argument `chart` to `'all'`:

```
sage: c = M.scalar_field(2, chart='all', name='c') ; c
Scalar field c on the 2-dimensional topological manifold M
sage: c.display()
c: M --> R
on U: (x, y) |--> 2
on V: (u, v) |--> 2
```

A shortcut is to use the method *constant_scalar_field()*:

```
sage: c == M.constant_scalar_field(2)
True
```

The constant value can be some unspecified parameter:

```
sage: var('a')
a
sage: c = M.constant_scalar_field(a, name='c') ; c
Scalar field c on the 2-dimensional topological manifold M
sage: c.display()
c: M --> R
on U: (x, y) |--> a
on V: (u, v) |--> a
```

A special case of constant field is the zero scalar field:

```
sage: zer = M.constant_scalar_field(0) ; zer
Scalar field zero on the 2-dimensional topological manifold M
sage: zer.display()
zero: M --> R
on U: (x, y) |--> 0
on V: (u, v) |--> 0
```

It can be obtained directly by means of the function *zero_scalar_field()*:

```
sage: zer is M.zero_scalar_field()
True
```

A third way is to get it as the zero element of the algebra $C^0(M)$ of scalar fields on M (see below):

```
sage: zer is M.scalar_field_algebra().zero()
True
```

By definition, a scalar field acts on the manifold's points, sending them to elements of the manifold's base field (real numbers in the present case):

```
sage: N = M.point((0,0), chart=c_uv) # the North pole
sage: S = M.point((0,0), chart=c_xy) # the South pole
sage: E = M.point((1,0), chart=c_xy) # a point at the equator
sage: f(N)
0
sage: f(S)
1
sage: f(E)
1/2
sage: h(E)
H(1, 0)
```

```
sage: c(E)
a
sage: zer(E)
0
```

A scalar field can be compared to another scalar field:

```
sage: f == g
False
```

...to a symbolic expression:

```
sage: f == x*y
False
sage: g == x*y
True
sage: c == a
True
```

...to a number:

```
sage: f == 2
False
sage: zer == 0
True
```

...to anything else:

```
sage: f == M
False
```

Standard mathematical functions are implemented:

```
sage: sqrt(f)
Scalar field sqrt(f) on the 2-dimensional topological manifold M
sage: sqrt(f).display()
sqrt(f): M --> R
on U: (x, y) |--> 1/sqrt(x^2 + y^2 + 1)
on V: (u, v) |--> sqrt(u^2 + v^2)/sqrt(u^2 + v^2 + 1)
```

```
sage: tan(f)
Scalar field tan(f) on the 2-dimensional topological manifold M
sage: tan(f).display()
tan(f): M --> R
on U: (x, y) |--> sin(1/(x^2 + y^2 + 1))/cos(1/(x^2 + y^2 + 1))
on V: (u, v) |--> sin((u^2 + v^2)/(u^2 + v^2 + 1))/cos((u^2 + v^2)/(u^2 + v^2 +␣
↪1))
```

Arithmetics of scalar fields

Scalar fields on M (resp. U) belong to the algebra $C^0(M)$ (resp. $C^0(U)$):

```
sage: f.parent()
Algebra of scalar fields on the 2-dimensional topological manifold M
sage: f.parent() is M.scalar_field_algebra()
```

```
True
sage: g.parent()
Algebra of scalar fields on the Open subset U of the 2-dimensional
 topological manifold M
sage: g.parent() is U.scalar_field_algebra()
True
```

Consequently, scalar fields can be added:

```
sage: s = f + c ; s
Scalar field f+c on the 2-dimensional topological manifold M
sage: s.display()
f+c: M --> R
on U: (x, y) |--> (a*x^2 + a*y^2 + a + 1)/(x^2 + y^2 + 1)
on V: (u, v) |--> ((a + 1)*u^2 + (a + 1)*v^2 + a)/(u^2 + v^2 + 1)
```

and subtracted:

```
sage: s = f - c ; s
Scalar field f-c on the 2-dimensional topological manifold M
sage: s.display()
f-c: M --> R
on U: (x, y) |--> -(a*x^2 + a*y^2 + a - 1)/(x^2 + y^2 + 1)
on V: (u, v) |--> -((a - 1)*u^2 + (a - 1)*v^2 + a)/(u^2 + v^2 + 1)
```

Some tests:

```
sage: f + zer == f
True
sage: f - f == zer
True
sage: f + (-f) == zer
True
sage: (f+c)-f == c
True
sage: (f-c)+c == f
True
```

We may add a number (interpreted as a constant scalar field) to a scalar field:

```
sage: s = f + 1 ; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M --> R
on U: (x, y) |--> (x^2 + y^2 + 2)/(x^2 + y^2 + 1)
on V: (u, v) |--> (2*u^2 + 2*v^2 + 1)/(u^2 + v^2 + 1)
sage: (f+1)-1 == f
True
```

The number can represented by a symbolic variable:

```
sage: s = a + f ; s
Scalar field on the 2-dimensional topological manifold M
sage: s == c + f
True
```

However if the symbolic variable is a chart coordinate, the addition is performed only on the chart domain:

```
sage: s = f + x; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M --> R
on U: (x, y) |--> (x^3 + x*y^2 + x + 1)/(x^2 + y^2 + 1)
sage: s = f + u; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M --> R
on V: (u, v) |--> (u^3 + (u + 1)*v^2 + u^2 + u)/(u^2 + v^2 + 1)
```

The addition of two scalar fields with different domains is possible if the domain of one of them is a subset of the domain of the other; the domain of the result is then this subset:

```
sage: f.domain()
2-dimensional topological manifold M
sage: g.domain()
Open subset U of the 2-dimensional topological manifold M
sage: s = f + g ; s
Scalar field on the Open subset U of the 2-dimensional topological
 manifold M
sage: s.domain()
Open subset U of the 2-dimensional topological manifold M
sage: s.display()
U --> R
(x, y) |--> (x*y^3 + (x^3 + x)*y + 1)/(x^2 + y^2 + 1)
on W: (u, v) |--> (u^6 + 3*u^4*v^2 + 3*u^2*v^4 + v^6 + u*v^3
 + (u^3 + u)*v)/(u^6 + v^6 + (3*u^2 + 1)*v^4 + u^4 + (3*u^4 + 2*u^2)*v^2)
```

The operation actually performed is $f|_U + g$:

```
sage: s == f.restrict(U) + g
True
```

In Sage framework, the addition of f and g is permitted because there is a *coercion* of the parent of f, namely $C^0(M)$, to the parent of g, namely $C^0(U)$ (see `ScalarFieldAlgebra`):

```
sage: CM = M.scalar_field_algebra()
sage: CU = U.scalar_field_algebra()
sage: CU.has_coerce_map_from(CM)
True
```

The coercion map is nothing but the restriction to domain U:

```
sage: CU.coerce(f) == f.restrict(U)
True
```

Since the algebra $C^0(M)$ is a vector space over $\mathbf{R}$, scalar fields can be multiplied by a number, either an explicit one:

```
sage: s = 2*f ; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M --> R
on U: (x, y) |--> 2/(x^2 + y^2 + 1)
on V: (u, v) |--> 2*(u^2 + v^2)/(u^2 + v^2 + 1)
```

or a symbolic one:

```
sage: s = a*f ; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M --> R
on U: (x, y) |--> a/(x^2 + y^2 + 1)
on V: (u, v) |--> (u^2 + v^2)*a/(u^2 + v^2 + 1)
```

However, if the symbolic variable is a chart coordinate, the multiplication is performed only in the corresponding chart:

```
sage: s = x*f; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M --> R
on U: (x, y) |--> x/(x^2 + y^2 + 1)
sage: s = u*f; s
Scalar field on the 2-dimensional topological manifold M
sage: s.display()
M --> R
on V: (u, v) |--> (u^2 + v^2)*u/(u^2 + v^2 + 1)
```

Some tests:

```
sage: 0*f == 0
True
sage: 0*f == zer
True
sage: 1*f == f
True
sage: (-2)*f == - f - f
True
```

The ring multiplication of the algebras $C^0(M)$ and $C^0(U)$ is the pointwise multiplication of functions:

```
sage: s = f*f ; s
Scalar field f*f on the 2-dimensional topological manifold M
sage: s.display()
f*f: M --> R
on U: (x, y) |--> 1/(x^4 + y^4 + 2*(x^2 + 1)*y^2 + 2*x^2 + 1)
on V: (u, v) |--> (u^4 + 2*u^2*v^2 + v^4)/(u^4 + v^4 + 2*(u^2 + 1)*v^2
 + 2*u^2 + 1)
sage: s = g*h ; s
Scalar field g*h on the Open subset U of the 2-dimensional topological
 manifold M
sage: s.display()
g*h: U --> R
   (x, y) |--> x*y*H(x, y)
on W: (u, v) |--> u*v*H(u/(u^2 + v^2), v/(u^2 + v^2))/(u^4 + 2*u^2*v^2 + v^4)
```

Thanks to the coercion $C^0(M) \to C^0(U)$ 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 on the Open subset U of the 2-dimensional topological
 manifold M
```

```
sage: s.display()
U --> R
(x, y) |--> x*y/(x^2 + y^2 + 1)
on W: (u, v) |--> u*v/(u^4 + v^4 + (2*u^2 + 1)*v^2 + u^2)
sage: s == f.restrict(U)*g
True
```

Scalar fields can be divided (pointwise division):

```
sage: s = f/c ; s
Scalar field f/c on the 2-dimensional topological manifold M
sage: s.display()
f/c: M --> R
on U: (x, y) |--> 1/(a*x^2 + a*y^2 + a)
on V: (u, v) |--> (u^2 + v^2)/(a*u^2 + a*v^2 + a)
sage: s = g/h ; s
Scalar field g/h on the Open subset U of the 2-dimensional topological
 manifold M
sage: s.display()
g/h: U --> R
   (x, y) |--> x*y/H(x, y)
on W: (u, v) |--> u*v/((u^4 + 2*u^2*v^2 + v^4)*H(u/(u^2 + v^2), v/(u^2 + v^2)))
sage: s = f/g ; s
Scalar field on the Open subset U of the 2-dimensional topological
 manifold M
sage: s.display()
U --> R
(x, y) |--> 1/(x*y^3 + (x^3 + x)*y)
on W: (u, v) |--> (u^6 + 3*u^4*v^2 + 3*u^2*v^4 + v^6)/(u*v^3 + (u^3 + u)*v)
sage: s == f.restrict(U)/g
True
```

For scalar fields defined on a single chart domain, we may perform some arithmetics with symbolic expressions involving the chart coordinates:

```
sage: s = g + x^2 - y ; s
Scalar field on the Open subset U of the 2-dimensional topological
 manifold M
sage: s.display()
U --> R
(x, y) |--> x^2 + (x - 1)*y
on W: (u, v) |--> -(v^3 - u^2 + (u^2 - u)*v)/(u^4 + 2*u^2*v^2 + v^4)
```

```
sage: s = g*x ; s
Scalar field on the Open subset U of the 2-dimensional topological
 manifold M
sage: s.display()
U --> R
(x, y) |--> x^2*y
on W: (u, v) |--> u^2*v/(u^6 + 3*u^4*v^2 + 3*u^2*v^4 + v^6)
```

```
sage: s = g/x ; s
Scalar field on the Open subset U of the 2-dimensional topological
 manifold M
sage: s.display()
U --> R
(x, y) |--> y
```

```
on W: (u, v) |--> v/(u^2 + v^2)
sage: s = x/g ; s
Scalar field on the Open subset U of the 2-dimensional topological
 manifold M
sage: s.display()
U --> R
(x, y) |--> 1/y
on W: (u, v) |--> (u^2 + v^2)/v
```

The test suite is passed:

```
sage: TestSuite(f).run()
sage: TestSuite(zer).run()
```

add_expr (*coord_expression*, *chart=None*)

Add some coordinate expression to the scalar field.

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

INPUT:

- `coord_expression` – coordinate expression of the scalar field
- `chart` – (default: `None`) chart in which `coord_expression` is defined; if `None` , the default chart of the scalar field's domain is assumed

Warning: If the scalar field has already expressions in other charts, it is the user's responsibility to make sure that the expression to be added is consistent with them.

EXAMPLES:

Adding scalar field expressions on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: f = M.scalar_field(x^2 + 2*x*y +1)
sage: f._express
{Chart (M, (x, y)): x^2 + 2*x*y + 1}
sage: f.add_expr(3*y)
sage: f._express  # the (x,y) expression has been changed:
{Chart (M, (x, y)): 3*y}
sage: c_uv.<u,v> = M.chart()
sage: f.add_expr(cos(u)-sin(v), c_uv)
sage: f._express # random (dict. output); f has now 2 expressions:
{Chart (M, (x, y)): 3*y, Chart (M, (u, v)): cos(u) - sin(v)}
```

add_expr_by_continuation (*chart*, *subdomain*)

Set coordinate expression in a chart by continuation of the coordinate expression in a subchart.

The continuation is performed by demanding that the coordinate expression is identical to that in the restriction of the chart to a given subdomain.

INPUT:

- `chart` – coordinate chart $(U,(x^i))$ in which the expression of the scalar field is to set
- `subdomain` – open subset $V \subset U$ in which the expression in terms of the restriction of the coordinate chart $(U,(x^i))$ to V is already known or can be evaluated by a change of coordinates.

EXAMPLES:

Scalar field on the sphere S^2:

```
sage: M = Manifold(2, 'S^2', structure='topological')
sage: U = M.open_subset('U') ; V = M.open_subset('V') # the complement of␣
↪resp. N pole and S pole
sage: M.declare_union(U,V)   # S^2 is the union of U and V
sage: c_xy.<x,y> = U.chart() ; c_uv.<u,v> = V.chart() # stereographic␣
↪coordinates
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                intersection_name='W', restrictions1= x^2+y^2!=0,
....:                restrictions2= u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: W =  U.intersection(V)  # S^2 minus the two poles
sage: f = M.scalar_field(atan(x^2+y^2), chart=c_xy, name='f')
```

The scalar field has been defined only on the domain covered by the chart `c_xy`, i.e. U:

```
sage: f.display()
f: S^2 --> R
on U: (x, y) |--> arctan(x^2 + y^2)
```

We note that on $W = U \cap V$, the expression of f in terms of coordinates (u, v) can be deduced from that in the coordinates (x, y) thanks to the transition map between the two charts:

```
sage: f.display(c_uv.restrict(W))
f: S^2 --> R
on W: (u, v) |--> arctan(1/(u^2 + v^2))
```

We use this fact to extend the definition of f to the open subset V, covered by the chart `c_uv`:

```
sage: f.add_expr_by_continuation(c_uv, W)
```

Then, f is known on the whole sphere:

```
sage: f.display()
f: S^2 --> R
on U: (x, y) |--> arctan(x^2 + y^2)
on V: (u, v) |--> arctan(1/(u^2 + v^2))
```

arccos ()

Arc cosine of the scalar field.

OUTPUT:

- the scalar field $\arccos f$, where f is the current scalar field

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x*y}, name='f', latex_name=r"\Phi")
sage: g = arccos(f) ; g
Scalar field arccos(f) on the 2-dimensional topological manifold M
sage: latex(g)
\arccos\left(\Phi\right)
sage: g.display()
arccos(f): M --> R
   (x, y) |--> arccos(x*y)
```

The notation `acos` can be used as well:

```
sage: acos(f)
Scalar field arccos(f) on the 2-dimensional topological manifold M
sage: acos(f) == g
True
```

Some tests:

```
sage: cos(g) == f
True
sage: arccos(M.constant_scalar_field(1)) == M.zero_scalar_field()
True
sage: arccos(M.zero_scalar_field()) == M.constant_scalar_field(pi/2)
True
```

arccosh ( )

Inverse hyperbolic cosine of the scalar field.

OUTPUT:

- the scalar field $\mathrm{arcosh}\, f$, where f is the current scalar field

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x*y}, name='f', latex_name=r"\Phi")
sage: g = arccosh(f) ; g
Scalar field arccosh(f) on the 2-dimensional topological manifold M
sage: latex(g)
\,\mathrm{arcosh}\left(\Phi\right)
sage: g.display()
arccosh(f): M --> R
   (x, y) |--> arccosh(x*y)
```

The notation `acosh` can be used as well:

```
sage: acosh(f)
Scalar field arccosh(f) on the 2-dimensional topological manifold M
sage: acosh(f) == g
True
```

Some tests:

```
sage: cosh(g) == f
True
sage: arccosh(M.constant_scalar_field(1)) == M.zero_scalar_field()
True
```

arcsin ( )

Arc sine of the scalar field.

OUTPUT:

- the scalar field $\arcsin f$, where f is the current scalar field

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x*y}, name='f', latex_name=r"\Phi")
sage: g = arcsin(f) ; g
Scalar field arcsin(f) on the 2-dimensional topological manifold M
sage: latex(g)
\arcsin\left(\Phi\right)
sage: g.display()
arcsin(f): M --> R
   (x, y) |--> arcsin(x*y)
```

The notation `asin` can be used as well:

```
sage: asin(f)
Scalar field arcsin(f) on the 2-dimensional topological manifold M
sage: asin(f) == g
True
```

Some tests:

```
sage: sin(g) == f
True
sage: arcsin(M.zero_scalar_field()) == M.zero_scalar_field()
True
sage: arcsin(M.constant_scalar_field(1)) == M.constant_scalar_field(pi/2)
True
```

arcsinh ( )

Inverse hyperbolic sine of the scalar field.

OUTPUT:

- the scalar field $\operatorname{arsinh} f$, where f is the current scalar field

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x*y}, name='f', latex_name=r"\Phi")
sage: g = arcsinh(f) ; g
Scalar field arcsinh(f) on the 2-dimensional topological manifold M
sage: latex(g)
\,\mathrm{arsinh}\left(\Phi\right)
sage: g.display()
arcsinh(f): M --> R
   (x, y) |--> arcsinh(x*y)
```

The notation `asinh` can be used as well:

```
sage: asinh(f)
Scalar field arcsinh(f) on the 2-dimensional topological manifold M
sage: asinh(f) == g
True
```

Some tests:

```
sage: sinh(g) == f
True
```

```
sage: arcsinh(M.zero_scalar_field()) == M.zero_scalar_field()
True
```

arctan ( )

Arc tangent of the scalar field.

OUTPUT:

•the scalar field $\arctan f$, where f is the current scalar field

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x*y}, name='f', latex_name=r"\Phi")
sage: g = arctan(f) ; g
Scalar field arctan(f) on the 2-dimensional topological manifold M
sage: latex(g)
\arctan\left(\Phi\right)
sage: g.display()
arctan(f): M --> R
   (x, y) |--> arctan(x*y)
```

The notation `atan` can be used as well:

```
sage: atan(f)
Scalar field arctan(f) on the 2-dimensional topological manifold M
sage: atan(f) == g
True
```

Some tests:

```
sage: tan(g) == f
True
sage: arctan(M.zero_scalar_field()) == M.zero_scalar_field()
True
sage: arctan(M.constant_scalar_field(1)) == M.constant_scalar_field(pi/4)
True
```

arctanh ( )

Inverse hyperbolic tangent of the scalar field.

OUTPUT:

•the scalar field $\mathrm{artanh}\, f$, where f is the current scalar field

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x*y}, name='f', latex_name=r"\Phi")
sage: g = arctanh(f) ; g
Scalar field arctanh(f) on the 2-dimensional topological manifold M
sage: latex(g)
\,\mathrm{artanh}\left(\Phi\right)
sage: g.display()
arctanh(f): M --> R
   (x, y) |--> arctanh(x*y)
```

The notation `atanh` can be used as well:

```
sage: atanh(f)
Scalar field arctanh(f) on the 2-dimensional topological manifold M
sage: atanh(f) == g
True
```

Some tests:

```
sage: tanh(g) == f
True
sage: arctanh(M.zero_scalar_field()) == M.zero_scalar_field()
True
sage: arctanh(M.constant_scalar_field(1/2)) == M.constant_scalar_field(log(3)/
↪2)
True
```

common_charts (*other*)

Find common charts for the expressions of the scalar field and `other` .

INPUT:

•`other` – a scalar field

OUPUT:

•list of common charts; if no common chart is found, `None` is returned (instead of an empty list)

EXAMPLES:

Search for common charts on a 2-dimensional manifold with 2 overlapping domains:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: U = M.open_subset('U')
sage: c_xy.<x,y> = U.chart()
sage: V = M.open_subset('V')
sage: c_uv.<u,v> = V.chart()
sage: M.declare_union(U,V)   # M is the union of U and V
sage: f = U.scalar_field(x^2)
sage: g = M.scalar_field(x+y)
sage: f.common_charts(g)
[Chart (U, (x, y))]
sage: g.add_expr(u, c_uv)
sage: f._express
{Chart (U, (x, y)): x^2}
sage: g._express  # random (dictionary output)
{Chart (U, (x, y)): x + y, Chart (V, (u, v)): u}
sage: f.common_charts(g)
[Chart (U, (x, y))]
```

Common charts found as subcharts: the subcharts are introduced via a transition map between charts c_xy and c_uv on the intersecting subdomain $W = U \cap V$:

```
sage: trans = c_xy.transition_map(c_uv, (x+y, x-y), 'W', x<0, u+v<0)
sage: M.atlas()
[Chart (U, (x, y)), Chart (V, (u, v)), Chart (W, (x, y)),
 Chart (W, (u, v))]
sage: c_xy_W = M.atlas()[2]
sage: c_uv_W = M.atlas()[3]
sage: trans.inverse()
Change of coordinates from Chart (W, (u, v)) to Chart (W, (x, y))
sage: f.common_charts(g)
```

```
[Chart (U, (x, y))]
sage: f.expr(c_xy_W)
x^2
sage: f._express  # random (dictionary output)
{Chart (U, (x, y)): x^2, Chart (W, (x, y)): x^2}
sage: g._express  # random (dictionary output)
{Chart (U, (x, y)): x + y, Chart (V, (u, v)): u}
sage: g.common_charts(f)  # c_xy_W is not returned because it is subchart of
↪'xy'
[Chart (U, (x, y))]
sage: f.expr(c_uv_W)
1/4*u^2 + 1/2*u*v + 1/4*v^2
sage: f._express  # random (dictionary output)
{Chart (U, (x, y)): x^2, Chart (W, (x, y)): x^2,
 Chart (W, (u, v)): 1/4*u^2 + 1/2*u*v + 1/4*v^2}
sage: g._express  # random (dictionary output)
{Chart (U, (x, y)): x + y, Chart (V, (u, v)): u}
sage: f.common_charts(g)
[Chart (U, (x, y)), Chart (W, (u, v))]
sage: # the expressions have been updated on the subcharts
sage: g._express #  random (dictionary output)
{Chart (U, (x, y)): x + y, Chart (V, (u, v)): u,
 Chart (W, (u, v)): u}
```

Common charts found by computing some coordinate changes:

```
sage: W = U.intersection(V)
sage: f = W.scalar_field(x^2, c_xy_W)
sage: g = W.scalar_field(u+1, c_uv_W)
sage: f._express
{Chart (W, (x, y)): x^2}
sage: g._express
{Chart (W, (u, v)): u + 1}
sage: f.common_charts(g)
[Chart (W, (u, v)), Chart (W, (x, y))]
sage: f._express # random (dictionary output)
{Chart (W, (u, v)): 1/4*u^2 + 1/2*u*v + 1/4*v^2,
 Chart (W, (x, y)): x^2}
sage: g._express # random (dictionary output)
{Chart (W, (u, v)): u + 1, Chart (W, (x, y)): x + y + 1}
```

coord_function (*chart=None*, *from_chart=None*)

Return the function of the coordinates representing the scalar field in a given chart.

INPUT:

- `chart` – (default: `None`) chart with respect to which the coordinate expression is to be returned; if `None` , the default chart of the scalar field's domain will be used
- `from_chart` – (default: `None`) chart from which the required expression is computed if it is not known already in the chart `chart` ; if `None` , a chart is picked in the known expressions

OUTPUT:

- instance of `CoordFunction` representing the coordinate function of the scalar field in the given chart

EXAMPLES:

Coordinate function on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: f = M.scalar_field(x*y^2)
sage: f.coord_function()
x*y^2
sage: f.coord_function(c_xy)  # equivalent form (since c_xy is the default␣
↪chart)
x*y^2
sage: type(f.coord_function())
<class 'sage.manifolds.coord_func_symb.CoordFunctionSymbRing_with_category.
↪element_class'>
```

Expression via a change of coordinates:

```
sage: c_uv.<u,v> = M.chart()
sage: c_uv.transition_map(c_xy, [u+v, u-v])
Change of coordinates from Chart (M, (u, v)) to Chart (M, (x, y))
sage: f._express # at this stage, f is expressed only in terms of (x,y)␣
↪coordinates
{Chart (M, (x, y)): x*y^2}
sage: f.coord_function(c_uv) # forces the computation of the expression of f␣
↪in terms of (u,v) coordinates
u^3 - u^2*v - u*v^2 + v^3
sage: f.coord_function(c_uv) == (u+v)*(u-v)^2  # check
True
sage: f._express  # random (dict. output); f has now 2 coordinate expressions:
{Chart (M, (x, y)): x*y^2, Chart (M, (u, v)): u^3 - u^2*v - u*v^2 + v^3}
```

Usage in a physical context (simple Lorentz transformation - boost in `x` direction, with relative velocity `v` between `o1` and `o2` frames):

```
sage: M = Manifold(2, 'M', structure='topological')
sage: o1.<t,x> = M.chart()
sage: o2.<T,X> = M.chart()
sage: f = M.scalar_field(x^2 - t^2)
sage: f.coord_function(o1)
-t^2 + x^2
sage: v = var('v'); gam = 1/sqrt(1-v^2)
sage: o2.transition_map(o1, [gam*(T - v*X), gam*(X - v*T)])
Change of coordinates from Chart (M, (T, X)) to Chart (M, (t, x))
sage: f.coord_function(o2)
-T^2 + X^2
```

copy ( )

Return an exact copy of the scalar field.

EXAMPLES:

Copy on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: f = M.scalar_field(x*y^2)
sage: g = f.copy()
sage: type(g)
<class 'sage.manifolds.scalarfield.ScalarFieldAlgebra_with_category.element_
↪class'>
sage: g.expr()
```

```
x*y^2
sage: g == f
True
sage: g is f
False
```

cos ( )

Cosine of the scalar field.

OUTPUT:

- the scalar field $\cos f$, where f is the current scalar field

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x*y}, name='f', latex_name=r"\Phi")
sage: g = cos(f) ; g
Scalar field cos(f) on the 2-dimensional topological manifold M
sage: latex(g)
\cos\left(\Phi\right)
sage: g.display()
cos(f): M --> R
   (x, y) |--> cos(x*y)
```

Some tests:

```
sage: cos(M.zero_scalar_field()) == M.constant_scalar_field(1)
True
sage: cos(M.constant_scalar_field(pi/2)) == M.zero_scalar_field()
True
```

cosh ( )

Hyperbolic cosine of the scalar field.

OUTPUT:

- the scalar field $\cosh f$, where f is the current scalar field

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x*y}, name='f', latex_name=r"\Phi")
sage: g = cosh(f) ; g
Scalar field cosh(f) on the 2-dimensional topological manifold M
sage: latex(g)
\cosh\left(\Phi\right)
sage: g.display()
cosh(f): M --> R
   (x, y) |--> cosh(x*y)
```

Some test:

```
sage: cosh(M.zero_scalar_field()) == M.constant_scalar_field(1)
True
```

disp (*chart=None*)

Display the expression of the scalar field in a given chart.

Without any argument, this function displays the expressions of the scalar field in all the charts defined on the scalar field's domain that are not restrictions of another chart to some subdomain (the "top charts").

INPUT:

- `chart` – (default: `None`) chart with respect to which the coordinate expression is to be displayed; if `None` , the display is performed in all the top charts in which the coordinate expression is known

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

EXAMPLES:

Various displays:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: f = M.scalar_field(sqrt(x+1), name='f')
sage: f.display()
f: M --> R
   (x, y) |--> sqrt(x + 1)
sage: latex(f.display())
\begin{array}{llcl} f:& M & \longrightarrow & \mathbb{R} \\ & \left(x,␣
↪y\right) & \longmapsto & \sqrt{x + 1} \end{array}
sage: g = M.scalar_field(function('G')(x, y), name='g')
sage: g.display()
g: M --> R
   (x, y) |--> G(x, y)
sage: latex(g.display())
\begin{array}{llcl} g:& M & \longrightarrow & \mathbb{R} \\ & \left(x,␣
↪y\right) & \longmapsto & G\left(x, y\right) \end{array}
```

A shortcut of `display()` is `disp()` :

```
sage: f.disp()
f: M --> R
   (x, y) |--> sqrt(x + 1)
```

display (*chart=None*)

Display the expression of the scalar field in a given chart.

Without any argument, this function displays the expressions of the scalar field in all the charts defined on the scalar field's domain that are not restrictions of another chart to some subdomain (the "top charts").

INPUT:

- `chart` – (default: `None`) chart with respect to which the coordinate expression is to be displayed; if `None` , the display is performed in all the top charts in which the coordinate expression is known

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

EXAMPLES:

Various displays:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: f = M.scalar_field(sqrt(x+1), name='f')
sage: f.display()
f: M --> R
   (x, y) |--> sqrt(x + 1)
sage: latex(f.display())
\begin{array}{llcl} f:& M & \longrightarrow & \mathbb{R} \\ & \left(x,␣
↪y\right) & \longmapsto & \sqrt{x + 1} \end{array}
```

```
sage: g = M.scalar_field(function('G')(x, y), name='g')
sage: g.display()
g: M --> R
   (x, y) |--> G(x, y)
sage: latex(g.display())
\begin{array}{llcl} g:& M & \longrightarrow & \mathbb{R} \\ & \left(x,␣
↪y\right) & \longmapsto & G\left(x, y\right) \end{array}
```

A shortcut of `display()` is `disp()`:

```
sage: f.disp()
f: M --> R
   (x, y) |--> sqrt(x + 1)
```

domain()

Return the open subset on which the scalar field is defined.

OUTPUT:

- instance of class *TopologicalManifold* representing the manifold's open subset on which the scalar field is defined

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: f = M.scalar_field(x+2*y)
sage: f.domain()
2-dimensional topological manifold M
sage: U = M.open_subset('U', coord_def={c_xy: x<0})
sage: g = f.restrict(U)
sage: g.domain()
Open subset U of the 2-dimensional topological manifold M
```

exp()

Exponential of the scalar field.

OUTPUT:

- the scalar field $\exp f$, where f is the current scalar field

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x+y}, name='f', latex_name=r"\Phi")
sage: g = exp(f) ; g
Scalar field exp(f) on the 2-dimensional topological manifold M
sage: g.display()
exp(f): M --> R
   (x, y) |--> e^(x + y)
sage: latex(g)
\exp\left(\Phi\right)
```

Automatic simplifications occur:

```
sage: f = M.scalar_field({X: 2*ln(1+x^2)}, name='f')
sage: exp(f).display()
exp(f): M --> R
   (x, y) |--> x^4 + 2*x^2 + 1
```

The inverse function is `log()` :

```
sage: log(exp(f)) == f
True
```

Some tests:

```
sage: exp(M.zero_scalar_field()) == M.constant_scalar_field(1)
True
sage: exp(M.constant_scalar_field(1)) == M.constant_scalar_field(e)
True
```

expr (*chart=None*, *from_chart=None*)

Return the coordinate expression of the scalar field in a given chart.

INPUT:

- `chart` – (default: `None`) chart with respect to which the coordinate expression is required; if `None` , the default chart of the scalar field's domain will be used
- `from_chart` – (default: `None`) chart from which the required expression is computed if it is not known already in the chart `chart` ; if `None` , a chart is picked in `self._express`

OUTPUT:

- symbolic expression representing the coordinate expression of the scalar field in the given chart.

EXAMPLES:

Expression of a scalar field on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: f = M.scalar_field(x*y^2)
sage: f.expr()
x*y^2
sage: f.expr(c_xy)  # equivalent form (since c_xy is the default chart)
x*y^2
sage: type(f.expr())
<type 'sage.symbolic.expression.Expression'>
```

Expression via a change of coordinates:

```
sage: c_uv.<u,v> = M.chart()
sage: c_uv.transition_map(c_xy, [u+v, u-v])
Change of coordinates from Chart (M, (u, v)) to Chart (M, (x, y))
sage: f._express # at this stage, f is expressed only in terms of (x,y)␣
↪coordinates
{Chart (M, (x, y)): x*y^2}
sage: f.expr(c_uv) # forces the computation of the expression of f in terms␣
↪of (u,v) coordinates
u^3 - u^2*v - u*v^2 + v^3
sage: bool( f.expr(c_uv) == (u+v)*(u-v)^2 ) # check
True
sage: f._express  # random (dict. output); f has now 2 coordinate expressions:
{Chart (M, (x, y)): x*y^2, Chart (M, (u, v)): u^3 - u^2*v - u*v^2 + v^3}
```

function_chart (*chart=None*, *from_chart=None*)

Deprecated.

Use `coord_function()` instead.

EXAMPLE:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: f = M.scalar_field(x*y^2)
sage: fc = f.function_chart()
doctest:...: DeprecationWarning: Use coord_function() instead.
See http://trac.sagemath.org/18640 for details.
sage: fc
x*y^2
```

log ( )

Natural logarithm of the scalar field.

OUTPUT:

- the scalar field $\ln f$, where f is the current scalar field

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x+y}, name='f', latex_name=r"\Phi")
sage: g = log(f) ; g
Scalar field ln(f) on the 2-dimensional topological manifold M
sage: g.display()
ln(f): M --> R
   (x, y) |--> log(x + y)
sage: latex(g)
\ln\left(\Phi\right)
```

The inverse function is `exp()` :

```
sage: exp(log(f)) == f
True
```

restrict (*subdomain*)

Restriction of the scalar field to an open subset of its domain of definition.

INPUT:

- `subdomain` – an open subset of the scalar field's domain

OUTPUT:

- instance of `ScalarField` representing the restriction of the scalar field to `subdomain`

EXAMPLES:

Restriction of a scalar field defined on $\mathbf{R}^2$ to the unit open disc:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()  # Cartesian coordinates
sage: U = M.open_subset('U', coord_def={X: x^2+y^2 < 1}) # U unit open disc
sage: f = M.scalar_field(cos(x*y), name='f')
sage: f_U = f.restrict(U) ; f_U
Scalar field f on the Open subset U of the 2-dimensional
 topological manifold M
sage: f_U.display()
f: U --> R
   (x, y) |--> cos(x*y)
sage: f.parent()
```

```
Algebra of scalar fields on the 2-dimensional topological
 manifold M
sage: f_U.parent()
Algebra of scalar fields on the Open subset U of the 2-dimensional
 topological manifold M
```

The restriction to the whole domain is the identity:

```
sage: f.restrict(M) is f
True
sage: f_U.restrict(U) is f_U
True
```

Restriction of the zero scalar field:

```
sage: M.zero_scalar_field().restrict(U)
Scalar field zero on the Open subset U of the 2-dimensional
 topological manifold M
sage: M.zero_scalar_field().restrict(U) is U.zero_scalar_field()
True
```

set_expr (*coord_expression*, *chart=None*)

Set the coordinate expression of the scalar field.

The expressions with respect to other charts are deleted, in order to avoid any inconsistency. To keep them, use `add_expr()` instead.

INPUT:

- `coord_expression` – coordinate expression of the scalar field
- `chart` – (default: `None`) chart in which `coord_expression` is defined; if `None` , the default chart of the scalar field's domain is assumed

EXAMPLES:

Setting scalar field expressions on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: f = M.scalar_field(x^2 + 2*x*y +1)
sage: f._express
{Chart (M, (x, y)): x^2 + 2*x*y + 1}
sage: f.set_expr(3*y)
sage: f._express  # the (x,y) expression has been changed:
{Chart (M, (x, y)): 3*y}
sage: c_uv.<u,v> = M.chart()
sage: f.set_expr(cos(u)-sin(v), c_uv)
sage: f._express # the (x,y) expression has been lost:
{Chart (M, (u, v)): cos(u) - sin(v)}
sage: f.set_expr(3*y)
sage: f._express # the (u,v) expression has been lost:
{Chart (M, (x, y)): 3*y}
```

set_name (*name=None*, *latex_name=None*)

Set (or change) the text name and LaTeX name of the scalar field.

INPUT:

- `name` – (string; default: `None`) name given to the scalar field

- `latex_name` – (string; default: `None`) LaTeX symbol to denote the scalar field; if `None` while `name` is provided, the LaTeX symbol is set to `name`

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x+y})
sage: f = M.scalar_field({X: x+y}); f
Scalar field on the 2-dimensional topological manifold M
sage: f.set_name('f'); f
Scalar field f on the 2-dimensional topological manifold M
sage: latex(f)
f
sage: f.set_name('f', latex_name=r'\Phi'); f
Scalar field f on the 2-dimensional topological manifold M
sage: latex(f)
\Phi
```

sin ( )

Sine of the scalar field.

OUTPUT:

- the scalar field $\sin f$, where f is the current scalar field

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x*y}, name='f', latex_name=r"\Phi")
sage: g = sin(f) ; g
Scalar field sin(f) on the 2-dimensional topological manifold M
sage: latex(g)
\sin\left(\Phi\right)
sage: g.display()
sin(f): M --> R
   (x, y) |--> sin(x*y)
```

Some tests:

```
sage: sin(M.zero_scalar_field()) == M.zero_scalar_field()
True
sage: sin(M.constant_scalar_field(pi/2)) == M.constant_scalar_field(1)
True
```

sinh ( )

Hyperbolic sine of the scalar field.

OUTPUT:

- the scalar field $\sinh f$, where f is the current scalar field

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x*y}, name='f', latex_name=r"\Phi")
sage: g = sinh(f) ; g
Scalar field sinh(f) on the 2-dimensional topological manifold M
sage: latex(g)
```

```
\sinh\left(\Phi\right)
sage: g.display()
sinh(f): M --> R
   (x, y) |--> sinh(x*y)
```

Some test:

```
sage: sinh(M.zero_scalar_field()) == M.zero_scalar_field()
True
```

sqrt ( )

Square root of the scalar field.

OUTPUT:

- the scalar field $\sqrt{f}$, where f is the current scalar field

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: 1+x^2+y^2}, name='f',
....:                     latex_name=r"\Phi")
sage: g = sqrt(f) ; g
Scalar field sqrt(f) on the 2-dimensional topological manifold M
sage: latex(g)
\sqrt{\Phi}
sage: g.display()
sqrt(f): M --> R
   (x, y) |--> sqrt(x^2 + y^2 + 1)
```

Some tests:

```
sage: g^2 == f
True
sage: sqrt(M.zero_scalar_field()) == M.zero_scalar_field()
True
```

tan ( )

Tangent of the scalar field.

OUTPUT:

- the scalar field $\tan f$, where f is the current scalar field

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x*y}, name='f', latex_name=r"\Phi")
sage: g = tan(f) ; g
Scalar field tan(f) on the 2-dimensional topological manifold M
sage: latex(g)
\tan\left(\Phi\right)
sage: g.display()
tan(f): M --> R
   (x, y) |--> sin(x*y)/cos(x*y)
```

Some tests:

```
sage: tan(f) == sin(f) / cos(f)
True
sage: tan(M.zero_scalar_field()) == M.zero_scalar_field()
True
sage: tan(M.constant_scalar_field(pi/4)) == M.constant_scalar_field(1)
True
```

tanh ( )

Hyperbolic tangent of the scalar field.

OUTPUT:

•the scalar field $\tanh f$, where f is the current scalar field

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: f = M.scalar_field({X: x*y}, name='f', latex_name=r"\Phi")
sage: g = tanh(f) ; g
Scalar field tanh(f) on the 2-dimensional topological manifold M
sage: latex(g)
\tanh\left(\Phi\right)
sage: g.display()
tanh(f): M --> R
   (x, y) |--> sinh(x*y)/cosh(x*y)
```

Some tests:

```
sage: tanh(f) == sinh(f) / cosh(f)
True
sage: tanh(M.zero_scalar_field()) == M.zero_scalar_field()
True
```

1.7 Continuous Maps

1.7.1 Sets of Morphisms between Topological Manifolds

The class *TopologicalManifoldHomset* implements sets of morphisms between two topological manifolds over the same topological field K, a morphism being a *continuous map* for the category of topological manifolds.

AUTHORS:

- Eric Gourgoulhon (2015): initial version
- Travis Scrimshaw (2016): review tweaks

REFERENCES:

- *[Lee11]* J.M. Lee : *Introduction to Topological Manifolds*, 2nd ed., Springer (New York) (2011)
- *[KN63]* S. Kobayashi & K. Nomizu : *Foundations of Differential Geometry*, vol. 1, Interscience Publishers (New York) (1963)

class sage.manifolds.manifold_homset.**TopologicalManifoldHomset** (*domain*, *codomain*, *name=None*, *latex_name=None*)

Bases: `sage.structure.unique_representation.UniqueRepresentation`, `sage.categories.homset.Homset`

Set of continuous maps between two topological manifolds.

Given two topological manifolds M and N over a topological field K, the class *`TopologicalManifoldHomset`* implements the set $\mathrm{Hom}(M,N)$ of morphisms (i.e. continuous maps) $M \to N$.

This is a Sage *parent* class, whose *element* class is *`ContinuousMap`*.

INPUT:

- `domain` – *`TopologicalManifold`*; the domain topological manifold M of the morphisms
- `codomain` – *`TopologicalManifold`*; the codomain topological manifold N of the morphisms
- `name` – (default: `None`) string; the name of `self`; if `None`, `Hom(M,N)` will be used
- `latex_name` – (default: `None`) string; LaTeX symbol to denote `self`; if `None`, $\mathrm{Hom}(M,N)$ will be used

EXAMPLES:

Set of continuous maps between a 2-dimensional manifold and a 3-dimensional one:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: N = Manifold(3, 'N', structure='topological')
sage: Y.<u,v,w> = N.chart()
sage: H = Hom(M, N) ; H
Set of Morphisms from 2-dimensional topological manifold M to
 3-dimensional topological manifold N in Category of manifolds over
 Real Field with 53 bits of precision
sage: type(H)
<class 'sage.manifolds.manifold_homset.TopologicalManifoldHomset_with_category'>
sage: H.category()
Category of homsets of topological spaces
sage: latex(H)
\mathrm{Hom}\left(M,N\right)
sage: H.domain()
2-dimensional topological manifold M
sage: H.codomain()
3-dimensional topological manifold N
```

An element of `H` is a continuous map from `M` to `N`:

```
sage: H.Element
<class 'sage.manifolds.continuous_map.ContinuousMap'>
sage: f = H.an_element() ; f
Continuous map from the 2-dimensional topological manifold M to the
 3-dimensional topological manifold N
sage: f.display()
M --> N
   (x, y) |--> (u, v, w) = (0, 0, 0)
```

The test suite is passed:

```
sage: TestSuite(H).run()
```

When the codomain coincides with the domain, the homset is a set of *endomorphisms* in the category of topological manifolds:

```
sage: E = Hom(M, M) ; E
Set of Morphisms from 2-dimensional topological manifold M to
 2-dimensional topological manifold M in Category of manifolds over
 Real Field with 53 bits of precision
sage: E.category()
Category of endsets of topological spaces
sage: E.is_endomorphism_set()
True
sage: E is End(M)
True
```

In this case, the homset is a monoid for the law of morphism composition:

```
sage: E in Monoids()
True
```

This was of course not the case of `H = Hom(M,N)` :

```
sage: H in Monoids()
False
```

The identity element of the monoid is of course the identity map of `M` :

```
sage: E.one()
Identity map Id_M of the 2-dimensional topological manifold M
sage: E.one() is M.identity_map()
True
sage: E.one().display()
Id_M: M --> M
   (x, y) |--> (x, y)
```

The test suite is passed by `E` :

```
sage: TestSuite(E).run()
```

This test suite includes more tests than in the case of `H` , since `E` has some extra structure (monoid).

Element
alias of `ContinuousMap`

one ( )
Return the identity element of `self` considered as a monoid (case of a set of endomorphisms).

This applies only when the codomain of the homset is equal to its domain, i.e. when the homset is of the type $\mathrm{Hom}(M, M)$. Indeed, $\mathrm{Hom}(M, M)$ equipped with the law of morphisms composition is a monoid, whose identity element is nothing but the identity map of M.

OUTPUT:

- the identity map of M, as an instance of *`ContinuousMap`*

EXAMPLE:

The identity map of a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: H = Hom(M, M) ; H
Set of Morphisms from 2-dimensional topological manifold M to
 2-dimensional topological manifold M in Category of manifolds over
 Real Field with 53 bits of precision
sage: H in Monoids()
True
sage: H.one()
Identity map Id_M of the 2-dimensional topological manifold M
sage: H.one().parent() is H
True
sage: H.one().display()
Id_M: M --> M
   (x, y) |--> (x, y)
```

The identity map is cached:

```
sage: H.one() is H.one()
True
```

If the homset is not a set of endomorphisms, the identity element is meaningless:

```
sage: N = Manifold(3, 'N', structure='topological')
sage: Y.<u,v,w> = N.chart()
sage: Hom(M, N).one()
Traceback (most recent call last):
...
TypeError: Set of Morphisms
 from 2-dimensional topological manifold M
 to 3-dimensional topological manifold N
 in Category of manifolds over Real Field with 53 bits of precision
 is not a monoid
```

1.7.2 Continuous Maps Between Topological Manifolds

`ContinuousMap` implements continuous maps from a topological manifold M to some topological manifold N over the same topological field K as M.

AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2013-2015): initial version
- Travis Scrimshaw (2016): review tweaks

REFERENCES:

- Chap. 1 of *[KN63]* S. Kobayashi & K. Nomizu : *Foundations of Differential Geometry*, vol. 1, Interscience Publishers (New York) (1963)
- *[Lee11]* J.M. Lee : *Introduction to Topological Manifolds*, 2nd ed., Springer (New York) (2011)

class `sage.manifolds.continuous_map.`**`ContinuousMap`**(*parent*, *coord_functions=None*, *name=None*, *latex_name=None*, *is_isomorphism=False*, *is_identity=False*)

Bases: `sage.categories.morphism.Morphism`

Continuous map between two topological manifolds.

This class implements continuous maps of the type

$$\Phi : M \longrightarrow N,$$

where M and N are topological manifolds over the same topological field K.

Continuous maps are the morphisms of the category of topological manifolds. The set of all continuous maps from M to N is therefore the homset between M and N, which is denoted by $\mathrm{Hom}(M, N)$.

The class *ContinuousMap* is a Sage *element* class, whose *parent* class is *TopologicalManifoldHomset*.

INPUT:

- `parent` – homset $\mathrm{Hom}(M, N)$ to which the continuous map belongs
- `coord_functions` – a dictionary of the coordinate expressions (as lists or tuples of the coordinates of the image expressed in terms of the coordinates of the considered point) with the pairs of charts `(chart1,chart2)` as keys (`chart1` being a chart on M and `chart2` a chart on N)
- `name` – (default: `None`) name given to `self`
- `latex_name` – (default: `None`) LaTeX symbol to denote the continuous map; if `None`, the LaTeX symbol is set to `name`
- `is_isomorphism` – (default: `False`) determines whether the constructed object is a isomorphism (i.e. a homeomorphism); if set to `True`, then the manifolds M and N must have the same dimension
- `is_identity` – (default: `False`) determines whether the constructed object is the identity map; if set to `True`, then N must be M and the entry `coord_functions` is not used

Note: If the information passed by means of the argument `coord_functions` is not sufficient to fully specify the continuous map, further coordinate expressions, in other charts, can be subsequently added by means of the method *add_expr()*.

EXAMPLES:

The standard embedding of the sphere S^2 into $\mathbf{R}^3$:

```
sage: M = Manifold(2, 'S^2', structure='topological') # the 2-dimensional sphere␣
↪S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V)   # S^2 is the union of U and V
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                                intersection_name='W',
....:                                restrictions1=x^2+y^2!=0,
....:                                restrictions2=u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: N = Manifold(3, 'R^3', latex_name=r'\RR^3', structure='topological')  # R^3
sage: c_cart.<X,Y,Z> = N.chart()  # Cartesian coordinates on R^3
sage: Phi = M.continuous_map(N,
....:   {(c_xy, c_cart): [2*x/(1+x^2+y^2), 2*y/(1+x^2+y^2), (x^2+y^2-1)/(1+x^2+y^
↪2)],
....:    (c_uv, c_cart): [2*u/(1+u^2+v^2), 2*v/(1+u^2+v^2), (1-u^2-v^2)/(1+u^2+v^
↪2)]},
....:   name='Phi', latex_name=r'\Phi')
sage: Phi
Continuous map Phi from the 2-dimensional topological manifold S^2
 to the 3-dimensional topological manifold R^3
sage: Phi.parent()
Set of Morphisms from 2-dimensional topological manifold S^2
```

```
 to 3-dimensional topological manifold R^3
 in Category of manifolds over Real Field with 53 bits of precision
sage: Phi.parent() is Hom(M, N)
True
sage: type(Phi)
<class 'sage.manifolds.continuous_map.TopologicalManifoldHomset_with_category.
↪element_class'>
sage: Phi.display()
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2 + y^
↪2 - 1)/(x^2 + y^2 + 1))
on V: (u, v) |--> (X, Y, Z) = (2*u/(u^2 + v^2 + 1), 2*v/(u^2 + v^2 + 1), -(u^2 +␣
↪v^2 - 1)/(u^2 + v^2 + 1))
```

It is possible to create the map using `continuous_map()` with only in a single pair of charts. The argument `coord_functions` is then a mere list of coordinate expressions (and not a dictionary) and the arguments `chart1` and `chart2` have to be provided if the charts differ from the default ones on the domain and/or codomain:

```
sage: Phi1 = M.continuous_map(N, [2*x/(1+x^2+y^2), 2*y/(1+x^2+y^2), (x^2+y^2-1)/
↪(1+x^2+y^2)],
....:                         chart1=c_xy, chart2=c_cart,
....:                         name='Phi', latex_name=r'\Phi')
```

Since `c_xy` and `c_cart` are the default charts on respectively `M` and `N`, they can be omitted, so that the above declaration is equivalent to:

```
sage: Phi1 = M.continuous_map(N, [2*x/(1+x^2+y^2), 2*y/(1+x^2+y^2), (x^2+y^2-1)/
↪(1+x^2+y^2)],
....:                         name='Phi', latex_name=r'\Phi')
```

With such a declaration, the continuous map `Phi1` is only partially defined on the manifold S^2 as it is known in only one chart:

```
sage: Phi1.display()
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2 + y^
↪2 - 1)/(x^2 + y^2 + 1))
```

The definition can be completed by using `add_expr()`:

```
sage: Phi1.add_expr(c_uv, c_cart, [2*u/(1+u^2+v^2), 2*v/(1+u^2+v^2), (1-u^2-v^2)/
↪(1+u^2+v^2)])
sage: Phi1.display()
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2 + y^
↪2 - 1)/(x^2 + y^2 + 1))
on V: (u, v) |--> (X, Y, Z) = (2*u/(u^2 + v^2 + 1), 2*v/(u^2 + v^2 + 1), -(u^2 +␣
↪v^2 - 1)/(u^2 + v^2 + 1))
```

At this stage, `Phi1` and `Phi` are fully equivalent:

```
sage: Phi1 == Phi
True
```

The map acts on points:

```
sage: np = M.point((0,0), chart=c_uv)  # the North pole
sage: Phi(np)
Point on the 3-dimensional topological manifold R^3
sage: Phi(np).coord()  # Cartesian coordinates
(0, 0, 1)
sage: sp = M.point((0,0), chart=c_xy)  # the South pole
sage: Phi(sp).coord()  # Cartesian coordinates
(0, 0, -1)
```

The test suite is passed:

```
sage: TestSuite(Phi).run()
sage: TestSuite(Phi1).run()
```

Continuous maps can be composed by means of the operator `*`. Let us introduce the map $\mathbf{R}^3 \to \mathbf{R}^2$ corresponding to the projection from the point $(X, Y, Z) = (0, 0, 1)$ onto the equatorial plane $Z = 0$:

```
sage: P = Manifold(2, 'R^2', latex_name=r'\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 $S^2 \to \mathbf{R}^2$:

```
sage: ster = Psi * Phi ; ster
Continuous map from the 2-dimensional topological manifold S^2
 to the 2-dimensional topological manifold R^2
```

Let us test on the South pole (`sp`) that `ster` is indeed the composite of `Psi` and `Phi`:

```
sage: ster(sp) == Psi(Phi(sp))
True
```

Actually `ster` is the stereographic projection from the North pole, as its coordinate expression reveals:

```
sage: ster.display()
S^2 --> R^2
on U: (x, y) |--> (xP, yP) = (x, y)
on V: (u, v) |--> (xP, yP) = (u/(u^2 + v^2), v/(u^2 + v^2))
```

If the codomain of a continuous map is 1-dimensional, the map can be defined by a single symbolic expression for each pair of charts and not by a list/tuple with a single element:

```
sage: N = Manifold(1, 'N', structure='topological')
sage: c_N = N.chart('X')
sage: Phi = M.continuous_map(N, {(c_xy, c_N): x^2+y^2,
....:                            (c_uv, c_N): 1/(u^2+v^2)})

sage: Psi = M.continuous_map(N, {(c_xy, c_N): [x^2+y^2],
....:                            (c_uv, c_N): [1/(u^2+v^2)]})
```

```
sage: Phi == Psi
True
```

Next we construct an example of continuous map $\mathbf{R} \to \mathbf{R}^2$:

```
sage: R = Manifold(1, 'R', structure='topological')  # field R
sage: T.<t> = R.chart()  # canonical chart on R
sage: R2 = Manifold(2, 'R^2', structure='topological')  # R^2
sage: c_xy.<x,y> = R2.chart() # Cartesian coordinates on R^2
sage: Phi = R.continuous_map(R2, [cos(t), sin(t)], name='Phi'); Phi
Continuous map Phi from the 1-dimensional topological manifold R
 to the 2-dimensional topological manifold R^2
sage: Phi.parent()
Set of Morphisms from 1-dimensional topological manifold R
 to 2-dimensional topological manifold R^2
 in Category of manifolds over Real Field with 53 bits of precision
sage: Phi.parent() is Hom(R, R2)
True
sage: Phi.display()
Phi: R --> R^2
   t |--> (x, y) = (cos(t), sin(t))
```

An example of homeomorphism between the unit open disk and the Euclidean plane $\mathbf{R}^2$:

```
sage: D = R2.open_subset('D', coord_def={c_xy: x^2+y^2<1}) # the open unit disk
sage: Phi = D.homeomorphism(R2, [x/sqrt(1-x^2-y^2), y/sqrt(1-x^2-y^2)],
....:                          name='Phi', latex_name=r'\Phi')
sage: Phi
Homeomorphism Phi from the Open subset D of the 2-dimensional
 topological manifold R^2 to the 2-dimensional topological manifold R^2
sage: Phi.parent()
Set of Morphisms from Open subset D of the 2-dimensional topological
 manifold R^2 to 2-dimensional topological manifold R^2 in Join of
 Category of subobjects of sets and Category of manifolds over Real
 Field with 53 bits of precision
sage: Phi.parent() is Hom(D, R2)
True
sage: Phi.display()
Phi: D --> R^2
   (x, y) |--> (x, y) = (x/sqrt(-x^2 - y^2 + 1), y/sqrt(-x^2 - y^2 + 1))
```

The image of a point:

```
sage: p = D.point((1/2,0))
sage: q = Phi(p) ; q
Point on the 2-dimensional topological manifold R^2
sage: q.coord()
(1/3*sqrt(3), 0)
```

The inverse homeomorphism is computed by `inverse()` :

```
sage: Phi.inverse()
Homeomorphism Phi^(-1) from the 2-dimensional topological manifold R^2
 to the Open subset D of the 2-dimensional topological manifold R^2
sage: Phi.inverse().display()
Phi^(-1): R^2 --> D
   (x, y) |--> (x, y) = (x/sqrt(x^2 + y^2 + 1), y/sqrt(x^2 + y^2 + 1))
```

Equivalently, one may use the notations `^(-1)` or `~` to get the inverse:

```
sage: Phi^(-1) is Phi.inverse()
True
sage: ~Phi is Phi.inverse()
True
```

Check that `~Phi` is indeed the inverse of `Phi`:

```
sage: (~Phi)(q) == p
True
sage: Phi * ~Phi == R2.identity_map()
True
sage: ~Phi * Phi == D.identity_map()
True
```

The coordinate expression of the inverse homeomorphism:

```
sage: (~Phi).display()
Phi^(-1): R^2 --> D
   (x, y) |--> (x, y) = (x/sqrt(x^2 + y^2 + 1), y/sqrt(x^2 + y^2 + 1))
```

A special case of homeomorphism: the identity map of the open unit disk:

```
sage: id = D.identity_map() ; id
Identity map Id_D of the Open subset D of the 2-dimensional topological
 manifold R^2
sage: latex(id)
\mathrm{Id}_{D}
sage: id.parent()
Set of Morphisms from Open subset D of the 2-dimensional topological
 manifold R^2 to Open subset D of the 2-dimensional topological
 manifold R^2 in Join of Category of subobjects of sets and Category of
 manifolds over Real Field with 53 bits of precision
sage: id.parent() is Hom(D, D)
True
sage: id is Hom(D,D).one()  # the identity element of the monoid Hom(D,D)
True
```

The identity map acting on a point:

```
sage: id(p)
Point on the 2-dimensional topological manifold R^2
sage: id(p) == p
True
sage: id(p) is p
True
```

The coordinate expression of the identity map:

```
sage: id.display()
Id_D: D --> D
   (x, y) |--> (x, y)
```

The identity map is its own inverse:

```
sage: id^(-1) is id
True
```

```
sage: ~id is id
True
```

add_expr (*chart1*, *chart2*, *coord_functions*)

Set a new coordinate representation of `self` .

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

INPUT:

- `chart1` – chart for the coordinates on the map's domain
- `chart2` – chart for the coordinates on the map's codomain
- `coord_functions` – the coordinate symbolic expression of the map in the above charts: list (or tuple) of the coordinates of the image expressed in terms of the coordinates of the considered point; if the dimension of the arrival manifold is 1, a single coordinate expression can be passed instead of a tuple with a single element

Warning: If the map has already expressions in other charts, it is the user's responsibility to make sure that the expression to be added is consistent with them.

EXAMPLES:

Polar representation of a planar rotation initally defined in Cartesian coordinates:

```
sage: M = Manifold(2, 'R^2', latex_name=r'\RR^2', structure='topological')  #␣
↪the Euclidean plane R^2
sage: c_xy.<x,y> = M.chart() # Cartesian coordinate on R^2
sage: U = M.open_subset('U', coord_def={c_xy: (y!=0, x<0)}) # the complement␣
↪of the segment y=0 and x>0
sage: c_cart = c_xy.restrict(U) # Cartesian coordinates on U
sage: c_spher.<r,ph> = U.chart(r'r:(0,+oo) ph:(0,2*pi):\phi') # spherical␣
↪coordinates on U
```

We construct the links between spherical coordinates and Cartesian ones:

```
sage: ch_cart_spher = c_cart.transition_map(c_spher, [sqrt(x*x+y*y),␣
↪atan2(y,x)])
sage: ch_cart_spher.set_inverse(r*cos(ph), r*sin(ph), verbose=True)
Check of the inverse coordinate transformation:
   x == x
   y == y
   r == r
   ph == arctan2(r*sin(ph), r*cos(ph))
sage: rot = U.continuous_map(U, ((x - sqrt(3)*y)/2, (sqrt(3)*x + y)/2),
....:                         name='R')
sage: rot.display(c_cart, c_cart)
R: U --> U
   (x, y) |--> (-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)
```

If we calculate the expression in terms of spherical coordinates, via the method `display()` , we notice some difficulties in `arctan2` simplifications:

```
sage: rot.display(c_spher, c_spher)
R: U --> U
   (r, ph) |--> (r, arctan2(1/2*(sqrt(3)*cos(ph) + sin(ph))*r, -1/
↪2*(sqrt(3)*sin(ph) - cos(ph))*r))
```

Therefore, we use the method *add_expr()* to set the spherical-coordinate expression by hand:

```
sage: rot.add_expr(c_spher, c_spher, (r, ph+pi/3))
sage: rot.display(c_spher, c_spher)
R: U --> U
   (r, ph) |--> (r, 1/3*pi + ph)
```

The call to *add_expr()* has not deleted the expression in terms of Cartesian coordinates, as we can check by printing the internal dictionary `_coord_expression`, which stores the various internal representations of the continuous map:

```
sage: rot._coord_expression # random (dictionary output)
{(Chart (U, (x, y)), Chart (U, (x, y))):
 Coordinate functions (-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)
  on the Chart (U, (x, y)),
 (Chart (U, (r, ph)), Chart (U, (r, ph))):
  Coordinate functions (r, 1/3*pi + ph) on the Chart (U, (r, ph))}
```

If, on the contrary, we use *set_expr()*, the expression in Cartesian coordinates is lost:

```
sage: rot.set_expr(c_spher, c_spher, (r, ph+pi/3))
sage: rot._coord_expression
{(Chart (U, (r, ph)), Chart (U, (r, ph))):
 Coordinate functions (r, 1/3*pi + ph) on the Chart (U, (r, ph))}
```

It is recovered (thanks to the known change of coordinates) by a call to *display()*:

```
sage: rot.display(c_cart, c_cart)
R: U --> U
   (x, y) |--> (-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)

sage: rot._coord_expression # random (dictionary output)
{(Chart (U, (x, y)), Chart (U, (x, y))):
 Coordinate functions (-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)
  on the Chart (U, (x, y)),
 (Chart (U, (r, ph)), Chart (U, (r, ph))):
 Coordinate functions (r, 1/3*pi + ph) on the Chart (U, (r, ph))}
```

The rotation can be applied to a point by means of either coordinate system:

```
sage: p = M.point((1,2))   #  p defined by its Cartesian coord.
sage: q = rot(p)   # q is computed by means of Cartesian coord.
sage: p1 = M.point((sqrt(5), arctan(2)), chart=c_spher) # p1 is defined only␣
↪in terms of c_spher
sage: q1 = rot(p1) # computation by means of spherical coordinates
sage: q1 == q
True
```

add_expression (*chart1*, *chart2*, *coord_functions*)

Set a new coordinate representation of `self`.

The previous expressions with respect to other charts are kept. To clear them, use *set_expr()* instead.

INPUT:

- `chart1` – chart for the coordinates on the map's domain
- `chart2` – chart for the coordinates on the map's codomain

- `coord_functions` – the coordinate symbolic expression of the map in the above charts: list (or tuple) of the coordinates of the image expressed in terms of the coordinates of the considered point; if the dimension of the arrival manifold is 1, a single coordinate expression can be passed instead of a tuple with a single element

Warning: If the map has already expressions in other charts, it is the user's responsibility to make sure that the expression to be added is consistent with them.

EXAMPLES:

Polar representation of a planar rotation initally defined in Cartesian coordinates:

```
sage: M = Manifold(2, 'R^2', latex_name=r'\RR^2', structure='topological')  #␣
↪the Euclidean plane R^2
sage: c_xy.<x,y> = M.chart() # Cartesian coordinate on R^2
sage: U = M.open_subset('U', coord_def={c_xy: (y!=0, x<0)}) # the complement␣
↪of the segment y=0 and x>0
sage: c_cart = c_xy.restrict(U) # Cartesian coordinates on U
sage: c_spher.<r,ph> = U.chart(r'r:(0,+oo) ph:(0,2*pi):\phi') # spherical␣
↪coordinates on U
```

We construct the links between spherical coordinates and Cartesian ones:

```
sage: ch_cart_spher = c_cart.transition_map(c_spher, [sqrt(x*x+y*y),␣
↪atan2(y,x)])
sage: ch_cart_spher.set_inverse(r*cos(ph), r*sin(ph), verbose=True)
Check of the inverse coordinate transformation:
   x == x
   y == y
   r == r
   ph == arctan2(r*sin(ph), r*cos(ph))
sage: rot = U.continuous_map(U, ((x - sqrt(3)*y)/2, (sqrt(3)*x + y)/2),
....:                          name='R')
sage: rot.display(c_cart, c_cart)
R: U --> U
   (x, y) |--> (-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)
```

If we calculate the expression in terms of spherical coordinates, via the method *display()*, we notice some difficulties in `arctan2` simplifications:

```
sage: rot.display(c_spher, c_spher)
R: U --> U
   (r, ph) |--> (r, arctan2(1/2*(sqrt(3)*cos(ph) + sin(ph))*r, -1/
↪2*(sqrt(3)*sin(ph) - cos(ph))*r))
```

Therefore, we use the method *add_expr()* to set the spherical-coordinate expression by hand:

```
sage: rot.add_expr(c_spher, c_spher, (r, ph+pi/3))
sage: rot.display(c_spher, c_spher)
R: U --> U
   (r, ph) |--> (r, 1/3*pi + ph)
```

The call to *add_expr()* has not deleted the expression in terms of Cartesian coordinates, as we can check by printing the internal dictionary `_coord_expression`, which stores the various internal representations of the continuous map:

```
sage: rot._coord_expression # random (dictionary output)
{(Chart (U, (x, y)), Chart (U, (x, y))):
 Coordinate functions (-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)
  on the Chart (U, (x, y)),
 (Chart (U, (r, ph)), Chart (U, (r, ph))):
  Coordinate functions (r, 1/3*pi + ph) on the Chart (U, (r, ph))}
```

If, on the contrary, we use `set_expr()` , the expression in Cartesian coordinates is lost:

```
sage: rot.set_expr(c_spher, c_spher, (r, ph+pi/3))
sage: rot._coord_expression
{(Chart (U, (r, ph)), Chart (U, (r, ph))):
 Coordinate functions (r, 1/3*pi + ph) on the Chart (U, (r, ph))}
```

It is recovered (thanks to the known change of coordinates) by a call to `display()` :

```
sage: rot.display(c_cart, c_cart)
R: U --> U
   (x, y) |--> (-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)

sage: rot._coord_expression # random (dictionary output)
{(Chart (U, (x, y)), Chart (U, (x, y))):
 Coordinate functions (-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)
  on the Chart (U, (x, y)),
 (Chart (U, (r, ph)), Chart (U, (r, ph))):
 Coordinate functions (r, 1/3*pi + ph) on the Chart (U, (r, ph))}
```

The rotation can be applied to a point by means of either coordinate system:

```
sage: p = M.point((1,2))  #  p defined by its Cartesian coord.
sage: q = rot(p)  # q is computed by means of Cartesian coord.
sage: p1 = M.point((sqrt(5), arctan(2)), chart=c_spher) # p1 is defined only␣
↪in terms of c_spher
sage: q1 = rot(p1) # computation by means of spherical coordinates
sage: q1 == q
True
```

coord_functions (*chart1=None*, *chart2=None*)

Return the functions of the coordinates representing `self` in a given pair of charts.

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

INPUT:

- `chart1` – (default: `None`) chart on the domain of `self` ; if `None` , the domain's default chart is assumed
- `chart2` – (default: `None`) chart on the codomain of `self` ; if `None` , the codomain's default chart is assumed

OUTPUT:

- a `MultiCoordFunction` representing the continuous map in the above two charts

EXAMPLES:

Continuous map from a 2-dimensional manifold to a 3-dimensional one:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: N = Manifold(3, 'N', structure='topological')
sage: c_uv.<u,v> = M.chart()
sage: c_xyz.<x,y,z> = N.chart()
sage: Phi = M.continuous_map(N, (u*v, u/v, u+v), name='Phi',
....:                         latex_name=r'\Phi')
sage: Phi.display()
Phi: M --> N
   (u, v) |--> (x, y, z) = (u*v, u/v, u + v)
sage: Phi.coord_functions(c_uv, c_xyz)
Coordinate functions (u*v, u/v, u + v) on the Chart (M, (u, v))
sage: Phi.coord_functions() # equivalent to above since 'uv' and 'xyz' are␣
↪default charts
Coordinate functions (u*v, u/v, u + v) on the Chart (M, (u, v))
sage: type(Phi.coord_functions())
<class 'sage.manifolds.coord_func.MultiCoordFunction'>
```

Coordinate representation in other charts:

```
sage: c_UV.<U,V> = M.chart()  # new chart on M
sage: ch_uv_UV = c_uv.transition_map(c_UV, [u-v, u+v])
sage: ch_uv_UV.inverse()(U,V)
(1/2*U + 1/2*V, -1/2*U + 1/2*V)
sage: c_XYZ.<X,Y,Z> = N.chart() # new chart on N
sage: ch_xyz_XYZ = c_xyz.transition_map(c_XYZ,
....:                                    [2*x-3*y+z, y+z-x, -x+2*y-z])
sage: ch_xyz_XYZ.inverse()(X,Y,Z)
(3*X + Y + 4*Z, 2*X + Y + 3*Z, X + Y + Z)
sage: Phi.coord_functions(c_UV, c_xyz)
Coordinate functions (-1/4*U^2 + 1/4*V^2, -(U + V)/(U - V), V) on
 the Chart (M, (U, V))
sage: Phi.coord_functions(c_uv, c_XYZ)
Coordinate functions (((2*u + 1)*v^2 + u*v - 3*u)/v,
 -((u - 1)*v^2 - u*v - u)/v, -((u + 1)*v^2 + u*v - 2*u)/v) on the
 Chart (M, (u, v))
sage: Phi.coord_functions(c_UV, c_XYZ)
Coordinate functions
 (-1/2*(U^3 - (U - 2)*V^2 + V^3 - (U^2 + 2*U + 6)*V - 6*U)/(U - V),
  1/4*(U^3 - (U + 4)*V^2 + V^3 - (U^2 - 4*U + 4)*V - 4*U)/(U - V),
  1/4*(U^3 - (U - 4)*V^2 + V^3 - (U^2 + 4*U + 8)*V - 8*U)/(U - V))
 on the Chart (M, (U, V))
```

Coordinate representation with respect to a subchart in the domain:

```
sage: A = M.open_subset('A', coord_def={c_uv: u>0})
sage: Phi.coord_functions(c_uv.restrict(A), c_xyz)
Coordinate functions (u*v, u/v, u + v) on the Chart (A, (u, v))
```

Coordinate representation with respect to a superchart in the codomain:

```
sage: B = N.open_subset('B', coord_def={c_xyz: x<0})
sage: c_xyz_B = c_xyz.restrict(B)
sage: Phi1 = M.continuous_map(B, {(c_uv, c_xyz_B): (u*v, u/v, u+v)})
sage: Phi1.coord_functions(c_uv, c_xyz_B) # definition charts
Coordinate functions (u*v, u/v, u + v) on the Chart (M, (u, v))
sage: Phi1.coord_functions(c_uv, c_xyz) # c_xyz = superchart of c_xyz_B
Coordinate functions (u*v, u/v, u + v) on the Chart (M, (u, v))
```

Coordinate representation with respect to a pair `(subchart,superchart)`:

```
sage: Phi1.coord_functions(c_uv.restrict(A), c_xyz)
Coordinate functions (u*v, u/v, u + v) on the Chart (A, (u, v))
```

disp (*chart1=None*, *chart2=None*)

Display the expression of `self` in one or more pair of charts.

If the expression is not known already, it is computed from some expression in other charts by means of change-of-coordinate formulas.

INPUT:

- `chart1` – (default: `None`) chart on the domain of `self`; if `None`, the display is performed on all the charts on the domain in which the map is known or computable via some change of coordinates
- `chart2` – (default: `None`) chart on the codomain of `self`; if `None`, the display is performed on all the charts on the codomain in which the map is known or computable via some change of coordinates

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

EXAMPLES:

Standard embedding of the sphere S^2 in $\mathbf{R}^3$:

```
sage: M = Manifold(2, 'S^2', structure='topological') # the 2-dimensional␣
↪sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V)   # S^2 is the union of U and V
sage: N = Manifold(3, 'R^3', latex_name=r'\RR^3', structure='topological')  #␣
↪R^3
sage: c_cart.<X,Y,Z> = N.chart()  # Cartesian coordinates on R^3
sage: Phi = M.continuous_map(N,
....:   {(c_xy, c_cart): [2*x/(1+x^2+y^2), 2*y/(1+x^2+y^2), (x^2+y^2-1)/(1+x^
↪2+y^2)],
....:    (c_uv, c_cart): [2*u/(1+u^2+v^2), 2*v/(1+u^2+v^2), (1-u^2-v^2)/(1+u^
↪2+v^2)]},
....:   name='Phi', latex_name=r'\Phi')
sage: Phi.display(c_xy, c_cart)
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2␣
↪+ y^2 - 1)/(x^2 + y^2 + 1))
sage: Phi.display(c_uv, c_cart)
Phi: S^2 --> R^3
on V: (u, v) |--> (X, Y, Z) = (2*u/(u^2 + v^2 + 1), 2*v/(u^2 + v^2 + 1), -(u^
↪2 + v^2 - 1)/(u^2 + v^2 + 1))
```

The LaTeX output:

```
sage: latex(Phi.display(c_xy, c_cart))
\begin{array}{llcl} \Phi:& S^2 & \longrightarrow & \RR^3
 \\ \mbox{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}
```

If the argument `chart2` is not specified, the display is performed on all the charts on the codomain in which the map is known or computable via some change of coordinates (here only one chart: `c_cart`):

```
sage: Phi.display(c_xy)
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2␣
↪+ y^2 - 1)/(x^2 + y^2 + 1))
```

Similarly, if the argument `chart1` is omitted, the display is performed on all the charts on the domain of `Phi` in which the map is known or computable via some change of coordinates:

```
sage: Phi.display(chart2=c_cart)
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2␣
↪+ y^2 - 1)/(x^2 + y^2 + 1))
on V: (u, v) |--> (X, Y, Z) = (2*u/(u^2 + v^2 + 1), 2*v/(u^2 + v^2 + 1), -(u^
↪2 + v^2 - 1)/(u^2 + v^2 + 1))
```

If neither `chart1` nor `chart2` is specified, the display is performed on all the pair of charts in which `Phi` is known or computable via some change of coordinates:

```
sage: Phi.display()
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2␣
↪+ y^2 - 1)/(x^2 + y^2 + 1))
on V: (u, v) |--> (X, Y, Z) = (2*u/(u^2 + v^2 + 1), 2*v/(u^2 + v^2 + 1), -(u^
↪2 + v^2 - 1)/(u^2 + v^2 + 1))
```

If a chart covers entirely the map's domain, the mention "on ..." is omitted:

```
sage: Phi.restrict(U).display()
Phi: U --> R^3
   (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2 +␣
↪y^2 - 1)/(x^2 + y^2 + 1))
```

A shortcut of `display()` is `disp()`:

```
sage: Phi.disp()
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2␣
↪+ y^2 - 1)/(x^2 + y^2 + 1))
on V: (u, v) |--> (X, Y, Z) = (2*u/(u^2 + v^2 + 1), 2*v/(u^2 + v^2 + 1), -(u^
↪2 + v^2 - 1)/(u^2 + v^2 + 1))
```

display (*chart1=None*, *chart2=None*)

Display the expression of `self` in one or more pair of charts.

If the expression is not known already, it is computed from some expression in other charts by means of change-of-coordinate formulas.

INPUT:

- `chart1` – (default: `None`) chart on the domain of `self`; if `None`, the display is performed on all the charts on the domain in which the map is known or computable via some change of coordinates
- `chart2` – (default: `None`) chart on the codomain of `self`; if `None`, the display is performed on all the charts on the codomain in which the map is known or computable via some change of coordinates

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

EXAMPLES:

Standard embedding of the sphere S^2 in $\mathbf{R}^3$:

```
sage: M = Manifold(2, 'S^2', structure='topological') # the 2-dimensional␣
↪sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V)   # S^2 is the union of U and V
sage: N = Manifold(3, 'R^3', latex_name=r'\RR^3', structure='topological')  #␣
↪R^3
sage: c_cart.<X,Y,Z> = N.chart()  # Cartesian coordinates on R^3
sage: Phi = M.continuous_map(N,
....:   {(c_xy, c_cart): [2*x/(1+x^2+y^2), 2*y/(1+x^2+y^2), (x^2+y^2-1)/(1+x^
↪2+y^2)],
....:    (c_uv, c_cart): [2*u/(1+u^2+v^2), 2*v/(1+u^2+v^2), (1-u^2-v^2)/(1+u^
↪2+v^2)]},
....:   name='Phi', latex_name=r'\Phi')
sage: Phi.display(c_xy, c_cart)
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2␣
↪+ y^2 - 1)/(x^2 + y^2 + 1))
sage: Phi.display(c_uv, c_cart)
Phi: S^2 --> R^3
on V: (u, v) |--> (X, Y, Z) = (2*u/(u^2 + v^2 + 1), 2*v/(u^2 + v^2 + 1), -(u^
↪2 + v^2 - 1)/(u^2 + v^2 + 1))
```

The LaTeX output:

```
sage: latex(Phi.display(c_xy, c_cart))
\begin{array}{llcl} \Phi:& S^2 & \longrightarrow & \RR^3
 \\ \mbox{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}
```

If the argument `chart2` is not specified, the display is performed on all the charts on the codomain in which the map is known or computable via some change of coordinates (here only one chart: `c_cart`):

```
sage: Phi.display(c_xy)
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2␣
↪+ y^2 - 1)/(x^2 + y^2 + 1))
```

Similarly, if the argument `chart1` is omitted, the display is performed on all the charts on the domain of `Phi` in which the map is known or computable via some change of coordinates:

```
sage: Phi.display(chart2=c_cart)
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2␣
↪+ y^2 - 1)/(x^2 + y^2 + 1))
on V: (u, v) |--> (X, Y, Z) = (2*u/(u^2 + v^2 + 1), 2*v/(u^2 + v^2 + 1), -(u^
↪2 + v^2 - 1)/(u^2 + v^2 + 1))
```

If neither `chart1` nor `chart2` is specified, the display is performed on all the pair of charts in which `Phi` is known or computable via some change of coordinates:

```
sage: Phi.display()
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2␣
↪+ y^2 - 1)/(x^2 + y^2 + 1))
on V: (u, v) |--> (X, Y, Z) = (2*u/(u^2 + v^2 + 1), 2*v/(u^2 + v^2 + 1), -(u^
↪2 + v^2 - 1)/(u^2 + v^2 + 1))
```

If a chart covers entirely the map's domain, the mention "on ..." is omitted:

```
sage: Phi.restrict(U).display()
Phi: U --> R^3
   (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2 +␣
↪y^2 - 1)/(x^2 + y^2 + 1))
```

A shortcut of `display()` is `disp()` :

```
sage: Phi.disp()
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1), (x^2␣
↪+ y^2 - 1)/(x^2 + y^2 + 1))
on V: (u, v) |--> (X, Y, Z) = (2*u/(u^2 + v^2 + 1), 2*v/(u^2 + v^2 + 1), -(u^
↪2 + v^2 - 1)/(u^2 + v^2 + 1))
```

expr (*chart1=None*, *chart2=None*)

Return the expression of `self` in terms of specified coordinates.

If the expression is not already known, it is computed from some known expression by means of change-of-chart formulas.

INPUT:

- `chart1` – (default: `None`) chart on the map's domain; if `None` , the domain's default chart is assumed
- `chart2` – (default: `None`) chart on the map's codomain; if `None` , the codomain's default chart is assumed

OUTPUT:

- symbolic expression representing the continuous map in the above two charts

EXAMPLES:

Continuous map from a 2-dimensional manifold to a 3-dimensional one:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: N = Manifold(3, 'N', structure='topological')
sage: c_uv.<u,v> = M.chart()
sage: c_xyz.<x,y,z> = N.chart()
sage: Phi = M.continuous_map(N, (u*v, u/v, u+v), name='Phi',
....:                        latex_name=r'\Phi')
sage: Phi.display()
Phi: M --> N
   (u, v) |--> (x, y, z) = (u*v, u/v, u + v)
sage: Phi.expr(c_uv, c_xyz)
(u*v, u/v, u + v)
sage: Phi.expr()  # equivalent to above since 'uv' and 'xyz' are default␣
↪charts
```

```
(u*v, u/v, u + v)
sage: type(Phi.expr()[0])
<type 'sage.symbolic.expression.Expression'>
```

Expressions in other charts:

```
sage: c_UV.<U,V> = M.chart()  # new chart on M
sage: ch_uv_UV = c_uv.transition_map(c_UV, [u-v, u+v])
sage: ch_uv_UV.inverse()(U,V)
(1/2*U + 1/2*V, -1/2*U + 1/2*V)
sage: c_XYZ.<X,Y,Z> = N.chart() # new chart on N
sage: ch_xyz_XYZ = c_xyz.transition_map(c_XYZ,
....:                                   [2*x-3*y+z, y+z-x, -x+2*y-z])
sage: ch_xyz_XYZ.inverse()(X,Y,Z)
(3*X + Y + 4*Z, 2*X + Y + 3*Z, X + Y + Z)
sage: Phi.expr(c_UV, c_xyz)
(-1/4*U^2 + 1/4*V^2, -(U + V)/(U - V), V)
sage: Phi.expr(c_uv, c_XYZ)
(((2*u + 1)*v^2 + u*v - 3*u)/v,
 -((u - 1)*v^2 - u*v - u)/v,
 -((u + 1)*v^2 + u*v - 2*u)/v)
sage: Phi.expr(c_UV, c_XYZ)
 (-1/2*(U^3 - (U - 2)*V^2 + V^3 - (U^2 + 2*U + 6)*V - 6*U)/(U - V),
  1/4*(U^3 - (U + 4)*V^2 + V^3 - (U^2 - 4*U + 4)*V - 4*U)/(U - V),
  1/4*(U^3 - (U - 4)*V^2 + V^3 - (U^2 + 4*U + 8)*V - 8*U)/(U - V))
```

A rotation in some Euclidean plane:

```
sage: M = Manifold(2, 'M', structure='topological') # the plane (minus a␣
↪segment to have global regular spherical coordinates)
sage: c_spher.<r,ph> = M.chart(r'r:(0,+oo) ph:(0,2*pi):\phi') # spherical␣
↪coordinates on the plane
sage: rot = M.continuous_map(M, (r, ph+pi/3), name='R') # pi/3 rotation␣
↪around r=0
sage: rot.expr()
(r, 1/3*pi + ph)
```

Expression of the rotation in terms of Cartesian coordinates:

```
sage: c_cart.<x,y> = M.chart() # Declaration of Cartesian coordinates
sage: ch_spher_cart = c_spher.transition_map(c_cart,
....:                 [r*cos(ph), r*sin(ph)]) # relation to spherical␣
↪coordinates
sage: ch_spher_cart.set_inverse(sqrt(x^2+y^2), atan2(y,x), verbose=True)
Check of the inverse coordinate transformation:
   r == r
   ph == arctan2(r*sin(ph), r*cos(ph))
   x == x
   y == y
sage: rot.expr(c_cart, c_cart)
(-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)
```

expression (*chart1=None*, *chart2=None*)

Return the expression of `self` in terms of specified coordinates.

If the expression is not already known, it is computed from some known expression by means of change-of-chart formulas.

INPUT:

- `chart1` – (default: `None`) chart on the map's domain; if `None` , the domain's default chart is assumed
- `chart2` – (default: `None`) chart on the map's codomain; if `None` , the codomain's default chart is assumed

OUTPUT:

- symbolic expression representing the continuous map in the above two charts

EXAMPLES:

Continuous map from a 2-dimensional manifold to a 3-dimensional one:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: N = Manifold(3, 'N', structure='topological')
sage: c_uv.<u,v> = M.chart()
sage: c_xyz.<x,y,z> = N.chart()
sage: Phi = M.continuous_map(N, (u*v, u/v, u+v), name='Phi',
....:                        latex_name=r'\Phi')
sage: Phi.display()
Phi: M --> N
   (u, v) |--> (x, y, z) = (u*v, u/v, u + v)
sage: Phi.expr(c_uv, c_xyz)
(u*v, u/v, u + v)
sage: Phi.expr()  # equivalent to above since 'uv' and 'xyz' are default␣
↪charts
(u*v, u/v, u + v)
sage: type(Phi.expr()[0])
<type 'sage.symbolic.expression.Expression'>
```

Expressions in other charts:

```
sage: c_UV.<U,V> = M.chart()  # new chart on M
sage: ch_uv_UV = c_uv.transition_map(c_UV, [u-v, u+v])
sage: ch_uv_UV.inverse()(U,V)
(1/2*U + 1/2*V, -1/2*U + 1/2*V)
sage: c_XYZ.<X,Y,Z> = N.chart() # new chart on N
sage: ch_xyz_XYZ = c_xyz.transition_map(c_XYZ,
....:                                   [2*x-3*y+z, y+z-x, -x+2*y-z])
sage: ch_xyz_XYZ.inverse()(X,Y,Z)
(3*X + Y + 4*Z, 2*X + Y + 3*Z, X + Y + Z)
sage: Phi.expr(c_UV, c_xyz)
(-1/4*U^2 + 1/4*V^2, -(U + V)/(U - V), V)
sage: Phi.expr(c_uv, c_XYZ)
(((2*u + 1)*v^2 + u*v - 3*u)/v,
 -((u - 1)*v^2 - u*v - u)/v,
 -((u + 1)*v^2 + u*v - 2*u)/v)
sage: Phi.expr(c_UV, c_XYZ)
 (-1/2*(U^3 - (U - 2)*V^2 + V^3 - (U^2 + 2*U + 6)*V - 6*U)/(U - V),
  1/4*(U^3 - (U + 4)*V^2 + V^3 - (U^2 - 4*U + 4)*V - 4*U)/(U - V),
  1/4*(U^3 - (U - 4)*V^2 + V^3 - (U^2 + 4*U + 8)*V - 8*U)/(U - V))
```

A rotation in some Euclidean plane:

```
sage: M = Manifold(2, 'M', structure='topological') # the plane (minus a␣
↪segment to have global regular spherical coordinates)
sage: c_spher.<r,ph> = M.chart(r'r:(0,+oo) ph:(0,2*pi):\phi') # spherical␣
↪coordinates on the plane
sage: rot = M.continuous_map(M, (r, ph+pi/3), name='R') # pi/3 rotation␣
↪around r=0
```

```
sage: rot.expr()
(r, 1/3*pi + ph)
```

Expression of the rotation in terms of Cartesian coordinates:

```
sage: c_cart.<x,y> = M.chart() # Declaration of Cartesian coordinates
sage: ch_spher_cart = c_spher.transition_map(c_cart,
....:                 [r*cos(ph), r*sin(ph)]) # relation to spherical␣
↪coordinates
sage: ch_spher_cart.set_inverse(sqrt(x^2+y^2), atan2(y,x), verbose=True)
Check of the inverse coordinate transformation:
   r == r
   ph == arctan2(r*sin(ph), r*cos(ph))
   x == x
   y == y
sage: rot.expr(c_cart, c_cart)
(-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)
```

inverse ( )

Return the inverse of `self` if it is an isomorphism.

OUTPUT:

- the inverse isomorphism

EXAMPLES:

The inverse of a rotation in the Euclidean plane:

```
sage: M = Manifold(2, 'R^2', latex_name=r'\RR^2', structure='topological')
sage: c_cart.<x,y> = M.chart()
sage: # A pi/3 rotation around the origin:
sage: rot = M.homeomorphism(M, ((x - sqrt(3)*y)/2, (sqrt(3)*x + y)/2),
....:                       name='R')
sage: rot.inverse()
Homeomorphism R^(-1) of the 2-dimensional topological manifold R^2
sage: rot.inverse().display()
R^(-1): R^2 --> R^2
   (x, y) |--> (1/2*sqrt(3)*y + 1/2*x, -1/2*sqrt(3)*x + 1/2*y)
```

Checking that applying successively the homeomorphism and its inverse results in the identity:

```
sage: (a, b) = var('a b')
sage: p = M.point((a,b)) # a generic point on M
sage: q = rot(p)
sage: p1 = rot.inverse()(q)
sage: p1 == p
True
```

The result is cached:

```
sage: rot.inverse() is rot.inverse()
True
```

The notations `^(-1)` or `~` can also be used for the inverse:

```
sage: rot^(-1) is rot.inverse()
True
```

```
sage: ~rot is rot.inverse()
True
```

An example with multiple charts: the equatorial symmetry on the 2-sphere:

```
sage: M = Manifold(2, 'M', structure='topological') # the 2-dimensional␣
↪sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V)   # S^2 is the union of U and V
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                               intersection_name='W',
....:                               restrictions1=x^2+y^2!=0,
....:                               restrictions2=u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: s = M.homeomorphism(M, {(c_xy, c_uv): [x, y], (c_uv, c_xy): [u, v]},
....:                    name='s')
sage: s.display()
s: M --> M
on U: (x, y) |--> (u, v) = (x, y)
on V: (u, v) |--> (x, y) = (u, v)
sage: si = s.inverse(); si
Homeomorphism s^(-1) of the 2-dimensional topological manifold M
sage: si.display()
s^(-1): M --> M
on U: (x, y) |--> (u, v) = (x, y)
on V: (u, v) |--> (x, y) = (u, v)
```

The equatorial symmetry is of course an involution:

```
sage: si == s
True
```

is_identity ( )

Check whether `self` is an identity map.

EXAMPLES:

Tests on continuous maps of a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: M.identity_map().is_identity()  # obviously...
True
sage: Hom(M, M).one().is_identity()  # a variant of the obvious
True
sage: a = M.continuous_map(M, coord_functions={(X,X): (x, y)})
sage: a.is_identity()
True
sage: a = M.continuous_map(M, coord_functions={(X,X): (x, y+1)})
sage: a.is_identity()
False
```

Of course, if the codomain of the map does not coincide with its domain, the outcome is `False`:

```
sage: N = Manifold(2, 'N', structure='topological')
sage: Y.<u,v> = N.chart()
sage: a = M.continuous_map(N, {(X,Y): (x, y)})
sage: a.display()
M --> N
   (x, y) |--> (u, v) = (x, y)
sage: a.is_identity()
False
```

restrict (*subdomain*, *subcodomain=None*)

Restriction of `self` to some open subset of its domain of definition.

INPUT:

- `subdomain` – *TopologicalManifold* ; an open subset of the domain of `self`
- `subcodomain` – (default: `None`) an open subset of the codomain of `self` ; if `None` , the codomain of `self` is assumed

OUTPUT:

- a *ContinuousMap* that is the restriction of `self` to `subdomain`

EXAMPLES:

Restriction to an annulus of a homeomorphism between the open unit disk and $\mathbf{R}^2$:

```
sage: M = Manifold(2, 'R^2', structure='topological')  # R^2
sage: c_xy.<x,y> = M.chart()  # Cartesian coord. on R^2
sage: D = M.open_subset('D', coord_def={c_xy: x^2+y^2<1}) # the open unit disk
sage: Phi = D.continuous_map(M, [x/sqrt(1-x^2-y^2), y/sqrt(1-x^2-y^2)],
....:                         name='Phi', latex_name=r'\Phi')
sage: Phi.display()
Phi: D --> R^2
   (x, y) |--> (x, y) = (x/sqrt(-x^2 - y^2 + 1), y/sqrt(-x^2 - y^2 + 1))
sage: c_xy_D = c_xy.restrict(D)
sage: U = D.open_subset('U', coord_def={c_xy_D: x^2+y^2>1/2}) # the annulus 1/
↪2 < r < 1
sage: Phi.restrict(U)
Continuous map Phi
 from the Open subset U of the 2-dimensional topological manifold R^2
 to the 2-dimensional topological manifold R^2
sage: Phi.restrict(U).parent()
Set of Morphisms
 from Open subset U of the 2-dimensional topological manifold R^2
 to 2-dimensional topological manifold R^2
 in Join of Category of subobjects of sets
    and Category of manifolds over Real Field with 53 bits of precision
sage: Phi.domain()
Open subset D of the 2-dimensional topological manifold R^2
sage: Phi.restrict(U).domain()
Open subset U of the 2-dimensional topological manifold R^2
sage: Phi.restrict(U).display()
Phi: U --> R^2
   (x, y) |--> (x, y) = (x/sqrt(-x^2 - y^2 + 1), y/sqrt(-x^2 - y^2 + 1))
```

The result is cached:

```
sage: Phi.restrict(U) is Phi.restrict(U)
True
```

The restriction of the identity map:

```
sage: id = D.identity_map() ; id
Identity map Id_D of the Open subset D of the 2-dimensional
 topological manifold R^2
sage: id.restrict(U)
Identity map Id_U of the Open subset U of the 2-dimensional
 topological manifold R^2
sage: id.restrict(U) is U.identity_map()
True
```

The codomain can be restricted (i.e. made tighter):

```
sage: Phi = D.continuous_map(M, [x/sqrt(1+x^2+y^2), y/sqrt(1+x^2+y^2)])
sage: Phi
Continuous map from
 the Open subset D of the 2-dimensional topological manifold R^2
 to the 2-dimensional topological manifold R^2
sage: Phi.restrict(D, subcodomain=D)
Continuous map from the Open subset D of the 2-dimensional
 topological manifold R^2 to itself
```

set_expr (*chart1*, *chart2*, *coord_functions*)

Set a new coordinate representation of `self`.

The expressions with respect to other charts are deleted, in order to avoid any inconsistency. To keep them, use *add_expr()* instead.

INPUT:

- `chart1` – chart for the coordinates on the domain of `self`
- `chart2` – chart for the coordinates on the codomain of `self`
- `coord_functions` – the coordinate symbolic expression of the map in the above charts: list (or tuple) of the coordinates of the image expressed in terms of the coordinates of the considered point; if the dimension of the arrival manifold is 1, a single coordinate expression can be passed instead of a tuple with a single element

EXAMPLES:

Polar representation of a planar rotation initally defined in Cartesian coordinates:

```
sage: M = Manifold(2, 'R^2', latex_name=r'\RR^2', structure='topological')  #␣
↪the Euclidean plane R^2
sage: c_xy.<x,y> = M.chart() # Cartesian coordinate on R^2
sage: U = M.open_subset('U', coord_def={c_xy: (y!=0, x<0)}) # the complement␣
↪of the segment y=0 and x>0
sage: c_cart = c_xy.restrict(U) # Cartesian coordinates on U
sage: c_spher.<r,ph> = U.chart(r'r:(0,+oo) ph:(0,2*pi):\phi') # spherical␣
↪coordinates on U
sage: # Links between spherical coordinates and Cartesian ones:
sage: ch_cart_spher = c_cart.transition_map(c_spher,
....:                                        [sqrt(x*x+y*y), atan2(y,x)])
sage: ch_cart_spher.set_inverse(r*cos(ph), r*sin(ph), verbose=True)
Check of the inverse coordinate transformation:
   x == x
   y == y
   r == r
   ph == arctan2(r*sin(ph), r*cos(ph))
sage: rot = U.continuous_map(U, ((x - sqrt(3)*y)/2, (sqrt(3)*x + y)/2),
....:                          name='R')
sage: rot.display(c_cart, c_cart)
R: U --> U
   (x, y) |--> (-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)
```

Let us use the method `set_expr()` to set the spherical-coordinate expression by hand:

```
sage: rot.set_expr(c_spher, c_spher, (r, ph+pi/3))
sage: rot.display(c_spher, c_spher)
R: U --> U
   (r, ph) |--> (r, 1/3*pi + ph)
```

The expression in Cartesian coordinates has been erased:

```
sage: rot._coord_expression
{(Chart (U, (r, ph)),
  Chart (U, (r, ph))): Coordinate functions (r, 1/3*pi + ph)
  on the Chart (U, (r, ph))}
```

It is recovered (thanks to the known change of coordinates) by a call to `display()`:

```
sage: rot.display(c_cart, c_cart)
R: U --> U
   (x, y) |--> (-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)

sage: rot._coord_expression  # random (dictionary output)
{(Chart (U, (x, y)),
  Chart (U, (x, y))): Coordinate functions (-1/2*sqrt(3)*y + 1/2*x,
  1/2*sqrt(3)*x + 1/2*y) on the Chart (U, (x, y)),
 (Chart (U, (r, ph)),
  Chart (U, (r, ph))): Coordinate functions (r, 1/3*pi + ph)
  on the Chart (U, (r, ph))}
```

TESTS:

We check that this does not change the equality nor the hash value:

```
sage: M = Manifold(2, 'R^2', latex_name=r'\RR^2', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: U = M.open_subset('U', coord_def={c_xy: (y!=0, x<0)})
sage: c_cart = c_xy.restrict(U)
sage: c_spher.<r,ph> = U.chart(r'r:(0,+oo) ph:(0,2*pi):\phi')
sage: ch_cart_spher = c_cart.transition_map(c_spher,
....:                                        [sqrt(x*x+y*y), atan2(y,x)])
sage: ch_cart_spher.set_inverse(r*cos(ph), r*sin(ph))
sage: rot = U.continuous_map(U, ((x - sqrt(3)*y)/2, (sqrt(3)*x + y)/2),
....:                          name='R')
sage: rot2 = copy(rot)
sage: rot == rot2 and hash(rot) == hash(rot2)
True
sage: rot.set_expr(c_spher, c_spher, (r, ph+pi/3))
sage: rot == rot2 and hash(rot) == hash(rot2)
True
```

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 initally defined in Cartesian coordinates:

```
sage: M = Manifold(2, 'R^2', latex_name=r'\RR^2', structure='topological')  #␣
↪the Euclidean plane R^2
sage: c_xy.<x,y> = M.chart() # Cartesian coordinate on R^2
sage: U = M.open_subset('U', coord_def={c_xy: (y!=0, x<0)}) # the complement␣
↪of the segment y=0 and x>0
sage: c_cart = c_xy.restrict(U) # Cartesian coordinates on U
sage: c_spher.<r,ph> = U.chart(r'r:(0,+oo) ph:(0,2*pi):\phi') # spherical␣
↪coordinates on U
sage: # Links between spherical coordinates and Cartesian ones:
sage: ch_cart_spher = c_cart.transition_map(c_spher,
....:                                       [sqrt(x*x+y*y), atan2(y,x)])
sage: ch_cart_spher.set_inverse(r*cos(ph), r*sin(ph), verbose=True)
Check of the inverse coordinate transformation:
   x == x
   y == y
   r == r
   ph == arctan2(r*sin(ph), r*cos(ph))
sage: rot = U.continuous_map(U, ((x - sqrt(3)*y)/2, (sqrt(3)*x + y)/2),
....:                        name='R')
sage: rot.display(c_cart, c_cart)
R: U --> U
   (x, y) |--> (-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)
```

Let us use the method `set_expr()` to set the spherical-coordinate expression by hand:

```
sage: rot.set_expr(c_spher, c_spher, (r, ph+pi/3))
sage: rot.display(c_spher, c_spher)
R: U --> U
   (r, ph) |--> (r, 1/3*pi + ph)
```

The expression in Cartesian coordinates has been erased:

```
sage: rot._coord_expression
{(Chart (U, (r, ph)),
  Chart (U, (r, ph))): Coordinate functions (r, 1/3*pi + ph)
  on the Chart (U, (r, ph))}
```

It is recovered (thanks to the known change of coordinates) by a call to `display()`:

```
sage: rot.display(c_cart, c_cart)
R: U --> U
   (x, y) |--> (-1/2*sqrt(3)*y + 1/2*x, 1/2*sqrt(3)*x + 1/2*y)

sage: rot._coord_expression  # random (dictionary output)
{(Chart (U, (x, y)),
  Chart (U, (x, y))): Coordinate functions (-1/2*sqrt(3)*y + 1/2*x,
  1/2*sqrt(3)*x + 1/2*y) on the Chart (U, (x, y)),
 (Chart (U, (r, ph)),
  Chart (U, (r, ph))): Coordinate functions (r, 1/3*pi + ph)
  on the Chart (U, (r, ph))}
```

TESTS:

We check that this does not change the equality nor the hash value:

```
sage: M = Manifold(2, 'R^2', latex_name=r'\RR^2', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: U = M.open_subset('U', coord_def={c_xy: (y!=0, x<0)})
sage: c_cart = c_xy.restrict(U)
sage: c_spher.<r,ph> = U.chart(r'r:(0,+oo) ph:(0,2*pi):\phi')
sage: ch_cart_spher = c_cart.transition_map(c_spher,
....:                                        [sqrt(x*x+y*y), atan2(y,x)])
sage: ch_cart_spher.set_inverse(r*cos(ph), r*sin(ph))
sage: rot = U.continuous_map(U, ((x - sqrt(3)*y)/2, (sqrt(3)*x + y)/2),
....:                        name='R')
sage: rot2 = copy(rot)
sage: rot == rot2 and hash(rot) == hash(rot2)
True
sage: rot.set_expr(c_spher, c_spher, (r, ph+pi/3))
sage: rot == rot2 and hash(rot) == hash(rot2)
True
```

CHAPTER

TWO

DIFFERENTIABLE MANIFOLDS

2.1 Differentiable Manifolds

Given a non-discrete topological field K (in most applications, $K = \mathbf{R}$ or $K = \mathbf{C}$; see however [4] for $K = \mathbf{Q}_p$ and [5] 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 $K = \mathbf{R}$ is assumed;
- if $K = \mathbf{C}$, any C^k-manifold with $k \geq 1$ is actually a C^∞-manifold (even an analytic manifold);
- if $K = \mathbf{R}$, any C^k-manifold with $k \geq 1$ admits a compatible C^∞-structure (Whitney's smoothing theorem).

Differentiable manifolds are implemented via the class `DifferentiableManifold`. Open subsets of differentiable manifolds are also implemented via `DifferentiableManifold`, since they are differentiable manifolds by themselves.

The user interface is provided by the generic function `Manifold()`, with the argument `structure` set to `'differentiable'` and the argument `diff_degree` set to k, or the argument `structure` set to `'smooth'` (the default value).

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

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

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

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

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

By default, the created object is a smooth manifold:

[4] J.-P. Serre : *Lie Algebras and Lie Groups*, 2nd ed., Springer (Berlin) (1992); doi:10.1007/978-3-540-70634-2

[5] W. Bertram : *Differential Geometry, Lie Groups and Symmetric Spaces over General Base Fields and Rings*, Memoirs of the American Mathematical Society, vol. 192 (2008); doi:10.1090/memo/0900; Arxiv math/0502168

```
sage: M.diff_degree()
 +Infinity
```

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

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

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

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

Thanks to the operator `<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)
<type 'sage.symbolic.expression.Expression'>
```

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

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

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

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

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

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

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

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

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

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

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

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

The inverse of the transition map is computed by the method `inverse()` :

```
sage: stereoN_to_S.inverse()
Change of coordinates from Chart (W, (u, v)) to Chart (W, (x, y))
sage: stereoN_to_S.inverse().display()
x = u/(u^2 + v^2)
y = v/(u^2 + v^2)
```

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

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

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

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

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

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

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

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

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

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

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

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

Given the definition of p, we have of course:

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

Similarly:

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

A differentiable scalar field on the sphere:

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

Example 2: the Riemann sphere as a differentiable manifold of dimension 1 over $\mathbf{C}$

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

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

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

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

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

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

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

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

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

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

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

```
sage: W.<w> = V.chart(); W
Chart (V, (w,))
```

```
sage: inf = M.point((0,), chart=W, name='inf', latex_name=r'\infty')
sage: inf
Point inf on the 1-dimensional complex manifold C*
```

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

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

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

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

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

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

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

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

and we have:

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

The following subsets and charts have been defined:

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

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

```
sage: f = M.constant_scalar_field(3+2*I, name='f'); f
Scalar field f on the 1-dimensional complex manifold C*
sage: f.display()
f: C* --> C
on U: z |--> 2*I + 3
on V: w |--> 2*I + 3
```

```
sage: f(O)
2*I + 3
sage: f(i)
2*I + 3
sage: f(inf)
2*I + 3
sage: f.parent()
Algebra of differentiable scalar fields on the 1-dimensional complex
 manifold C*
sage: f.parent().category()
Category of commutative algebras over Symbolic Ring
```

AUTHORS:

- Eric Gourgoulhon (2015): initial version

REFERENCES:

class `sage.manifolds.differentiable.manifold.`**`DifferentiableManifold`**(*n*, *name*, *field*, *structure*, *ambient=None*, *diff_degree=+Infinity*, *latex_name=None*, *start_index=0*, *category=None*, *unique_tag=None*)

Bases: *`sage.manifolds.manifold.TopologicalManifold`*

Differentiable manifold over a topological field K.

Given a non-discrete topological field K (in most applications, $K = \mathbf{R}$ or $K = \mathbf{C}$; see however [4] for $K = \mathbf{Q}_p$ and [5] 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 $K = \mathbf{R}$ is assumed;
- if $K = \mathbf{C}$, any C^k-manifold with $k \geq 1$ is actually a C^∞-manifold (even an analytic manifold);
- if $K = \mathbf{R}$, any C^k-manifold with $k \geq 1$ admits a compatible C^∞-structure (Whitney's smoothing theorem).

INPUT:

- `n` – positive integer; dimension of the manifold
- `name` – string; name (symbol) given to the manifold
- `field` – field K on which the manifold is defined; allowed values are
  - `'real'` or an object of type `RealField` (e.g., `RR`) for a manifold over $\mathbf{R}$
  - `'complex'` or an object of type `ComplexField` (e.g., `CC`) for a manifold over $\mathbf{C}$
  - an object in the category of topological fields (see `Fields` and `TopologicalSpaces`) for other types of manifolds

- `structure` – manifold structure (see *DifferentialStructure* or *RealDifferentialStructure*)
- `ambient` – (default: `None`) if not `None` , must be a differentiable manifold; the created object is then an open subset of `ambient`
- `diff_degree` – (default: `infinity`) degree k of differentiability
- `latex_name` – (default: `None`) string; LaTeX symbol to denote the manifold; if none is provided, it is set to `name`
- `start_index` – (default: 0) integer; lower value of the range of indices used for "indexed objects" on the manifold, e.g. coordinates in a chart
- `category` – (default: `None`) to specify the category; if `None` , `Manifolds(field).Differentiable()` (or `Manifolds(field).Smooth()` if `diff_degree` = `infinity`) is assumed (see the category `Manifolds`)
- `unique_tag` – (default: `None`) tag used to force the construction of a new object when all the other arguments have been used previously (without `unique_tag` , the `UniqueRepresentation` behavior inherited from *ManifoldSubset* , via *TopologicalManifold* , would return the previously constructed object corresponding to these arguments).

EXAMPLES:

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

```
sage: M = Manifold(4, 'M', latex_name=r'\mathcal{M}'); M
4-dimensional differentiable manifold M
sage: type(M)
<class 'sage.manifolds.differentiable.manifold.DifferentiableManifold_with_
↪category'>
sage: latex(M)
\mathcal{M}
sage: dim(M)
4
```

Since the base field has not been specified, $\mathbf{R}$ has been assumed:

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

Since the degree of differentiability has not been specified, the default value, C^∞, has been assumed:

```
sage: M.diff_degree()
+Infinity
```

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

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

A complex manifold:

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

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

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

A differentiable manifold is of course a topological manifold:

```
sage: isinstance(M, sage.manifolds.manifold.TopologicalManifold)
True
sage: isinstance(N, sage.manifolds.manifold.TopologicalManifold)
True
```

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

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

The corresponding Sage *elements* are points:

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

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

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

Since an open subset of a differentiable manifold M is itself a differentiable manifold, open subsets of M have all attributes of manifolds:

```
sage: U = M.open_subset('U', coord_def={X: t>0}); U
Open subset U of the 4-dimensional differentiable manifold M
sage: U.category()
Join of Category of subobjects of sets and Category of smooth manifolds
 over Real Field with 53 bits of precision
```

```
sage: U.base_field() == M.base_field()
True
sage: dim(U) == dim(M)
True
```

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

```
sage: TestSuite(M).run()
```

diff_degree ( )

Return the manifold's degree of differentiability.

The degree of differentiability is the integer k (possibly $k = \infty$) such that the manifold is a C^k-manifold over its base field.

EXAMPLES:

```
sage: M = Manifold(2, 'M')
sage: M.diff_degree()
+Infinity
sage: M = Manifold(2, 'M', structure='differentiable', diff_degree=3)
sage: M.diff_degree()
3
```

diff_map (*codomain*, *coord_functions=None*, *chart1=None*, *chart2=None*, *name=None*, *latex_name=None*)

Define a differentiable map between the current differentiable manifold and a differentiable manifold over the same topological field.

See `DiffMap` for a complete documentation.

INPUT:

- `codomain` – the map codomain (a differentiable manifold over the same topological field as the current differentiable manifold)
- `coord_functions` – (default: `None`) if not `None` , must be either
  - (i) a dictionary of the coordinate expressions (as lists (or tuples) of the coordinates of the image expressed in terms of the coordinates of the considered point) with the pairs of charts (chart1, chart2) as keys (chart1 being a chart on the current manifold and chart2 a chart on `codomain`)
  - (ii) a single coordinate expression in a given pair of charts, the latter being provided by the arguments `chart1` and `chart2`

  In both cases, if the dimension of the arrival manifold is 1, a single coordinate expression can be passed instead of a tuple with a single element
- `chart1` – (default: `None` ; used only in case (ii) above) chart on the current manifold defining the start coordinates involved in `coord_functions` for case (ii); if none is provided, the coordinates are assumed to refer to the manifold's default chart
- `chart2` – (default: `None` ; used only in case (ii) above) chart on `codomain` defining the arrival coordinates involved in `coord_functions` for case (ii); if none is provided, the coordinates are assumed to refer to the default chart of `codomain`
- `name` – (default: `None`) name given to the differentiable map
- `latex_name` – (default: `None`) LaTeX symbol to denote the differentiable map; if none is provided, the LaTeX symbol is set to `name`

OUTPUT:

•the differentiable map, as an instance of *DiffMap*

EXAMPLES:

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

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

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

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

The differentiable map acting on a point:

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

See the documentation of class *DiffMap* for more examples.

diff_mapping (*codomain*, *coord_functions=None*, *chart1=None*, *chart2=None*, *name=None*, *latex_name=None*)

Deprecated.

Use *diff_map()* instead.

EXAMPLE:

```
sage: M = Manifold(2, 'M'); X.<x,y> = M.chart()
sage: N = Manifold(2, 'N'); Y.<u,v> = N.chart()
sage: Phi = M.diff_mapping(N, {(X,Y): [x+y, x-y]}, name='Phi')
doctest:...: DeprecationWarning: Use diff_map() instead.
See http://trac.sagemath.org/18783 for details.
sage: Phi
Differentiable map Phi from the 2-dimensional differentiable
 manifold M to the 2-dimensional differentiable manifold N
```

diffeomorphism (*codomain*, *coord_functions=None*, *chart1=None*, *chart2=None*, *name=None*, *latex_name=None*)

Define a diffeomorphism between the current manifold and another one.

See *DiffMap* for a complete documentation.

INPUT:

- `codomain` – codomain of the diffeomorphism (the arrival manifold or some subset of it)
- `coord_functions` – (default: `None`) if not `None` , must be either
  - (i) a dictionary of the coordinate expressions (as lists (or tuples) of the coordinates of the image expressed in terms of the coordinates of the considered point) with the pairs of charts (chart1, chart2) as keys (chart1 being a chart on the current manifold and chart2 a chart on `codomain`)
  - (ii) a single coordinate expression in a given pair of charts, the latter being provided by the arguments `chart1` and `chart2`

  In both cases, if the dimension of the arrival manifold is 1, a single coordinate expression can be passed instead of a tuple with a single element
- `chart1` – (default: `None` ; used only in case (ii) above) chart on the current manifold defining the start coordinates involved in `coord_functions` for case (ii); if none is provided, the coordinates are assumed to refer to the manifold's default chart
- `chart2` – (default: `None` ; used only in case (ii) above) chart on `codomain` defining the arrival coordinates involved in `coord_functions` for case (ii); if none is provided, the coordinates are assumed to refer to the default chart of `codomain`
- `name` – (default: `None`) name given to the diffeomorphism
- `latex_name` – (default: `None`) LaTeX symbol to denote the diffeomorphism; if none is provided, the LaTeX symbol is set to `name`

OUTPUT:

- the diffeomorphism, as an instance of *`DiffMap`*

EXAMPLE:

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

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

The inverse diffeomorphism:

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

See the documentation of class *`DiffMap`* for more examples.

open_subset (*name*, *latex_name=None*, *coord_def={}*)

Create an open subset of the manifold.

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

INPUT:

- `name` – name given to the open subset
- `latex_name` – (default: `None`) LaTeX symbol to denote the subset; if none is provided, it is set to `name`
- `coord_def` – (default: {}) definition of the subset in terms of coordinates; `coord_def` must a be dictionary with keys charts in the manifold's atlas and values the symbolic expressions formed by the coordinates to define the subset.

OUTPUT:

- the open subset, as an instance of `DifferentiableManifold`

EXAMPLES:

Creating an open subset of a differentiable manifold:

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

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

```
sage: A.category()
Join of Category of subobjects of sets and Category of smooth
 manifolds over Real Field with 53 bits of precision
sage: A.base_field() == M.base_field()
True
sage: dim(A) == dim(M)
True
```

Creating an open subset of `A`:

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

We have then:

```
sage: A.list_of_subsets()
[Open subset A of the 2-dimensional differentiable manifold M,
 Open subset B of the 2-dimensional differentiable manifold M]
sage: B.is_subset(A)
True
sage: B.is_subset(M)
True
```

Defining an open subset by some coordinate restrictions: the open unit disk in of the Euclidean plane:

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

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

```
sage: U.atlas()
[Chart (U, (x, y))]
sage: U.default_chart()
Chart (U, (x, y))
sage: U.default_chart() is X.restrict(U)
True
```

An point in `U` :

```
sage: p = U.an_element(); p
Point on the 2-dimensional differentiable manifold M
sage: X(p)  # the coordinates (x,y) of p
(0, 0)
sage: p in U
True
```

Checking whether various points, defined by their coordinates w.r.t. chart `X` , are in `U` :

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

2.2 Coordinate Charts on Differentiable Manifolds

The class *`DiffChart`* implements coordinate charts on a differentiable manifold over a topological field K (in most applications, $K = \mathbf{R}$ or $K = \mathbf{C}$).

The subclass *`RealDiffChart`* is devoted to the case $K = \mathbf{R}$, for which the concept of coordinate range is meaningful. Moreover, *`RealDiffChart`* is endowed with some plotting capabilities (cf. method *`plot()`*).

Transition maps between charts are implemented via the class *`DiffCoordChange`* .

AUTHORS:

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

REFERENCES:

- Chap. 1 of *[Lee13]* J.M. Lee : *Introduction to Smooth Manifolds*, 2nd ed., Springer (New York) (2013)

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

Bases: *`sage.manifolds.chart.Chart`*

Chart on a differentiable manifold.

Given a differentiable manifold M of dimension n over a topological field K, a *chart* is a member (U, φ) of the manifold's differentiable atlas; U is then an open subset of M and $\varphi : U \to V \subset K^n$ is a homeomorphism from U to an open subset V of K^n.

The components $(x^1,\ldots,x^n)$ of φ, 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,φ).

INPUT:

- `domain` – open subset U on which the chart is defined
- `coordinates` – (default: '' (empty string)) single string defining the coordinate symbols, with ' ' (whitespace) as a separator; each item has at most two fields, separated by ':':
  1. The coordinate symbol (a letter or a few letters)
  2. (optional) The LaTeX spelling of the coordinate; if not provided the coordinate symbol given in the first field will be used.

  If it contains any LaTeX expression, the string `coordinates` must be declared with the prefix 'r' (for "raw") to allow for a proper treatment of LaTeX's backslash character (see examples below). If no LaTeX spelling is to be set for any coordinate, the argument `coordinates` can be omitted when the shortcut operator `<,>` is used via Sage preparser (see examples below)
- `names` – (default: `None`) unused argument, except if `coordinates` is not provided; it must then be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator `<,>` is used).

EXAMPLES:

A chart on a complex 2-dimensional differentiable manifold:

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

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

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

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

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

The coordinates are then immediately accessible:

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

The trick is performed by Sage 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.

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

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

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

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

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

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

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

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

Each constructed chart is automatically added to the manifold's user atlas:

```
sage: M.atlas()
[Chart (M, (x, y)), Chart (U, (z1, z2))]
```

and to the atlas of the chart's domain:

```
sage: U.atlas()
[Chart (U, (z1, z2))]
```

Manifold subsets have a *default chart*, which, unless changed via the method *`set_default_chart()`* , is the first defined chart on the subset (or on 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 φ 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
```

See also:

`RealDiffChart` for charts on differentiable manifolds over $\mathbf{R}$.

`transition_map` (*other*, *transformations*, *intersection_name=None*, *restrictions1=None*, *restrictions2=None*)

Construct the transition map between the current chart, (U, φ) say, and another one, (V, ψ) 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, ψ) in terms of the coordinates $(x^1, \ldots, x^n)$ of (U, φ) on the open subset where the two charts intersect, i.e. on $U \cap V$.

By definition, the transition map $\psi \circ \varphi^{-1}$ must be of classe C^k, where k is the degree of differentiability of the manifold (cf. *`diff_degree()`*).

INPUT:

- `other` – the chart (V, ψ)
- `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 != 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, ψ) that define $U \cap V$ if the latter differs from V (see `restrictions1` for the syntax)

OUTPUT:

- The transition map $\psi \circ \varphi^{-1}$ defined on $U \cap V$, as an instance of *`DiffCoordChange`* .

EXAMPLES:

Transition map between two stereographic charts on the circle S^1:

```
sage: M = Manifold(1, 'S^1')
sage: U = M.open_subset('U') # Complement of the North pole
sage: cU.<x> = U.chart() # Stereographic chart from the North pole
sage: V = M.open_subset('V') # Complement of the South pole
sage: cV.<y> = V.chart() # Stereographic chart from the South pole
sage: M.declare_union(U,V)   # S^1 is the union of U and V
sage: trans = cU.transition_map(cV, 1/x, intersection_name='W',
....:                           restrictions1= x!=0, restrictions2 = y!=0)
sage: trans
Change of coordinates from Chart (W, (x,)) to Chart (W, (y,))
sage: trans.display()
y = 1/x
```

The subset W, intersection of U and V, has been created by `transition_map()` :

```
sage: M.list_of_subsets()
[1-dimensional differentiable manifold S^1,
 Open subset U of the 1-dimensional differentiable manifold S^1,
 Open subset V of the 1-dimensional differentiable manifold S^1,
 Open subset W of the 1-dimensional differentiable manifold S^1]
sage: W = M.list_of_subsets()[3]
sage: W is U.intersection(V)
True
sage: M.atlas()
[Chart (U, (x,)), Chart (V, (y,)), Chart (W, (x,)), Chart (W, (y,))]
```

Transition map between the polar chart and the Cartesian one on $\mathbf{R}^2$:

```
sage: M = Manifold(2, 'R^2')
sage: c_cart.<x,y> = M.chart()
sage: U = M.open_subset('U') # the complement of the half line {y=0, x >= 0}
sage: c_spher.<r,phi> = U.chart(r'r:(0,+oo) phi:(0,2*pi):\phi')
sage: trans = c_spher.transition_map(c_cart, (r*cos(phi), r*sin(phi)),
....:                                    restrictions2=(y!=0, x<0))
sage: trans
Change of coordinates from Chart (U, (r, phi)) to Chart (U, (x, y))
sage: trans.display()
x = r*cos(phi)
y = r*sin(phi)
```

In this case, no new subset has been created since $U \cap M = U$:

```
sage: M.list_of_subsets()
[2-dimensional differentiable manifold R^2,
 Open subset U of the 2-dimensional differentiable manifold R^2]
```

but a new chart has been created: $(U, (x, y))$:

```
sage: M.atlas()
[Chart (R^2, (x, y)), Chart (U, (r, phi)), Chart (U, (x, y))]
```

class `sage.manifolds.differentiable.chart.`**`DiffCoordChange`** (*chart1*, *chart2*, **transformations*)

Bases: *`sage.manifolds.chart.CoordChange`*

Transition map between two charts of a differentiable manifold.

Giving two coordinate charts (U, φ) and (V, ψ) on a differentiable manifold M of dimension n over a topological field K, the *transition map from* (U, φ) *to* (V, ψ) 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, ψ) in terms of the coordinates $(x^1, \ldots, x^n)$ of (U, φ) on the open subset where the two charts intersect, i.e. on $U \cap V$.

By definition, the transition map $\psi \circ \varphi^{-1}$ must be of classe C^k, where k is the degree of differentiability of the manifold (cf. *`diff_degree()`*).

INPUT:

- `chart1` – chart (U, φ)
- `chart2` – chart (V, ψ)
- `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
```

class `sage.manifolds.differentiable.chart.`**`RealDiffChart`** (*domain*, *coordinates=''*, *names=None*)

Bases: `sage.manifolds.differentiable.chart.DiffChart`, `sage.manifolds.chart.RealChart`

Chart on a differentiable manifold over $\mathbf{R}$.

Given a differentiable manifold M of dimension n over $\mathbf{R}$, a *chart* is a member (U, φ) of the manifold's differentiable atlas; U is then an open subset of M and $\varphi : U \to V \subset \mathbf{R}^n$ is a homeomorphism from U to an open subset V of $\mathbf{R}^n$.

The components $(x^1, \ldots, x^n)$ of φ, defined by $\varphi(p) = (x^1(p), \ldots, x^n(p)) \in \mathbf{R}^n$ for any point $p \in U$, are called the *coordinates* of the chart (U, φ).

INPUT:

- `domain` – open subset U on which the chart is defined
- `coordinates` – (default: '' (empty string)) single string defining the coordinate symbols and ranges, with ' ' (whitespace) as a separator; each item has at most three fields, separated by ':':

  1. The coordinate symbol (a letter or a few letters)
  2. (optional) The interval I defining the coordinate range: if not provided, the coordinate is assumed to span all $\mathbf{R}$; otherwise I must be provided in the form `(a,b)` (or equivalently `]a,b[`). The bounds `a` and `b` can be `+/-Infinity`, `Inf`, `infinity`, `inf` or `oo`. For *singular* coordinates, non-open intervals such as `[a,b]` and `(a,b]` (or equivalently `]a,b]`) are allowed. Note that the interval declaration must not contain any whitespace.
  3. (optional) The LaTeX spelling of the coordinate; if not provided the coordinate symbol given in the first field will be used.

  The order of the fields 2 and 3 does not matter and each of them can be omitted. If it contains any LaTeX expression, the string `coordinates` must be declared with the prefix 'r' (for "raw") to allow for a proper treatment of LaTeX backslash characters (see examples below). If no interval range and no LaTeX spelling is to be set for any coordinate, the argument `coordinates` can be omitted when the shortcut operator `<,>` is used via Sage preparser (see examples below)
- `names` – (default: `None`) unused argument, except if `coordinates` is not provided; it must then be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator `<,>` is used).

EXAMPLES:

Cartesian coordinates on $\mathbf{R}^3$:

```
sage: M = Manifold(3, 'R^3', r'\RR^3', start_index=1)
sage: c_cart = M.chart('x y z'); c_cart
Chart (R^3, (x, y, z))
sage: type(c_cart)
<class 'sage.manifolds.differentiable.chart.RealDiffChart'>
```

To have the coordinates accessible as global variables, one has to set:

```
sage: (x,y,z) = c_cart[:]
```

However, a shortcut is to use the declarator `<x,y,z>` in the left-hand side of the chart declaration (there is then no need to pass the string `'x y z'` to `chart()`):

```
sage: M = Manifold(3, 'R^3', r'\RR^3', start_index=1)
sage: c_cart.<x,y,z> = M.chart(); c_cart
Chart (R^3, (x, y, z))
```

The coordinates are then immediately accessible:

```
sage: y
y
sage: y is c_cart[2]
True
```

The trick is performed by Sage preparser:

```
sage: 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:

```
sage: M = Manifold(3, 'R^3', r'\RR^3', start_index=1)
sage: c_cart.<x1,y1,z1> = M.chart('x y z'); c_cart
Chart (R^3, (x, y, z))
```

Then `y` is not known as a global variable and the coordinate y is accessible only through the global variable `y1` :

```
sage: y1
y
sage: y1 is c_cart[2]
True
```

However, having the name of the Python variable coincide with the coordinate symbol is quite convenient; so it is recommended to declare:

```
sage: forget()   # for doctests only
sage: M = Manifold(3, 'R^3', r'\RR^3', start_index=1)
sage: c_cart.<x,y,z> = M.chart()
```

Spherical coordinates on the subset U of $\mathbf{R}^3$ that is the complement of the half-plane $\{y = 0, x \geq 0\}$:

```
sage: U = M.open_subset('U')
sage: c_spher.<r,th,ph> = U.chart(r'r:(0,+oo) th:(0,pi):\theta ph:(0,2*pi):\phi')
sage: c_spher
Chart (U, (r, th, ph))
```

Note the prefix 'r' for the string defining the coordinates in the arguments of `chart` .

Coordinates are Sage symbolic variables (see `sage.symbolic.expression`):

```
sage: type(th)
<type 'sage.symbolic.expression.Expression'>
sage: latex(th)
{\theta}
sage: assumptions(th)
[th is real, th > 0, th < pi]
```

Coordinate are also accessible by their indices:

```
sage: x1 = c_spher[1]; x2 = c_spher[2]; x3 = c_spher[3]
sage: [x1, x2, x3]
[r, th, ph]
sage: (x1, x2, x3) == (r, th, ph)
True
```

The full set of coordinates is obtained by means of the operator [:]:

```
sage: c_cart[:]
(x, y, z)
sage: c_spher[:]
(r, th, ph)
```

Let us check that the declared coordinate ranges have been taken into account:

```
sage: c_cart.coord_range()
x: (-oo, +oo); y: (-oo, +oo); z: (-oo, +oo)
sage: c_spher.coord_range()
r: (0, +oo); th: (0, pi); ph: (0, 2*pi)
sage: bool(th>0 and th<pi)
True
sage: assumptions()  # list all current symbolic assumptions
[x is real, y is real, z is real, r is real, r > 0, th is real,
 th > 0, th < pi, ph is real, ph > 0, ph < 2*pi]
```

The coordinate ranges are used for simplifications:

```
sage: simplify(abs(r)) # r has been declared to lie in the interval (0,+oo)
r
sage: simplify(abs(x)) # no positive range has been declared for x
abs(x)
```

Each constructed chart is automatically added to the manifold's user atlas:

```
sage: M.atlas()
[Chart (R^3, (x, y, z)), Chart (U, (r, th, ph))]
```

and to the atlas of its domain:

```
sage: U.atlas()
[Chart (U, (r, th, ph))]
```

Manifold subsets have a *default chart*, which, unless changed via the method `set_default_chart()`, is the first defined chart on the subset (or on a open subset of it):

```
sage: M.default_chart()
Chart (R^3, (x, y, z))
sage: U.default_chart()
Chart (U, (r, th, ph))
```

The default charts are not privileged charts on the manifold, but rather charts whose name can be skipped in the argument list of functions having an optional `chart=` argument.

The action of the chart map φ on a point is obtained by means of the call operator, i.e. the operator `()`:

```
sage: p = M.point((1,0,-2)); p
Point on the 3-dimensional differentiable manifold R^3
sage: c_cart(p)
(1, 0, -2)
sage: c_cart(p) == p.coord(c_cart)
True
sage: q = M.point((2,pi/2,pi/3), chart=c_spher) # point defined by its spherical␣
↪coordinates
```

```
sage: c_spher(q)
(2, 1/2*pi, 1/3*pi)
sage: c_spher(q) == q.coord(c_spher)
True
sage: a = U.point((1,pi/2,pi)) # the default coordinates on U are the spherical␣
↪ones
sage: c_spher(a)
(1, 1/2*pi, pi)
sage: c_spher(a) == a.coord(c_spher)
True
```

Cartesian coordinates on U as an example of chart construction with coordinate restrictions: since U is the complement of the half-plane $\{y = 0, x \geq 0\}$, we must have $y \neq 0$ or $x < 0$ on U. Accordingly, we set:

```
sage: c_cartU.<x,y,z> = U.chart()
sage: c_cartU.add_restrictions((y!=0, x<0)) # the tuple (y!=0, x<0) means y!=0 or␣
↪x<0
sage: # c_cartU.add_restrictions([y!=0, x<0]) would have meant y!=0 AND x<0
sage: U.atlas()
[Chart (U, (r, th, ph)), Chart (U, (x, y, z))]
sage: M.atlas()
[Chart (R^3, (x, y, z)), Chart (U, (r, th, ph)), Chart (U, (x, y, z))]
sage: c_cartU.valid_coordinates(-1,0,2)
True
sage: c_cartU.valid_coordinates(1,0,2)
False
sage: c_cart.valid_coordinates(1,0,2)
True
```

Chart grids can be drawn in 2D or 3D graphics thanks to the method `plot()` .

2.3 Scalar Fields

2.3.1 Algebra of Differentiable Scalar Fields

The class `DiffScalarFieldAlgebra` implements the commutative algebra $C^k(M)$ of differentiable scalar fields on a differentiable manifold M of class C^k over a topological field K (in most applications, $K = \mathbf{R}$ or $K = \mathbf{C}$). By *differentiable scalar field*, it is meant a function $M \to K$ 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:

- Eric Gourgoulhon, Michal Bejger (2014-2015): initial version

REFERENCES:

class `sage.manifolds.differentiable.scalarfield_algebra.`**`DiffScalarFieldAlgebra`**(*domain*)
Bases: `sage.manifolds.scalarfield_algebra.ScalarFieldAlgebra`

Commutative algebra of differentiable scalar fields on a differentiable manifold.

If M is a differentiable manifold of class C^k over a topological field K, the *commutative algebra of scalar fields on* M is the set $C^k(M)$ of all k-times continuously differentiable maps $M \to K$. 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 $K = \mathbf{R}$ or $K = \mathbf{C}$, the field K over which the algebra $C^k(M)$ is constructed is represented by Sage's Symbolic Ring `SR`, since there is no exact representation of $\mathbf{R}$ nor $\mathbf{C}$ in Sage.

Via its base class *`ScalarFieldAlgebra`*, the class *`DiffScalarFieldAlgebra`* inherits from `Parent`, with the category set to `CommutativeAlgebras`. The corresponding *element* class is *`DiffScalarField`*.

INPUT:

- `domain` – the differentiable manifold M on which the scalar fields are defined (must be an instance of class *`DifferentiableManifold`*)

EXAMPLES:

Algebras of scalar fields on the sphere S^2 and on some open subset of it:

```
sage: M = Manifold(2, 'M') # the 2-dimensional sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V)   # S^2 is the union of U and V
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                 intersection_name='W', restrictions1= x^2+y^2!=0,
....:                 restrictions2= u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: CM = M.scalar_field_algebra() ; CM
Algebra of differentiable scalar fields on the 2-dimensional
 differentiable manifold M
sage: W = U.intersection(V)  # S^2 minus the two poles
sage: CW = W.scalar_field_algebra() ; CW
Algebra of differentiable scalar fields on the Open subset W of the
 2-dimensional differentiable manifold M
```

$C^k(M)$ and $C^k(W)$ belong to the category of commutative algebras over $\mathbf{R}$ (represented here by Sage's Symbolic Ring):

```
sage: CM.category()
Category of commutative algebras over Symbolic Ring
sage: CM.base_ring()
Symbolic Ring
sage: CW.category()
Category of commutative algebras over Symbolic Ring
sage: CW.base_ring()
Symbolic Ring
```

The elements of $C^k(M)$ are scalar fields on M:

```
sage: CM.an_element()
Scalar field on the 2-dimensional differentiable manifold M
sage: CM.an_element().display()  # this sample element is a constant field
M --> R
on U: (x, y) |--> 2
on V: (u, v) |--> 2
```

Those of $C^k(W)$ are scalar fields on W:

```
sage: CW.an_element()
Scalar field on the Open subset W of the 2-dimensional differentiable
 manifold M
```

```
sage: CW.an_element().display()  # this sample element is a constant field
W --> R
(x, y) |--> 2
(u, v) |--> 2
```

The zero element:

```
sage: CM.zero()
Scalar field zero on the 2-dimensional differentiable manifold M
sage: CM.zero().display()
zero: M --> R
on U: (x, y) |--> 0
on V: (u, v) |--> 0
```

```
sage: CW.zero()
Scalar field zero on the Open subset W of the 2-dimensional
 differentiable manifold M
sage: CW.zero().display()
zero: W --> R
   (x, y) |--> 0
   (u, v) |--> 0
```

The unit element:

```
sage: CM.one()
Scalar field 1 on the 2-dimensional differentiable manifold M
sage: CM.one().display()
1: M --> R
on U: (x, y) |--> 1
on V: (u, v) |--> 1
```

```
sage: CW.one()
Scalar field 1 on the Open subset W of the 2-dimensional differentiable
 manifold M
sage: CW.one().display()
1: W --> R
(x, y) |--> 1
(u, v) |--> 1
```

A generic element can be constructed as for any parent in Sage, namely by means of the `__call__` operator on the parent (here with the dictionary of the coordinate expressions defining the scalar field):

```
sage: f = CM({c_xy: atan(x^2+y^2), c_uv: pi/2 - atan(u^2+v^2)}); f
Scalar field on the 2-dimensional differentiable manifold M
sage: f.display()
M --> R
on U: (x, y) |--> arctan(x^2 + y^2)
on V: (u, v) |--> 1/2*pi - arctan(u^2 + v^2)
sage: f.parent()
Algebra of differentiable scalar fields on the 2-dimensional
 differentiable manifold M
```

Specific elements can also be constructed in this way:

```
sage: CM(0) == CM.zero()
True
```

```
sage: CM(1) == CM.one()
True
```

Note that the zero scalar field is cached:

```
sage: CM(0) is CM.zero()
True
```

Elements can also be constructed by means of the method `scalar_field()` acting on the domain (this allows one to set the name of the scalar field at the construction):

```
sage: f1 = M.scalar_field({c_xy: atan(x^2+y^2), c_uv: pi/2 - atan(u^2+v^2)},
....:                     name='f')
sage: f1.parent()
Algebra of differentiable scalar fields on the 2-dimensional
 differentiable manifold M
sage: f1 == f
True
sage: M.scalar_field(0, chart='all') == CM.zero()
True
```

The algebra $C^k(M)$ coerces to $C^k(W)$ since W is an open subset of M:

```
sage: CW.has_coerce_map_from(CM)
True
```

The reverse is of course false:

```
sage: CM.has_coerce_map_from(CW)
False
```

The coercion map is nothing but the restriction to W of scalar fields on M:

```
sage: fW = CW(f) ; fW
Scalar field on the Open subset W of the 2-dimensional differentiable
 manifold M
sage: fW.display()
W --> R
(x, y) |--> arctan(x^2 + y^2)
(u, v) |--> 1/2*pi - arctan(u^2 + v^2)
```

```
sage: CW(CM.one()) == CW.one()
True
```

The coercion map allows for the addition of elements of $C^k(W)$ with elements of $C^k(M)$, the result being an element of $C^k(W)$:

```
sage: s = fW + f
sage: s.parent()
Algebra of differentiable scalar fields on the Open subset W of the
 2-dimensional differentiable manifold M
sage: s.display()
W --> R
(x, y) |--> 2*arctan(x^2 + y^2)
(u, v) |--> pi - 2*arctan(u^2 + v^2)
```

Another coercion is that from the Symbolic Ring, the parent of all symbolic expressions (cf. `SymbolicRing`). Since the Symbolic Ring is the base ring for the algebra `CM` , the coercion of a symbolic expression `s` is performed by the operation `s*CM.one()` , which invokes the reflected multiplication operator `sage.manifolds.scalarfield.ScalarField._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 --> R
on U: (x, y) |--> sqrt(2)*pi
on V: (u, v) |--> sqrt(2)*pi
sage: a = var('a')
sage: h = CM(a); h.display()
M --> R
on U: (x, y) |--> a
on V: (u, v) |--> a
```

If the symbolic expression involves some coordinate of one of the manifold's charts, the outcome is initialized only on the chart domain:

```
sage: h = CM(a+x); h.display()
M --> R
on U: (x, y) |--> a + x
sage: h = CM(a+u); h.display()
M --> R
on V: (u, v) |--> a + u
```

If the symbolic expression involves coordinates of different charts, the scalar field is created as a Python object, but is not initialized, in order to avoid any ambiguity:

```
sage: h = CM(x+u); h.display()
M --> R
```

TESTS OF THE ALGEBRA LAWS:

Ring laws:

```
sage: h = CM(pi*sqrt(2))
sage: s = f + h ; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M --> R
on U: (x, y) |--> sqrt(2)*pi + arctan(x^2 + y^2)
on V: (u, v) |--> 1/2*pi*(2*sqrt(2) + 1) - arctan(u^2 + v^2)
```

```
sage: s = f - h ; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M --> R
on U: (x, y) |--> -sqrt(2)*pi + arctan(x^2 + y^2)
on V: (u, v) |--> -1/2*pi*(2*sqrt(2) - 1) - arctan(u^2 + v^2)
```

```
sage: s = f*h ; s
Scalar field on the 2-dimensional differentiable manifold M
```

```
sage: s.display()
M --> R
on U: (x, y) |--> sqrt(2)*pi*arctan(x^2 + y^2)
on V: (u, v) |--> 1/2*sqrt(2)*(pi^2 - 2*pi*arctan(u^2 + v^2))
```

```
sage: s = f/h ; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M --> R
on U: (x, y) |--> 1/2*sqrt(2)*arctan(x^2 + y^2)/pi
on V: (u, v) |--> 1/4*(sqrt(2)*pi - 2*sqrt(2)*arctan(u^2 + v^2))/pi
```

```
sage: f*(h+f) == f*h + f*f
True
```

Ring laws with coercion:

```
sage: f - fW == CW.zero()
True
sage: f/fW == CW.one()
True
sage: s = f*fW ; s
Scalar field on the Open subset W of the 2-dimensional differentiable
 manifold M
sage: s.display()
W --> R
(x, y) |--> arctan(x^2 + y^2)^2
(u, v) |--> 1/4*pi^2 - pi*arctan(u^2 + v^2) + arctan(u^2 + v^2)^2
sage: s/f == fW
True
```

Multiplication by a number:

```
sage: s = 2*f ; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M --> R
on U: (x, y) |--> 2*arctan(x^2 + y^2)
on V: (u, v) |--> pi - 2*arctan(u^2 + v^2)
```

```
sage: 0*f == CM.zero()
True
sage: 1*f == f
True
sage: 2*(f/2) == f
True
sage: (f+2*f)/3 == f
True
sage: 1/3*(f+2*f) == f
True
```

The Sage test suite for algebras is passed:

```
sage: TestSuite(CM).run()
```

It is passed also for $C^k(W)$:

```
sage: TestSuite(CW).run()
```

Element
alias of `DiffScalarField`

2.3.2 Differentiable Scalar Fields

Given a differentiable manifold M of class C^k over a topological field K (in most applications, $K = \mathbf{R}$ or $K = \mathbf{C}$), a *differentiable scalar field* on M is a map

$$f : M \longrightarrow K$$

of class C^k.

Differentiable scalar fields are implemented by the class `DiffScalarField`.

AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2013-2015): initial version

REFERENCES:

class `sage.manifolds.differentiable.scalarfield.`**`DiffScalarField`**(*parent*, *coord_expression=None*, *chart=None*, *name=None*, *latex_name=None*)

Bases: `sage.manifolds.scalarfield.ScalarField`

Differentiable scalar field on a differentiable manifold.

Given a differentiable manifold M of class C^k over a topological field K (in most applications, $K = \mathbf{R}$ or $K = \mathbf{C}$), a *differentiable scalar field* defined on M is a map

$$f : M \longrightarrow K$$

that is k-times continuously differentiable.

The class `DiffScalarField` is a Sage *element* class, whose *parent* class is `DiffScalarFieldAlgebra`. It inherits from the class `ScalarField` devoted to generic continuous scalar fields on topological manifolds.

INPUT:

- `parent` – the algebra of scalar fields containing the scalar field (must be an instance of class `DiffScalarFieldAlgebra`)
- `coord_expression` – (default: `None`) coordinate expression(s) of the scalar field; this can be either
  - a dictionary of coordinate expressions in various charts on the domain, with the charts as keys;
  - a single coordinate expression; if the argument `chart` is `'all'`, this expression is set to all the charts defined on the open set; otherwise, the expression is set in the specific chart provided by the argument `chart`

  NB: If `coord_expression` is `None` or incomplete, coordinate expressions can be added after the creation of the object, by means of the methods `add_expr()`, `add_expr_by_continuation()` and `set_expr()`
- `chart` – (default: `None`) chart defining the coordinates used in `coord_expression` when the latter is a single coordinate expression; if none is provided (default), the default chart of the open set is assumed. If `chart=='all'`, `coord_expression` is assumed to be independent of the chart (constant scalar field).

- `name` – (default: `None`) string; name (symbol) given to the scalar field
- `latex_name` – (default: `None`) string; LaTeX symbol to denote the scalar field; if none is provided, the LaTeX symbol is set to `name`

EXAMPLES:

A scalar field on the 2-sphere:

```
sage: M = Manifold(2, 'M') # the 2-dimensional sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V)   # S^2 is the union of U and V
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                                intersection_name='W',
....:                                restrictions1= x^2+y^2!=0,
....:                                restrictions2= u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: f = M.scalar_field({c_xy: 1/(1+x^2+y^2), c_uv: (u^2+v^2)/(1+u^2+v^2)},
....:                    name='f') ; f
Scalar field f on the 2-dimensional differentiable manifold M
sage: f.display()
f: M --> R
on U: (x, y) |--> 1/(x^2 + y^2 + 1)
on V: (u, v) |--> (u^2 + v^2)/(u^2 + v^2 + 1)
```

For scalar fields defined by a single coordinate expression, the latter can be passed instead of the dictionary over the charts:

```
sage: g = U.scalar_field(x*y, chart=c_xy, name='g') ; g
Scalar field g on the Open subset U of the 2-dimensional differentiable
 manifold M
```

The above is indeed equivalent to:

```
sage: g = U.scalar_field({c_xy: x*y}, name='g') ; g
Scalar field g on the Open subset U of the 2-dimensional differentiable
 manifold M
```

Since `c_xy` is the default chart of `U` , the argument `chart` can be skipped:

```
sage: g = U.scalar_field(x*y, name='g') ; g
Scalar field g on the Open subset U of the 2-dimensional differentiable
 manifold M
```

The scalar field g is defined on U and has an expression in terms of the coordinates (u, v) on $W = U \cap V$:

```
sage: g.display()
g: U --> R
   (x, y) |--> x*y
on W: (u, v) |--> u*v/(u^4 + 2*u^2*v^2 + v^4)
```

Scalar fields on M can also be declared with a single chart:

```
sage: f = M.scalar_field(1/(1+x^2+y^2), chart=c_xy, name='f') ; f
Scalar field f on the 2-dimensional differentiable manifold M
```

Their definition must then be completed by providing the expressions on other charts, via the method `add_expr()` , to get a global cover of the manifold:

```
sage: f.add_expr((u^2+v^2)/(1+u^2+v^2), chart=c_uv)
sage: f.display()
f: M --> R
on U: (x, y) |--> 1/(x^2 + y^2 + 1)
on V: (u, v) |--> (u^2 + v^2)/(u^2 + v^2 + 1)
```

We can even first declare the scalar field without any coordinate expression and provide them subsequently:

```
sage: f = M.scalar_field(name='f')
sage: f.add_expr(1/(1+x^2+y^2), chart=c_xy)
sage: f.add_expr((u^2+v^2)/(1+u^2+v^2), chart=c_uv)
sage: f.display()
f: M --> R
on U: (x, y) |--> 1/(x^2 + y^2 + 1)
on V: (u, v) |--> (u^2 + v^2)/(u^2 + v^2 + 1)
```

We may also use the method `add_expr_by_continuation()` to complete the coordinate definition using the analytic continuation from domains in which charts overlap:

```
sage: f = M.scalar_field(1/(1+x^2+y^2), chart=c_xy, name='f') ; f
Scalar field f on the 2-dimensional differentiable manifold M
sage: f.add_expr_by_continuation(c_uv, U.intersection(V))
sage: f.display()
f: M --> R
on U: (x, y) |--> 1/(x^2 + y^2 + 1)
on V: (u, v) |--> (u^2 + v^2)/(u^2 + v^2 + 1)
```

A scalar field can also be defined by some unspecified function of the coordinates:

```
sage: h = U.scalar_field(function('H')(x, y), name='h') ; h
Scalar field h on the Open subset U of the 2-dimensional differentiable
 manifold M
sage: h.display()
h: U --> R
   (x, y) |--> H(x, y)
on W: (u, v) |--> H(u/(u^2 + v^2), v/(u^2 + v^2))
```

We may use the argument `latex_name` to specify the LaTeX symbol denoting the scalar field if the latter is different from `name` :

```
sage: latex(f)
f
sage: f = M.scalar_field({c_xy: 1/(1+x^2+y^2), c_uv: (u^2+v^2)/(1+u^2+v^2)},
....:                    name='f', latex_name=r'\mathcal{F}')
sage: latex(f)
\mathcal{F}
```

The coordinate expression in a given chart is obtained via the method `expr()` , which returns a symbolic expression:

```
sage: f.expr(c_uv)
(u^2 + v^2)/(u^2 + v^2 + 1)
sage: type(f.expr(c_uv))
<type 'sage.symbolic.expression.Expression'>
```

The method `coord_function()` returns instead a function of the chart coordinates, i.e. an instance of `CoordFunction`:

```
sage: f.coord_function(c_uv)
(u^2 + v^2)/(u^2 + v^2 + 1)
sage: type(f.coord_function(c_uv))
<class 'sage.manifolds.coord_func_symb.CoordFunctionSymbRing_with_category.
↪element_class'>
sage: f.coord_function(c_uv).display()
(u, v) |--> (u^2 + v^2)/(u^2 + v^2 + 1)
```

The value returned by the method `expr()` is actually the coordinate expression of the chart function:

```
sage: f.expr(c_uv) is f.coord_function(c_uv).expr()
True
```

A constant scalar field is declared by setting the argument `chart` to `'all'`:

```
sage: c = M.scalar_field(2, chart='all', name='c') ; c
Scalar field c on the 2-dimensional differentiable manifold M
sage: c.display()
c: M --> R
on U: (x, y) |--> 2
on V: (u, v) |--> 2
```

A shortcut is to use the method `constant_scalar_field()`:

```
sage: c == M.constant_scalar_field(2)
True
```

The constant value can be some unspecified parameter:

```
sage: var('a')
a
sage: c = M.constant_scalar_field(a, name='c') ; c
Scalar field c on the 2-dimensional differentiable manifold M
sage: c.display()
c: M --> R
on U: (x, y) |--> a
on V: (u, v) |--> a
```

A special case of constant field is the zero scalar field:

```
sage: zer = M.constant_scalar_field(0) ; zer
Scalar field zero on the 2-dimensional differentiable manifold M
sage: zer.display()
zero: M --> R
on U: (x, y) |--> 0
on V: (u, v) |--> 0
```

It can be obtained directly by means of the function `zero_scalar_field()`:

```
sage: zer is M.zero_scalar_field()
True
```

A third way is to get it as the zero element of the algebra $C^k(M)$ of scalar fields on M (see below):

```
sage: zer is M.scalar_field_algebra().zero()
True
```

By definition, a scalar field acts on the manifold's points, sending them to elements of the manifold's base field (real numbers in the present case):

```
sage: N = M.point((0,0), chart=c_uv) # the North pole
sage: S = M.point((0,0), chart=c_xy) # the South pole
sage: E = M.point((1,0), chart=c_xy) # a point at the equator
sage: f(N)
0
sage: f(S)
1
sage: f(E)
1/2
sage: h(E)
H(1, 0)
sage: c(E)
a
sage: zer(E)
0
```

A scalar field can be compared to another scalar field:

```
sage: f == g
False
```

...to a symbolic expression:

```
sage: f == x*y
False
sage: g == x*y
True
sage: c == a
True
```

...to a number:

```
sage: f == 2
False
sage: zer == 0
True
```

...to anything else:

```
sage: f == M
False
```

Standard mathematical functions are implemented:

```
sage: sqrt(f)
Scalar field sqrt(f) on the 2-dimensional differentiable manifold M
sage: sqrt(f).display()
sqrt(f): M --> R
on U: (x, y) |--> 1/sqrt(x^2 + y^2 + 1)
on V: (u, v) |--> sqrt(u^2 + v^2)/sqrt(u^2 + v^2 + 1)
```

```
sage: tan(f)
Scalar field tan(f) on the 2-dimensional differentiable manifold M
sage: tan(f).display()
tan(f): M --> R
on U: (x, y) |--> sin(1/(x^2 + y^2 + 1))/cos(1/(x^2 + y^2 + 1))
on V: (u, v) |--> sin((u^2 + v^2)/(u^2 + v^2 + 1))/cos((u^2 + v^2)/(u^2 + v^2 +␣
↪1))
```

Arithmetics of scalar fields

Scalar fields on M (resp. U) belong to the algebra $C^k(M)$ (resp. $C^k(U)$):

```
sage: f.parent()
Algebra of differentiable scalar fields on the 2-dimensional
 differentiable manifold M
sage: f.parent() is M.scalar_field_algebra()
True
sage: g.parent()
Algebra of differentiable scalar fields on the Open subset U of the
 2-dimensional differentiable manifold M
sage: g.parent() is U.scalar_field_algebra()
True
```

Consequently, scalar fields can be added:

```
sage: s = f + c ; s
Scalar field f+c on the 2-dimensional differentiable manifold M
sage: s.display()
f+c: M --> R
on U: (x, y) |--> (a*x^2 + a*y^2 + a + 1)/(x^2 + y^2 + 1)
on V: (u, v) |--> ((a + 1)*u^2 + (a + 1)*v^2 + a)/(u^2 + v^2 + 1)
```

and subtracted:

```
sage: s = f - c ; s
Scalar field f-c on the 2-dimensional differentiable manifold M
sage: s.display()
f-c: M --> R
on U: (x, y) |--> -(a*x^2 + a*y^2 + a - 1)/(x^2 + y^2 + 1)
on V: (u, v) |--> -((a - 1)*u^2 + (a - 1)*v^2 + a)/(u^2 + v^2 + 1)
```

Some tests:

```
sage: f + zer == f
True
sage: f - f == zer
True
sage: f + (-f) == zer
True
sage: (f+c)-f == c
True
sage: (f-c)+c == f
True
```

We may add a number (interpreted as a constant scalar field) to a scalar field:

```
sage: s = f + 1 ; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M --> R
on U: (x, y) |--> (x^2 + y^2 + 2)/(x^2 + y^2 + 1)
on V: (u, v) |--> (2*u^2 + 2*v^2 + 1)/(u^2 + v^2 + 1)
sage: (f+1)-1 == f
True
```

The number can represented by a symbolic variable:

```
sage: s = a + f ; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s == c + f
True
```

However if the symbolic variable is a chart coordinate, the addition is performed only on the chart domain:

```
sage: s = f + x; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M --> R
on U: (x, y) |--> (x^3 + x*y^2 + x + 1)/(x^2 + y^2 + 1)
sage: s = f + u; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M --> R
on V: (u, v) |--> (u^3 + (u + 1)*v^2 + u^2 + u)/(u^2 + v^2 + 1)
```

The addition of two scalar fields with different domains is possible if the domain of one of them is a subset of the domain of the other; the domain of the result is then this subset:

```
sage: f.domain()
2-dimensional differentiable manifold M
sage: g.domain()
Open subset U of the 2-dimensional differentiable manifold M
sage: s = f + g ; s
Scalar field on the Open subset U of the 2-dimensional differentiable
 manifold M
sage: s.domain()
Open subset U of the 2-dimensional differentiable manifold M
sage: s.display()
U --> R
(x, y) |--> (x*y^3 + (x^3 + x)*y + 1)/(x^2 + y^2 + 1)
on W: (u, v) |--> (u^6 + 3*u^4*v^2 + 3*u^2*v^4 + v^6 + u*v^3
 + (u^3 + u)*v)/(u^6 + v^6 + (3*u^2 + 1)*v^4 + u^4 + (3*u^4 + 2*u^2)*v^2)
```

The operation actually performed is $f|_U + g$:

```
sage: s == f.restrict(U) + g
True
```

In Sage framework, the addition of f and g is permitted because there is a *coercion* of the parent of f, namely $C^k(M)$, to the parent of g, namely $C^k(U)$ (see `DiffScalarFieldAlgebra`):

```
sage: CM = M.scalar_field_algebra()
sage: CU = U.scalar_field_algebra()
```

```
sage: CU.has_coerce_map_from(CM)
True
```

The coercion map is nothing but the restriction to domain U:

```
sage: CU.coerce(f) == f.restrict(U)
True
```

Since the algebra $C^k(M)$ is a vector space over $\mathbf{R}$, scalar fields can be multiplied by a number, either an explicit one:

```
sage: s = 2*f ; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M --> R
on U: (x, y) |--> 2/(x^2 + y^2 + 1)
on V: (u, v) |--> 2*(u^2 + v^2)/(u^2 + v^2 + 1)
```

or a symbolic one:

```
sage: s = a*f ; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M --> R
on U: (x, y) |--> a/(x^2 + y^2 + 1)
on V: (u, v) |--> (u^2 + v^2)*a/(u^2 + v^2 + 1)
```

However, if the symbolic variable is a chart coordinate, the multiplication is performed only in the corresponding chart:

```
sage: s = x*f; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M --> R
on U: (x, y) |--> x/(x^2 + y^2 + 1)
sage: s = u*f; s
Scalar field on the 2-dimensional differentiable manifold M
sage: s.display()
M --> R
on V: (u, v) |--> (u^2 + v^2)*u/(u^2 + v^2 + 1)
```

Some tests:

```
sage: 0*f == 0
True
sage: 0*f == zer
True
sage: 1*f == f
True
sage: (-2)*f == - f - f
True
```

The ring multiplication of the algebras $C^k(M)$ and $C^k(U)$ is the pointwise multiplication of functions:

```
sage: s = f*f ; s
Scalar field f*f on the 2-dimensional differentiable manifold M
sage: s.display()
f*f: M --> R
```

```
on U: (x, y) |--> 1/(x^4 + y^4 + 2*(x^2 + 1)*y^2 + 2*x^2 + 1)
on V: (u, v) |--> (u^4 + 2*u^2*v^2 + v^4)/(u^4 + v^4 + 2*(u^2 + 1)*v^2 + 2*u^2 +
↪1)
sage: s = g*h ; s
Scalar field g*h on the Open subset U of the 2-dimensional
 differentiable manifold M
sage: s.display()
g*h: U --> R
   (x, y) |--> x*y*H(x, y)
on W: (u, v) |--> u*v*H(u/(u^2 + v^2), v/(u^2 + v^2))/(u^4 + 2*u^2*v^2 + v^4)
```

Thanks to the coercion $C^k(M) \to C^k(U)$ mentionned above, it is possible to multiply a scalar field defined on M by a scalar field defined on U, the result being a scalar field defined on U:

```
sage: f.domain(), g.domain()
(2-dimensional differentiable manifold M,
 Open subset U of the 2-dimensional differentiable manifold M)
sage: s = f*g ; s
Scalar field on the Open subset U of the 2-dimensional differentiable
 manifold M
sage: s.display()
U --> R
(x, y) |--> x*y/(x^2 + y^2 + 1)
on W: (u, v) |--> u*v/(u^4 + v^4 + (2*u^2 + 1)*v^2 + u^2)
sage: s == f.restrict(U)*g
True
```

Scalar fields can be divided (pointwise division):

```
sage: s = f/c ; s
Scalar field f/c on the 2-dimensional differentiable manifold M
sage: s.display()
f/c: M --> R
on U: (x, y) |--> 1/(a*x^2 + a*y^2 + a)
on V: (u, v) |--> (u^2 + v^2)/(a*u^2 + a*v^2 + a)
sage: s = g/h ; s
Scalar field g/h on the Open subset U of the 2-dimensional
 differentiable manifold M
sage: s.display()
g/h: U --> R
   (x, y) |--> x*y/H(x, y)
on W: (u, v) |--> u*v/((u^4 + 2*u^2*v^2 + v^4)*H(u/(u^2 + v^2), v/(u^2 + v^2)))
sage: s = f/g ; s
Scalar field on the Open subset U of the 2-dimensional differentiable
 manifold M
sage: s.display()
U --> R
(x, y) |--> 1/(x*y^3 + (x^3 + x)*y)
on W: (u, v) |--> (u^6 + 3*u^4*v^2 + 3*u^2*v^4 + v^6)/(u*v^3 + (u^3 + u)*v)
sage: s == f.restrict(U)/g
True
```

For scalar fields defined on a single chart domain, we may perform some arithmetics with symbolic expressions involving the chart coordinates:

```
sage: s = g + x^2 - y ; s
Scalar field on the Open subset U of the 2-dimensional differentiable
 manifold M
```

```
sage: s.display()
U --> R
(x, y) |--> x^2 + (x - 1)*y
on W: (u, v) |--> -(v^3 - u^2 + (u^2 - u)*v)/(u^4 + 2*u^2*v^2 + v^4)
```

```
sage: s = g*x ; s
Scalar field on the Open subset U of the 2-dimensional differentiable
 manifold M
sage: s.display()
U --> R
(x, y) |--> x^2*y
on W: (u, v) |--> u^2*v/(u^6 + 3*u^4*v^2 + 3*u^2*v^4 + v^6)
```

```
sage: s = g/x ; s
Scalar field on the Open subset U of the 2-dimensional differentiable
 manifold M
sage: s.display()
U --> R
(x, y) |--> y
on W: (u, v) |--> v/(u^2 + v^2)
sage: s = x/g ; s
Scalar field on the Open subset U of the 2-dimensional differentiable
 manifold M
sage: s.display()
U --> R
(x, y) |--> 1/y
on W: (u, v) |--> (u^2 + v^2)/v
```

The test suite is passed:

```
sage: TestSuite(f).run()
sage: TestSuite(zer).run()
```

2.4 Differentiable Maps

2.4.1 Sets of Morphisms between Differentiable Manifolds

The class `DifferentiableManifoldHomset` implements sets of morphisms between two differentiable manifolds over the same topological field K (in most applications, $K = \mathbf{R}$ or $K = \mathbf{C}$), a morphism being a *differentiable map* for the category of differentiable manifolds.

AUTHORS:

- Eric Gourgoulhon (2015): initial version

REFERENCES:

- *[Lee13]* J.M. Lee : *Introduction to Smooth Manifolds*, 2nd ed., Springer (New York) (2013)
- *[KN63]* S. Kobayashi & K. Nomizu : *Foundations of Differential Geometry*, vol. 1, Interscience Publishers (New York) (1963)

class sage.manifolds.differentiable.manifold_homset.**DifferentiableManifoldHomset** (*domain*, *codomain*, *name=None*, *latex_name=Non*

Bases: *sage.manifolds.manifold_homset.TopologicalManifoldHomset*

Set of differentiable maps between two differentiable manifolds.

Given two differentiable manifolds M and N over a topological field K, the class *DifferentiableManifoldHomset* implements the set $\mathrm{Hom}(M,N)$ of morphisms (i.e. differentiable maps) $M \to N$.

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 , Hom(M,N) will be used
- latex_name – (default: None) string; LaTeX symbol to denote the homset; if None , $\mathrm{Hom}(M,N)$ will be used

EXAMPLES:

Set of differentiable maps between a 2-dimensional differentiable manifold and a 3-dimensional one:

```
sage: M = Manifold(2, 'M')
sage: X.<x,y> = M.chart()
sage: N = Manifold(3, 'N')
sage: Y.<u,v,w> = N.chart()
sage: H = Hom(M, N) ; H
Set of Morphisms from 2-dimensional differentiable manifold M to
 3-dimensional differentiable manifold N in Category of smooth
 manifolds over Real Field with 53 bits of precision
sage: type(H)
<class 'sage.manifolds.differentiable.manifold_homset.
↪DifferentiableManifoldHomset_with_category'>
sage: H.category()
Category of homsets of topological spaces
sage: latex(H)
\mathrm{Hom}\left(M,N\right)
sage: H.domain()
2-dimensional differentiable manifold M
sage: H.codomain()
3-dimensional differentiable manifold N
```

An element of H is a differentiable map from M to N :

```
sage: H.Element
<class 'sage.manifolds.differentiable.diff_map.DiffMap'>
sage: f = H.an_element() ; f
Differentiable map from the 2-dimensional differentiable manifold M to the
 3-dimensional differentiable manifold N
sage: f.display()
M --> N
   (x, y) |--> (u, v, w) = (0, 0, 0)
```

The test suite is passed:

```
sage: TestSuite(H).run()
```

When the codomain coincides with the domain, the homset is a set of *endomorphisms* in the category of differentiable manifolds:

```
sage: E = Hom(M, M) ; E
Set of Morphisms from 2-dimensional differentiable manifold M to
 2-dimensional differentiable manifold M in Category of smooth
 manifolds over Real Field with 53 bits of precision
sage: E.category()
Category of endsets of topological spaces
sage: E.is_endomorphism_set()
True
sage: E is End(M)
True
```

In this case, the homset is a monoid for the law of morphism composition:

```
sage: E in Monoids()
True
```

This was of course not the case for `H = Hom(M,N)`:

```
sage: H in Monoids()
False
```

The identity element of the monoid is of course the identity map of `M`:

```
sage: E.one()
Identity map Id_M of the 2-dimensional differentiable manifold M
sage: E.one() is M.identity_map()
True
sage: E.one().display()
Id_M: M --> M
   (x, y) |--> (x, y)
```

The test suite is passed by `E`:

```
sage: TestSuite(E).run()
```

This test suite includes more tests than in the case of `H`, since `E` has some extra structure (monoid).

Element
  alias of `DiffMap`

2.4.2 Differentiable Maps between Differentiable Manifolds

The class *`DiffMap`* implements differentiable maps from a differentiable manifold M to a differentiable manifold N over the same topological field K as M (in most applications, $K = \mathbf{R}$ or $K = \mathbf{C}$):

$$\Phi : M \longrightarrow N$$

AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2013-2015): initial version

REFERENCES:

- Chap. 1 of *[KN63]* S. Kobayashi & K. Nomizu : *Foundations of Differential Geometry*, vol. 1, Interscience Publishers (New York) (1963)
- Chaps. 2 and 3 of *[Lee13]* J.M. Lee : *Introduction to Smooth Manifolds*, 2nd ed., Springer (New York) (2013)

class `sage.manifolds.differentiable.diff_map.`**`DiffMap`** (*parent*, *coord_functions=None*, *name=None*, *latex_name=None*, *is_isomorphism=False*, *is_identity=False*)

Bases: *`sage.manifolds.continuous_map.ContinuousMap`*

Differentiable map between two differentiable manifolds.

This class implements differentiable maps of the type

$$\Phi : M \longrightarrow N$$

where M and N are differentiable manifolds over the same topological field K (in most applications, $K = \mathbf{R}$ or $K = \mathbf{C}$).

Differentiable maps are the *morphisms* of the *category* of differentiable manifolds. The set of all differentiable maps from M to N is therefore the homset between M and N, which is denoted by $\mathrm{Hom}(M, N)$.

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 $\mathrm{Hom}(M, N)$ to which the differentiable map belongs
- `coord_functions` – (default: `None`) if not `None`, must be a dictionary of the coordinate expressions (as lists (or tuples) of the coordinates of the image expressed in terms of the coordinates of the considered point) with the pairs of charts (chart1, chart2) as keys (chart1 being a chart on M and chart2 a chart on N). If the dimension of the map's codomain is 1, a single coordinate expression can be passed instead of a tuple with a single element
- `name` – (default: `None`) name given to the differentiable map
- `latex_name` – (default: `None`) LaTeX symbol to denote the differentiable map; if `None`, the LaTeX symbol is set to `name`
- `is_isomorphism` – (default: `False`) determines whether the constructed object is a isomorphism (i.e. a diffeomorphism); if set to `True`, then the manifolds M and N must have the same dimension.
- `is_identity` – (default: `False`) determines whether the constructed object is the identity map; if set to `True`, then N must be M and the entry `coord_functions` is not used.

Note: If the information passed by means of the argument `coord_functions` is not sufficient to fully specify the differentiable map, further coordinate expressions, in other charts, can be subsequently added by means of the method *`add_expr()`*

EXAMPLES:

The standard embedding of the sphere S^2 into $\mathbf{R}^3$:

```
sage: M = Manifold(2, 'S^2') # the 2-dimensional sphere S^2
sage: U = M.open_subset('U') # complement of the North pole
sage: c_xy.<x,y> = U.chart() # stereographic coordinates from the North pole
```

```
sage: V = M.open_subset('V') # complement of the South pole
sage: c_uv.<u,v> = V.chart() # stereographic coordinates from the South pole
sage: M.declare_union(U,V)   # S^2 is the union of U and V
sage: xy_to_uv = c_xy.transition_map(c_uv, (x/(x^2+y^2), y/(x^2+y^2)),
....:                intersection_name='W', restrictions1= x^2+y^2!=0,
....:                restrictions2= u^2+v^2!=0)
sage: uv_to_xy = xy_to_uv.inverse()
sage: N = Manifold(3, 'R^3', r'\RR^3')  # R^3
sage: c_cart.<X,Y,Z> = N.chart()  # Cartesian coordinates on R^3
sage: Phi = M.diff_map(N,
....: {(c_xy, c_cart): [2*x/(1+x^2+y^2), 2*y/(1+x^2+y^2), (x^2+y^2-1)/(1+x^2+y^
↪2)],
....:  (c_uv, c_cart): [2*u/(1+u^2+v^2), 2*v/(1+u^2+v^2), (1-u^2-v^2)/(1+u^2+v^
↪2)]},
....: name='Phi', latex_name=r'\Phi')
sage: Phi
Differentiable map Phi from the 2-dimensional differentiable manifold
 S^2 to the 3-dimensional differentiable manifold R^3
sage: Phi.parent()
Set of Morphisms from 2-dimensional differentiable manifold S^2 to
 3-dimensional differentiable manifold R^3 in Category of smooth
 manifolds over Real Field with 53 bits of precision
sage: Phi.parent() is Hom(M, N)
True
sage: type(Phi)
<class 'sage.manifolds.differentiable.diff_map.DifferentiableManifoldHomset_with_
↪category.element_class'>
sage: Phi.display()
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1),
 (x^2 + y^2 - 1)/(x^2 + y^2 + 1))
on V: (u, v) |--> (X, Y, Z) = (2*u/(u^2 + v^2 + 1), 2*v/(u^2 + v^2 + 1),
 -(u^2 + v^2 - 1)/(u^2 + v^2 + 1))
```

It is possible to create the map via the method *diff_map()* only in a single pair of charts: the argument `coord_functions` is then a mere list of coordinate expressions (and not a dictionary) and the arguments `chart1` and `chart2` have to be provided if the charts differ from the default ones on the domain and/or the codomain:

```
sage: Phi1 = M.diff_map(N, [2*x/(1+x^2+y^2), 2*y/(1+x^2+y^2),
....:                        (x^2+y^2-1)/(1+x^2+y^2)],
....:                     chart1=c_xy, chart2=c_cart, name='Phi',
....:                     latex_name=r'\Phi')
```

Since `c_xy` and `c_cart` are the default charts on respectively `M` and `N`, they can be omitted, so that the above declaration is equivalent to:

```
sage: Phi1 = M.diff_map(N, [2*x/(1+x^2+y^2), 2*y/(1+x^2+y^2),
....:                        (x^2+y^2-1)/(1+x^2+y^2)],
....:                     name='Phi', latex_name=r'\Phi')
```

With such a declaration, the differentiable map is only partially defined on the manifold S^2, being known in only one chart:

```
sage: Phi1.display()
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1),
 (x^2 + y^2 - 1)/(x^2 + y^2 + 1))
```

The definition can be completed by means of the method *add_expr()* :

```
sage: Phi1.add_expr(c_uv, c_cart, [2*u/(1+u^2+v^2), 2*v/(1+u^2+v^2),
....:                              (1-u^2-v^2)/(1+u^2+v^2)])
sage: Phi1.display()
Phi: S^2 --> R^3
on U: (x, y) |--> (X, Y, Z) = (2*x/(x^2 + y^2 + 1), 2*y/(x^2 + y^2 + 1),
 (x^2 + y^2 - 1)/(x^2 + y^2 + 1))
on V: (u, v) |--> (X, Y, Z) = (2*u/(u^2 + v^2 + 1), 2*v/(u^2 + v^2 + 1),
 -(u^2 + v^2 - 1)/(u^2 + v^2 + 1))
```

At this stage, `Phi1` and `Phi` are fully equivalent:

```
sage: Phi1 == Phi
True
```

The test suite is passed:

```
sage: TestSuite(Phi).run()
sage: TestSuite(Phi1).run()
```

The map acts on points:

```
sage: np = M.point((0,0), chart=c_uv, name='N')  # the North pole
sage: Phi(np)
Point Phi(N) on the 3-dimensional differentiable manifold R^3
sage: Phi(np).coord() # Cartesian coordinates
(0, 0, 1)
sage: sp = M.point((0,0), chart=c_xy, name='S')  # the South pole
sage: Phi(sp).coord() # Cartesian coordinates
(0, 0, -1)
```

Differentiable maps can be composed by means of the operator `*` : let us introduce the map $\mathbf{R}^3 \to \mathbf{R}^2$ corresponding to the projection from the point $(X, Y, Z) = (0, 0, 1)$ onto the equatorial plane $Z = 0$:

```
sage: P = Manifold(2, 'R^2', r'\RR^2') # R^2 (equatorial plane)
sage: cP.<xP, yP> = P.chart()
sage: Psi = N.diff_map(P, (X/(1-Z), Y/(1-Z)), name='Psi',
....:                  latex_name=r'\Psi')
sage: Psi
Differentiable map Psi from the 3-dimensional differentiable manifold
 R^3 to the 2-dimensional differentiable manifold R^2
sage: Psi.display()
Psi: R^3 --> R^2
   (X, Y, Z) |--> (xP, yP) = (-X/(Z - 1), -Y/(Z - 1))
```

Then we compose `Psi` with `Phi` , thereby getting a map $S^2 \to \mathbf{R}^2$:

```
sage: ster = Psi*Phi ; ster
Differentiable map from the 2-dimensional differentiable manifold S^2
 to the 2-dimensional differentiable manifold R^2
```

Let us test on the South pole (`sp`) that `ster` is indeed the composite of `Psi` and `Phi` :

```
sage: ster(sp) == Psi(Phi(sp))
True
```

Actually `ster` is the stereographic projection from the North pole, as its coordinate expression reveals:

```
sage: ster.display()
S^2 --> R^2
on U: (x, y) |--> (xP, yP) = (x, y)
on V: (u, v) |--> (xP, yP) = (u/(u^2 + v^2), v/(u^2 + v^2))
```

If its codomain is 1-dimensional, a differentiable map must be defined by a single symbolic expression for each pair of charts, and not by a list/tuple with a single element:

```
sage: N = Manifold(1, 'N')
sage: c_N = N.chart('X')
sage: Phi = M.diff_map(N, {(c_xy, c_N): x^2+y^2,
....: (c_uv, c_N): 1/(u^2+v^2)})  # not ...[1/(u^2+v^2)] or (1/(u^2+v^2),)
```

An example of differentiable map $\mathbf{R} \to \mathbf{R}^2$:

```
sage: R = Manifold(1, 'R')  # field R
sage: T.<t> = R.chart()  # canonical chart on R
sage: R2 = Manifold(2, 'R^2')  # R^2
sage: c_xy.<x,y> = R2.chart() # Cartesian coordinates on R^2
sage: Phi = R.diff_map(R2, [cos(t), sin(t)], name='Phi') ; Phi
Differentiable map Phi from the 1-dimensional differentiable manifold R
 to the 2-dimensional differentiable manifold R^2
sage: Phi.parent()
Set of Morphisms from 1-dimensional differentiable manifold R to
 2-dimensional differentiable manifold R^2 in Category of smooth
 manifolds over Real Field with 53 bits of precision
sage: Phi.parent() is Hom(R, R2)
True
sage: Phi.display()
Phi: R --> R^2
   t |--> (x, y) = (cos(t), sin(t))
```

An example of diffeomorphism between the unit open disk and the Euclidean plane $\mathbf{R}^2$:

```
sage: D = R2.open_subset('D', coord_def={c_xy: x^2+y^2<1}) # the open unit disk
sage: Phi = D.diffeomorphism(R2, [x/sqrt(1-x^2-y^2), y/sqrt(1-x^2-y^2)],
....:                             name='Phi', latex_name=r'\Phi')
sage: Phi
Diffeomorphism Phi from the Open subset D of the 2-dimensional
 differentiable manifold R^2 to the 2-dimensional differentiable
 manifold R^2
sage: Phi.parent()
Set of Morphisms from Open subset D of the 2-dimensional differentiable
 manifold R^2 to 2-dimensional differentiable manifold R^2 in Join of
 Category of subobjects of sets and Category of smooth manifolds over
 Real Field with 53 bits of precision
sage: Phi.parent() is Hom(D, R2)
True
sage: Phi.display()
Phi: D --> R^2
   (x, y) |--> (x, y) = (x/sqrt(-x^2 - y^2 + 1), y/sqrt(-x^2 - y^2 + 1))
```

The image of a point:

```
sage: p = D.point((1/2,0))
sage: q = Phi(p) ; q
Point on the 2-dimensional differentiable manifold R^2
sage: q.coord()
(1/3*sqrt(3), 0)
```

The inverse diffeomorphism is computed by means of the method `inverse()` :

```
sage: Phi.inverse()
Diffeomorphism Phi^(-1) from the 2-dimensional differentiable manifold R^2
 to the Open subset D of the 2-dimensional differentiable manifold R^2
sage: Phi.inverse().display()
Phi^(-1): R^2 --> D
   (x, y) |--> (x, y) = (x/sqrt(x^2 + y^2 + 1), y/sqrt(x^2 + y^2 + 1))
```

Equivalently, one may use the notations `^(-1)` or `~` to get the inverse:

```
sage: Phi^(-1) is Phi.inverse()
True
sage: ~Phi is Phi.inverse()
True
```

Check that `~Phi` is indeed the inverse of `Phi` :

```
sage: (~Phi)(q) == p
True
sage: Phi * ~Phi == R2.identity_map()
True
sage: ~Phi * Phi == D.identity_map()
True
```

The coordinate expression of the inverse diffeomorphism:

```
sage: (~Phi).display()
Phi^(-1): R^2 --> D
   (x, y) |--> (x, y) = (x/sqrt(x^2 + y^2 + 1), y/sqrt(x^2 + y^2 + 1))
```

A special case of diffeomorphism: the identity map of the open unit disk:

```
sage: id = D.identity_map() ; id
Identity map Id_D of the Open subset D of the 2-dimensional
 differentiable manifold R^2
sage: latex(id)
\mathrm{Id}_{D}
sage: id.parent()
Set of Morphisms from Open subset D of the 2-dimensional differentiable
 manifold R^2 to Open subset D of the 2-dimensional differentiable
 manifold R^2 in Join of Category of subobjects of sets and Category of
 smooth manifolds over Real Field with 53 bits of precision
sage: id.parent() is Hom(D, D)
True
sage: id is Hom(D,D).one()   # the identity element of the monoid Hom(D,D)
True
```

The identity map acting on a point:

```
sage: id(p)
Point on the 2-dimensional differentiable manifold R^2
```

```
sage: id(p) == p
True
sage: id(p) is p
True
```

The coordinate expression of the identity map:

```
sage: id.display()
Id_D: D --> D
   (x, y) |--> (x, y)
```

The identity map is its own inverse:

```
sage: id^(-1) is id
True
sage: ~id is id
True
```

CHAPTER

THREE

UTILITIES FOR CALCULUS

This module defines helper functions which are used for simplifications and display of symbolic expressions.

AUTHORS:

- Michal Bejger (2015) : class *ExpressionNice*
- Eric Gourgoulhon (2015) : simplification functions
- Travis Scrimshaw (2016): review tweaks

class `sage.manifolds.utilities.`**`ExpressionNice`**(*ex*)

Bases: `sage.symbolic.expression.Expression`

Subclass of `Expression` for a "human-friendly" display of partial derivatives and the possibility to shorten the display by skipping the arguments of symbolic functions.

INPUT:

- `ex` – symbolic expression

EXAMPLES:

An expression formed with callable symbolic expressions:

```
sage: var('x y z')
(x, y, z)
sage: f = function('f')(x, y)
sage: g = f.diff(y).diff(x)
sage: h = function('h')(y, z)
sage: k = h.diff(z)
sage: fun = x*g + y*(k-z)^2
```

The standard Pynac display of partial derivatives:

```
sage: fun
y*(z - D[1](h)(y, z))^2 + x*D[0, 1](f)(x, y)
sage: latex(fun)
y {\left(z - D[1]\left(h\right)\left(y, z\right)\right)}^{2}
 + x D[0, 1]\left(f\right)\left(x, y\right)
```

With *ExpressionNice*, the Pynac notation `D[...]` is replaced by textbook-like notation:

```
sage: from sage.manifolds.utilities import ExpressionNice
sage: ExpressionNice(fun)
y*(z - d(h)/dz)^2 + x*d^2(f)/dxdy
sage: latex(ExpressionNice(fun))
y {\left(z - \frac{\partial\,h}{\partial z}\right)}^{2}
 + x \frac{\partial^2\,f}{\partial x\partial y}
```

An example when function variables are themselves functions:

```
sage: f = function('f')(x, y)
sage: g = function('g')(x, f)  # the second variable is the function f
sage: fun = (g.diff(x))*x - x^2*f.diff(x,y)
sage: fun
-x^2*D[0, 1](f)(x, y) + (D[0](f)(x, y)*D[1](g)(x, f(x, y)) + D[0](g)(x, f(x,␣
↪y)))*x
sage: ExpressionNice(fun)
-x^2*d^2(f)/dxdy + (d(f)/dx*d(g)/d(f(x, y)) + d(g)/dx)*x
sage: latex(ExpressionNice(fun))
-x^{2} \frac{\partial^2\,f}{\partial x\partial y}
 + {\left(\frac{\partial\,f}{\partial x}
   \frac{\partial\,g}{\partial \left( f\left(x, y\right) \right)}
 + \frac{\partial\,g}{\partial x}\right)} x
```

Note that `D[1](g)(x,f(x,y))` is rendered as `d(g)/d(f(x,y))` .

An example with multiple differentiations:

```
sage: fun = f.diff(x,x,y,y,x)*x
sage: fun
x*D[0, 0, 0, 1, 1](f)(x, y)
sage: ExpressionNice(fun)
x*d^5(f)/dx^3dy^2
sage: latex(ExpressionNice(fun))
x \frac{\partial^5\,f}{\partial x ^ 3\partial y ^ 2}
```

Parentheses are added around powers of partial derivatives to avoid any confusion:

```
sage: fun = f.diff(y)^2
sage: fun
D[1](f)(x, y)^2
sage: ExpressionNice(fun)
(d(f)/dy)^2
sage: latex(ExpressionNice(fun))
\left(\frac{\partial\,f}{\partial y}\right)^{2}
```

The explicit mention of function arguments can be omitted for the sake of brevity:

```
sage: fun = fun*f
sage: ExpressionNice(fun)
f(x, y)*(d(f)/dy)^2
sage: Manifold.options.omit_function_arguments=True
sage: ExpressionNice(fun)
f*(d(f)/dy)^2
sage: latex(ExpressionNice(fun))
f \left(\frac{\partial\,f}{\partial y}\right)^{2}
sage: Manifold.options._reset()
sage: ExpressionNice(fun)
f(x, y)*(d(f)/dy)^2
sage: latex(ExpressionNice(fun))
f\left(x, y\right) \left(\frac{\partial\,f}{\partial y}\right)^{2}
```

`sage.manifolds.utilities.`**`set_axes_labels`** (*graph*, *xlabel*, *ylabel*, *zlabel*, ***kwds*)

Set axes labels for a 3D graphics object `graph` .

This is a workaround for the lack of axes labels in 3D plots. This sets the labels as `text3d()` objects at locations determined from the bounding box of the graphic object `graph` .

INPUT:

- `graph` – `Graphics3d`; a 3D graphic object
- `xlabel` – string for the x-axis label
- `ylabel` – string for the y-axis label
- `zlabel` – string for the z-axis label
- `**kwds` – options (e.g. color) for text3d

OUTPUT:

- the 3D graphic object with text3d labels added

EXAMPLES:

```
sage: g = sphere()
sage: g.all
[Graphics3d Object]
sage: from sage.manifolds.utilities import set_axes_labels
sage: ga = set_axes_labels(g, 'X', 'Y', 'Z', color='red')
sage: ga.all   # the 3D frame has now axes labels
[Graphics3d Object, Graphics3d Object,
 Graphics3d Object, Graphics3d Object]
```

`sage.manifolds.utilities.`**`simplify_abs_trig`**`(`*expr*`)`

Simplify `abs(sin(...))` in symbolic expressions.

EXAMPLES:

```
sage: forget()   # for doctests only
sage: M = Manifold(3, 'M', structure='topological')
sage: X.<x,y,z> = M.chart(r'x y:(0,pi) z:(-pi/3,0)')
sage: X.coord_range()
x: (-oo, +oo); y: (0, pi); z: (-1/3*pi, 0)
```

Since `x` spans all $\mathbf{R}$, no simplification of `abs(sin(x))` occurs, while `abs(sin(y))` and `abs(sin(3*z))` are correctly simplified, given that $y \in (0, \pi)$ and $z \in (-\pi/3, 0)$:

```
sage: from sage.manifolds.utilities import simplify_abs_trig
sage: simplify_abs_trig( abs(sin(x)) + abs(sin(y)) + abs(sin(3*z)) )
abs(sin(x)) + sin(y) - sin(3*z)
```

Note that neither Sage's function `simplify_trig()` nor `simplify_full()` works in this case:

```
sage: s = abs(sin(x)) + abs(sin(y)) + abs(sin(3*z))
sage: s.simplify_trig()
abs(4*cos(z)^2 - 1)*abs(sin(z)) + abs(sin(x)) + abs(sin(y))
sage: s.simplify_full()
abs(4*cos(z)^2 - 1)*abs(sin(z)) + abs(sin(x)) + abs(sin(y))
```

despite the following assumptions hold:

```
sage: assumptions()
[x is real, y is real, y > 0, y < pi, z is real, z > -1/3*pi, z < 0]
```

Additional checks are:

```
sage: simplify_abs_trig( abs(sin(y/2)) )  # shall simplify
sin(1/2*y)
sage: simplify_abs_trig( abs(sin(2*y)) )  # must not simplify
abs(sin(2*y))
sage: simplify_abs_trig( abs(sin(z/2)) )  # shall simplify
-sin(1/2*z)
sage: simplify_abs_trig( abs(sin(4*z)) )  # must not simplify
abs(sin(4*z))
```

sage.manifolds.utilities.**simplify_chain_generic**(*expr*)

Apply a chain of simplifications to a symbolic expression.

This is the simplification chain used in calculus involving coordinate functions on manifolds over fields different from $\mathbf{R}$, as implemented in *CoordFunctionSymb* .

The chain is formed by the following functions, called successively:

1. simplify_factorial()
2. simplify_rectform()
3. simplify_trig()
4. simplify_rational()
5. expand_sum()

NB: for the time being, this is identical to simplify_full() .

EXAMPLES:

We consider variables that are coordinates of a chart on a complex manifold:

```
sage: forget()  # for doctest only
sage: M = Manifold(2, 'M', structure='topological', field='complex')
sage: X.<x,y> = M.chart()
```

Then neither x nor y is assumed to be real:

```
sage: assumptions()
[]
```

Accordingly, simplify_chain_generic does not simplify sqrt(x^2) to abs(x) :

```
sage: from sage.manifolds.utilities import simplify_chain_generic
sage: s = sqrt(x^2)
sage: simplify_chain_generic(s)
sqrt(x^2)
```

This contrasts with the behavior of *simplify_chain_real()* .

Other simplifications:

```
sage: s = (x+y)^2 - x^2 -2*x*y - y^2
sage: simplify_chain_generic(s)
0
sage: s = (x^2 - 2*x + 1) / (x^2 -1)
sage: simplify_chain_generic(s)
(x - 1)/(x + 1)
sage: s = cos(2*x) - 2*cos(x)^2 + 1
sage: simplify_chain_generic(s)
0
```

`sage.manifolds.utilities.`**`simplify_chain_real`**`(`*expr*`)`

Apply a chain of simplifications to a symbolic expression, assuming the real domain.

This is the simplification chain used in calculus involving coordinate functions on real manifolds, as implemented in *`CoordFunctionSymb`*.

The chain is formed by the following functions, called successively:

1. `simplify_factorial()`
2. `simplify_trig()`
3. `simplify_rational()`
4. *`simplify_sqrt_real()`*
5. *`simplify_abs_trig()`*
6. `canonicalize_radical()`
7. `simplify_log()`
8. `simplify_rational()`
9. `simplify_trig()`

EXAMPLES:

We consider variables that are coordinates of a chart on a real manifold:

```
sage: forget()  # for doctest only
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart('x:(0,1) y')
```

The following assumptions then hold:

```
sage: assumptions()
[x is real, x > 0, x < 1, y is real]
```

and we have:

```
sage: from sage.manifolds.utilities import simplify_chain_real
sage: s = sqrt(y^2)
sage: simplify_chain_real(s)
abs(y)
```

The above result is correct since `y` is real. It is obtained by `simplify_real()` as well, but not by `simplify_full()`:

```
sage: s.simplify_real()
abs(y)
sage: s.simplify_full()
sqrt(y^2)
```

Furthermore, we have:

```
sage: s = sqrt(x^2-2*x+1)
sage: simplify_chain_real(s)
-x + 1
```

which is correct since $x \in (0,1)$. On this example, neither `simplify_real()` nor `simplify_full()`, nor `canonicalize_radical()` give satisfactory results:

```
sage: s.simplify_real()  # unsimplified output
sqrt(x^2 - 2*x + 1)
sage: s.simplify_full()  # unsimplified output
sqrt(x^2 - 2*x + 1)
sage: s.canonicalize_radical()  # wrong output since x in (0,1)
x - 1
```

Other simplifications:

```
sage: s = abs(sin(pi*x))
sage: simplify_chain_real(s)  # correct output since x in (0,1)
sin(pi*x)
sage: s.simplify_real()  # unsimplified output
abs(sin(pi*x))
sage: s.simplify_full()  # unsimplified output
abs(sin(pi*x))
```

```
sage: s = cos(y)^2 + sin(y)^2
sage: simplify_chain_real(s)
1
sage: s.simplify_real()  # unsimplified output
cos(y)^2 + sin(y)^2
sage: s.simplify_full()  # OK
1
```

sage.manifolds.utilities.**simplify_sqrt_real**(*expr*)

Simplify `sqrt` in symbolic expressions in the real domain.

EXAMPLES:

Simplifications of basic expressions:

```
sage: from sage.manifolds.utilities import simplify_sqrt_real
sage: simplify_sqrt_real( sqrt(x^2) )
abs(x)
sage: assume(x<0)
sage: simplify_sqrt_real( sqrt(x^2) )
-x
sage: simplify_sqrt_real( sqrt(x^2-2*x+1) )
-x + 1
sage: simplify_sqrt_real( sqrt(x^2) + sqrt(x^2-2*x+1) )
-2*x + 1
```

This improves over Sage's `canonicalize_radical()`, which yields incorrect results when `x < 0`:

```
sage: forget()  # removes the assumption x<0
sage: sqrt(x^2).canonicalize_radical()
x
sage: assume(x<0)
sage: sqrt(x^2).canonicalize_radical() # wrong output
x
sage: sqrt(x^2-2*x+1).canonicalize_radical() # wrong output
x - 1
sage: ( sqrt(x^2) + sqrt(x^2-2*x+1) ).canonicalize_radical() # wrong output
2*x - 1
```

Simplification of nested `sqrt` 's:

```
sage: forget()  # removes the assumption x<0
sage: simplify_sqrt_real( sqrt(1 + sqrt(x^2)) )
sqrt(abs(x) + 1)
sage: assume(x<0)
sage: simplify_sqrt_real( sqrt(1 + sqrt(x^2)) )
sqrt(-x + 1)
sage: simplify_sqrt_real( sqrt(x^2 + sqrt(4*x^2) + 1) )
-x + 1
```

Again, `canonicalize_radical()` fails on the last one:

```
sage: (sqrt(x^2 + sqrt(4*x^2) + 1)).canonicalize_radical()  # wrong output
x + 1
```

CHAPTER

FOUR

INDICES AND TABLES

- Index
- Module Index
- Search Page

[Lee11] J.M. Lee : *Introduction to Topological Manifolds*, 2nd ed., Springer (New York) (2011).

[Lee13] J.M. Lee : *Introduction to Smooth Manifolds*, 2nd ed., Springer (New York) (2013)

[KN63] S. Kobayashi & K. Nomizu : *Foundations of Differential Geometry*, vol. 1, Interscience Publishers (New York) (1963).

[Huybrechts05] D. Huybrechts : *Complex Geometry*, Springer (Berlin) (2005).

PYTHON MODULE INDEX

INDEX