Euclidean Spaces and Vector Calculus

Release 9.6

The Sage Development Team

May 16, 2022

CONTENTS

CHAPTER

ONE

EUCLIDEAN SPACES

An *Euclidean space of dimension* n is an affine space E, whose associated vector space is a n-dimensional vector space over $\mathbf{R}$ and is equipped with a positive definite symmetric bilinear form, called the *scalar product* or *dot product* [Ber1987]. An Euclidean space of dimension n can also be viewed as a Riemannian manifold that is diffeomorphic to $\mathbf{R}^n$ and that has a flat metric g. The Euclidean scalar product is then that defined by the Riemannian metric g.

The current implementation of Euclidean spaces is based on the second point of view. This allows for the introduction of various coordinate systems in addition to the usual the Cartesian systems. Standard curvilinear systems (planar, spherical and cylindrical coordinates) are predefined for 2-dimensional and 3-dimensional Euclidean spaces, along with the corresponding transition maps between them. Another benefit of such an implementation is the direct use of methods for vector calculus already implemented at the level of Riemannian manifolds (see, e.g., the methods `cross_product()` and `curl()`, as well as the module *operators*).

Euclidean spaces are implemented via the following classes:

- *EuclideanSpace* for generic values n,
- *EuclideanPlane* for $n = 2$,
- *Euclidean3dimSpace* for $n = 3$.

The user interface is provided by *EuclideanSpace*.

Example 1: the Euclidean plane

We start by declaring the Euclidean plane `E`, with `(x, y)` as Cartesian coordinates:

```
sage: E.<x,y> = EuclideanSpace()
sage: E
Euclidean plane E^2
sage: dim(E)
2
```

`E` is automatically endowed with the chart of Cartesian coordinates:

```
sage: E.atlas()
[Chart (E^2, (x, y))]
sage: cartesian = E.default_chart(); cartesian
Chart (E^2, (x, y))
```

Thanks to the use of `<x,y>` when declaring `E`, the coordinates (x, y) have been injected in the global namespace, i.e. the Python variables `x` and `y` have been created and are available to form symbolic expressions:

```
sage: y
y
sage: type(y)
<class 'sage.symbolic.expression.Expression'>
sage: assumptions()
[x is real, y is real]
```

The metric tensor of E is predefined:

```
sage: g = E.metric(); g
Riemannian metric g on the Euclidean plane E^2
sage: g.display()
g = dx⊗dx + dy⊗dy
sage: g[:]
[1 0]
[0 1]
```

It is a *flat* metric, i.e. it has a vanishing Riemann tensor:

```
sage: g.riemann()
Tensor field Riem(g) of type (1,3) on the Euclidean plane E^2
sage: g.riemann().display()
Riem(g) = 0
```

Polar coordinates (r, ϕ) are introduced by:

```
sage: polar.<r,ph> = E.polar_coordinates()
sage: polar
Chart (E^2, (r, ph))
```

E is now endowed with two coordinate charts:

```
sage: E.atlas()
[Chart (E^2, (x, y)), Chart (E^2, (r, ph))]
```

The ranges of the coordinates introduced so far are:

```
sage: cartesian.coord_range()
x: (-oo, +oo); y: (-oo, +oo)
sage: polar.coord_range()
r: (0, +oo); ph: [0, 2*pi] (periodic)
```

The transition map from polar coordinates to Cartesian ones is:

```
sage: E.coord_change(polar, cartesian).display()
x = r*cos(ph)
y = r*sin(ph)
```

while the reverse one is:

```
sage: E.coord_change(cartesian, polar).display()
r = sqrt(x^2 + y^2)
ph = arctan2(y, x)
```

A point of E is constructed from its coordinates (by default in the Cartesian chart):

```
sage: p = E((-1,1), name='p'); p
Point p on the Euclidean plane E^2
sage: p.parent()
Euclidean plane E^2
```

The coordinates of a point are obtained by letting the corresponding chart act on it:

```
sage: cartesian(p)
(-1, 1)
sage: polar(p)
(sqrt(2), 3/4*pi)
```

At this stage, E is endowed with three vector frames:

```
sage: E.frames()
[Coordinate frame (E^2, (e_x,e_y)),
 Coordinate frame (E^2, (∂/∂r,∂/∂ph)),
 Vector frame (E^2, (e_r,e_ph))]
```

The third one is the standard orthonormal frame associated with polar coordinates, as we can check from the metric components in it:

```
sage: polar_frame = E.polar_frame(); polar_frame
Vector frame (E^2, (e_r,e_ph))
sage: g[polar_frame,:]
[1 0]
[0 1]
```

The expression of the metric tensor in terms of polar coordinates is:

```
sage: g.display(polar)
g = dr⊗dr + r^2 dph⊗dph
```

A vector field on E:

```
sage: v = E.vector_field(-y, x, name='v'); v
Vector field v on the Euclidean plane E^2
sage: v.display()
v = -y e_x + x e_y
sage: v[:]
[-y, x]
```

By default, the components of v, as returned by `display` or the bracket operator, refer to the Cartesian frame on E; to get the components with respect to the orthonormal polar frame, one has to specify it explicitly, generally along with the polar chart for the coordinate expression of the components:

```
sage: v.display(polar_frame, polar)
v = r e_ph
sage: v[polar_frame,:,polar]
[0, r]
```

Note that the default frame for the display of vector fields can be changed thanks to the method `set_default_frame()`; in the same vein, the default coordinates can be changed via the method `set_default_chart()`:

```
sage: E.set_default_frame(polar_frame)
sage: E.set_default_chart(polar)
sage: v.display()
v = r e_ph
sage: v[:]
[0, r]
sage: E.set_default_frame(E.cartesian_frame())  # revert to Cartesian frame
sage: E.set_default_chart(cartesian)            # and chart
```

When defining a vector field from components relative to a vector frame different from the default one, the vector frame has to be specified explicitly:

```
sage: v = E.vector_field(1, 0, frame=polar_frame)
sage: v.display(polar_frame)
e_r
sage: v.display()
x/sqrt(x^2 + y^2) e_x + y/sqrt(x^2 + y^2) e_y
```

The argument `chart` must be used to specify in which coordinate chart the components are expressed:

```
sage: v = E.vector_field(0, r, frame=polar_frame, chart=polar)
sage: v.display(polar_frame, polar)
r e_ph
sage: v.display()
-y e_x + x e_y
```

It is also possible to pass the components as a dictionary, with a pair (vector frame, chart) as a key:

```
sage: v = E.vector_field({(polar_frame, polar): (0, r)})
sage: v.display(polar_frame, polar)
r e_ph
```

The key can be reduced to the vector frame if the chart is the default one:

```
sage: v = E.vector_field({polar_frame: (0, 1)})
sage: v.display(polar_frame)
e_ph
```

Finally, it is possible to construct the vector field without initializing any component:

```
sage: v = E.vector_field(); v
Vector field on the Euclidean plane E^2
```

The components can then by set in a second stage, via the square bracket operator, the unset components being assumed to be zero:

```
sage: v[1] = -y
sage: v.display()  # v[2] is zero
-y e_x
sage: v[2] = x
sage: v.display()
-y e_x + x e_y
```

The above is equivalent to:

```
sage: v[:] = -y, x
sage: v.display()
-y e_x + x e_y
```

The square bracket operator can also be used to set components in a vector frame that is not the default one:

```
sage: v = E.vector_field(name='v')
sage: v[polar_frame, 2, polar] = r
sage: v.display(polar_frame, polar)
v = r e_ph
sage: v.display()
v = -y e_x + x e_y
```

The value of the vector field `v` at point `p`:

```
sage: vp = v.at(p); vp
Vector v at Point p on the Euclidean plane E^2
sage: vp.display()
v = -e_x - e_y
sage: vp.display(polar_frame.at(p))
v = sqrt(2) e_ph
```

A scalar field on `E`:

```
sage: f = E.scalar_field(x*y, name='f'); f
Scalar field f on the Euclidean plane E^2
sage: f.display()
f: E^2 → ℝ
   (x, y) ↦ x*y
   (r, ph) ↦ r^2*cos(ph)*sin(ph)
```

The value of `f` at point `p`:

```
sage: f(p)
-1
```

The gradient of `f`:

```
sage: from sage.manifolds.operators import * # to get grad, div, etc.
sage: w = grad(f); w
Vector field grad(f) on the Euclidean plane E^2
sage: w.display()
grad(f) = y e_x + x e_y
sage: w.display(polar_frame, polar)
grad(f) = 2*r*cos(ph)*sin(ph) e_r + (2*cos(ph)^2 - 1)*r e_ph
```

The dot product of two vector fields:

```
sage: s = v.dot(w); s
Scalar field v.grad(f) on the Euclidean plane E^2
sage: s.display()
v.grad(f): E^2 → ℝ
   (x, y) ↦ x^2 - y^2
   (r, ph) ↦ (2*cos(ph)^2 - 1)*r^2
```

(continues on next page)

(continued from previous page)

```
sage: s.expr()
x^2 - y^2
```

The norm is related to the dot product by the standard formula:

```
sage: norm(v)^2 == v.dot(v)
True
```

The divergence of the vector field v:

```
sage: s = div(v); s
Scalar field div(v) on the Euclidean plane E^2
sage: s.display()
div(v): E^2 → ℝ
   (x, y) ↦ 0
   (r, ph) ↦ 0
```

Example 2: Vector calculus in the Euclidean 3-space

We start by declaring the 3-dimensional Euclidean space E, with (x,y,z) as Cartesian coordinates:

```
sage: E.<x,y,z> = EuclideanSpace()
sage: E
Euclidean space E^3
```

A simple vector field on E:

```
sage: v = E.vector_field(-y, x, 0, name='v')
sage: v.display()
v = -y e_x + x e_y
sage: v[:]
[-y, x, 0]
```

The Euclidean norm of v:

```
sage: s = norm(v); s
Scalar field |v| on the Euclidean space E^3
sage: s.display()
|v|: E^3 → ℝ
   (x, y, z) ↦ sqrt(x^2 + y^2)
sage: s.expr()
sqrt(x^2 + y^2)
```

The divergence of v is zero:

```
sage: from sage.manifolds.operators import *
sage: div(v)
Scalar field div(v) on the Euclidean space E^3
sage: div(v).display()
div(v): E^3 → ℝ
   (x, y, z) ↦ 0
```

while its curl is a constant vector field along e_z:

```
sage: w = curl(v); w
Vector field curl(v) on the Euclidean space E^3
sage: w.display()
curl(v) = 2 e_z
```

The gradient of a scalar field:

```
sage: f = E.scalar_field(sin(x*y*z), name='f')
sage: u = grad(f); u
Vector field grad(f) on the Euclidean space E^3
sage: u.display()
grad(f) = y*z*cos(x*y*z) e_x + x*z*cos(x*y*z) e_y + x*y*cos(x*y*z) e_z
```

The curl of a gradient is zero:

```
sage: curl(u).display()
curl(grad(f)) = 0
```

The dot product of two vector fields:

```
sage: s = u.dot(v); s
Scalar field grad(f).v on the Euclidean space E^3
sage: s.expr()
(x^2 - y^2)*z*cos(x*y*z)
```

The cross product of two vector fields:

```
sage: a = u.cross(v); a
Vector field grad(f) x v on the Euclidean space E^3
sage: a.display()
grad(f) x v = -x^2*y*cos(x*y*z) e_x - x*y^2*cos(x*y*z) e_y
 + 2*x*y*z*cos(x*y*z) e_z
```

The scalar triple product of three vector fields:

```
sage: triple_product = E.scalar_triple_product()
sage: s = triple_product(u, v, w); s
Scalar field epsilon(grad(f),v,curl(v)) on the Euclidean space E^3
sage: s.expr()
4*x*y*z*cos(x*y*z)
```

Let us check that the scalar triple product of u, v and w is $u \cdot (v \times w)$:

```
sage: s == u.dot(v.cross(w))
True
```

AUTHORS:

- Eric Gourgoulhon (2018): initial version

REFERENCES:

- M. Berger: *Geometry I* [Ber1987]

class sage.manifolds.differentiable.examples.euclidean.**Euclidean3dimSpace**(*name=None*, *latex_name=None*, *coordinates='Cartesian'*, *symbols=None*, *metric_name='g'*, *metric_latex_name=None*, *start_index=1*, *base_manifold=None*, *category=None*, *unique_tag=None*)

Bases: *sage.manifolds.differentiable.examples.euclidean.EuclideanSpace*

3-dimensional Euclidean space.

A *3-dimensional Euclidean space* is an affine space E, whose associated vector space is a 3-dimensional vector space over $\mathbf{R}$ and is equipped with a positive definite symmetric bilinear form, called the *scalar product* or *dot product*.

The class *Euclidean3dimSpace* inherits from PseudoRiemannianManifold (via *EuclideanSpace*) since a 3-dimensional Euclidean space can be viewed as a Riemannian manifold that is diffeomorphic to $\mathbf{R}^3$ and that has a flat metric g. The Euclidean scalar product is the one defined by the Riemannian metric g.

INPUT:

- name – (default: None) string; name (symbol) given to the Euclidean 3-space; if None, the name will be set to 'E^3'
- latex_name – (default: None) string; LaTeX symbol to denote the Euclidean 3-space; if None, it is set to '\mathbb{E}^{3}' if name is None and to name otherwise
- coordinates – (default: 'Cartesian') string describing the type of coordinates to be initialized at the Euclidean 3-space creation; allowed values are 'Cartesian' (see *cartesian_coordinates()*), 'spherical' (see *spherical_coordinates()*) and 'cylindrical' (see *cylindrical_coordinates()*)
- symbols – (default: None) string defining the coordinate text symbols and LaTeX symbols, with the same conventions as the argument coordinates in RealDiffChart, namely symbols is a string of coordinate fields separated by a blank space, where each field contains the coordinate's text symbol and possibly the coordinate's LaTeX symbol (when the latter is different from the text symbol), both symbols being separated by a colon (:); if None, the symbols will be automatically generated according to the value of coordinates
- metric_name – (default: 'g') string; name (symbol) given to the Euclidean metric tensor
- metric_latex_name – (default: None) string; LaTeX symbol to denote the Euclidean metric tensor; if none is provided, it is set to metric_name
- start_index – (default: 1) integer; lower value of the range of indices used for "indexed objects" in the Euclidean 3-space, e.g. coordinates of a chart
- base_manifold – (default: None) if not None, must be an Euclidean 3-space; the created object is then an open subset of base_manifold
- category – (default: None) to specify the category; if None, Manifolds(RR).Smooth() & MetricSpaces().Complete() is assumed
- names – (default: None) unused argument, except if symbols is not provided; it must then be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator <,> is used)

- `init_coord_methods` – (default: `None`) dictionary of methods to initialize the various type of coordinates, with each key being a string describing the type of coordinates; to be used by derived classes only
- `unique_tag` – (default: `None`) tag used to force the construction of a new object when all the other arguments have been used previously (without `unique_tag`, the `UniqueRepresentation` behavior inherited from `PseudoRiemannianManifold` would return the previously constructed object corresponding to these arguments)

EXAMPLES:

A 3-dimensional Euclidean space:

```
sage: E = EuclideanSpace(3); E
Euclidean space E^3
sage: latex(E)
\mathbb{E}^{3}
```

`E` belongs to the class *Euclidean3dimSpace* (actually to a dynamically generated subclass of it via SageMath's category framework):

```
sage: type(E)
<class 'sage.manifolds.differentiable.examples.euclidean.Euclidean3dimSpace_with_
↪category'>
```

`E` is both a real smooth manifold of dimension 3 and a complete metric space:

```
sage: E.category()
Join of Category of smooth manifolds over Real Field with 53 bits of
 precision and Category of connected manifolds over Real Field with
 53 bits of precision and Category of complete metric spaces
sage: dim(E)
3
```

It is endowed with a default coordinate chart, which is that of Cartesian coordinates (x, y, z):

```
sage: E.atlas()
[Chart (E^3, (x, y, z))]
sage: E.default_chart()
Chart (E^3, (x, y, z))
sage: cartesian = E.cartesian_coordinates()
sage: cartesian is E.default_chart()
True
```

A point of `E`:

```
sage: p = E((3,-2,1)); p
Point on the Euclidean space E^3
sage: cartesian(p)
(3, -2, 1)
sage: p in E
True
sage: p.parent() is E
True
```

`E` is endowed with a default metric tensor, which defines the Euclidean scalar product:

```
sage: g = E.metric(); g
Riemannian metric g on the Euclidean space E^3
sage: g.display()
g = dx⊗dx + dy⊗dy + dz⊗dz
```

Curvilinear coordinates can be introduced on E: see *spherical_coordinates()* and *cylindrical_coordinates()*.

See also:

Example 2: Vector calculus in the Euclidean 3-space

cartesian_coordinates(*symbols=None*, *names=None*)

Return the chart of Cartesian coordinates, possibly creating it if it does not already exist.

INPUT:

- `symbols` – (default: `None`) string defining the coordinate text symbols and LaTeX symbols, with the same conventions as the argument `coordinates` in `RealDiffChart`; this is used only if the Cartesian chart has not been already defined; if `None` the symbols are generated as (x, y, z).
- `names` – (default: `None`) unused argument, except if `symbols` is not provided; it must be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator <,> is used)

OUTPUT:

- the chart of Cartesian coordinates, as an instance of `RealDiffChart`

EXAMPLES:

```
sage: E = EuclideanSpace(3)
sage: E.cartesian_coordinates()
Chart (E^3, (x, y, z))
sage: E.cartesian_coordinates().coord_range()
x: (-oo, +oo); y: (-oo, +oo); z: (-oo, +oo)
```

An example where the Cartesian coordinates have not been previously created:

```
sage: E = EuclideanSpace(3, coordinates='spherical')
sage: E.atlas()  # only spherical coordinates have been initialized
[Chart (E^3, (r, th, ph))]
sage: E.cartesian_coordinates(symbols='X Y Z')
Chart (E^3, (X, Y, Z))
sage: E.atlas()  # the Cartesian chart has been added to the atlas
[Chart (E^3, (r, th, ph)), Chart (E^3, (X, Y, Z))]
```

The coordinate variables are returned by the square bracket operator:

```
sage: E.cartesian_coordinates()[1]
X
sage: E.cartesian_coordinates()[3]
Z
sage: E.cartesian_coordinates()[:]
(X, Y, Z)
```

It is also possible to use the operator <,> to set symbolic variable containing the coordinates:

```
sage: E = EuclideanSpace(3, coordinates='spherical')
sage: cartesian.<u,v,w> = E.cartesian_coordinates()
sage: cartesian
Chart (E^3, (u, v, w))
sage: u, v, w
(u, v, w)
```

The command `cartesian.<u,v,w> = E.cartesian_coordinates()` is actually a shortcut for:

```
sage: cartesian = E.cartesian_coordinates(symbols='u v w')
sage: u, v, w = cartesian[:]
```

cylindrical_coordinates(*symbols=None*, *names=None*)

Return the chart of cylindrical coordinates, possibly creating it if it does not already exist.

INPUT:

- `symbols` – (default: `None`) string defining the coordinate text symbols and LaTeX symbols, with the same conventions as the argument `coordinates` in `RealDiffChart`; this is used only if the cylindrical chart has not been already defined; if `None` the symbols are generated as (ρ, ϕ, z).
- `names` – (default: `None`) unused argument, except if `symbols` is not provided; it must be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator `<,>` is used)

OUTPUT:

- the chart of cylindrical coordinates, as an instance of `RealDiffChart`

EXAMPLES:

```
sage: E = EuclideanSpace(3)
sage: E.cylindrical_coordinates()
Chart (E^3, (rh, ph, z))
sage: latex(_)
\left(\mathbb{E}^{3},({\rho}, {\phi}, z)\right)
sage: E.cylindrical_coordinates().coord_range()
rh: (0, +oo); ph: [0, 2*pi] (periodic); z: (-oo, +oo)
```

The relation to Cartesian coordinates is:

```
sage: E.coord_change(E.cylindrical_coordinates(),
....:                E.cartesian_coordinates()).display()
x = rh*cos(ph)
y = rh*sin(ph)
z = z
sage: E.coord_change(E.cartesian_coordinates(),
....:                E.cylindrical_coordinates()).display()
rh = sqrt(x^2 + y^2)
ph = arctan2(y, x)
z = z
```

The coordinate variables are returned by the square bracket operator:

```
sage: E.cylindrical_coordinates()[1]
rh
sage: E.cylindrical_coordinates()[3]
```

(continues on next page)

(continued from previous page)

```
z
sage: E.cylindrical_coordinates()[:]
(rh, ph, z)
```

They can also be obtained via the operator <,>:

```
sage: cylindrical.<rh,ph,z> = E.cylindrical_coordinates()
sage: cylindrical
Chart (E^3, (rh, ph, z))
sage: rh, ph, z
(rh, ph, z)
```

Actually, `cylindrical.<rh,ph,z> = E.cylindrical_coordinates()` is a shortcut for:

```
sage: cylindrical = E.cylindrical_coordinates()
sage: rh, ph, z = cylindrical[:]
```

The coordinate symbols can be customized:

```
sage: E = EuclideanSpace(3)
sage: E.cylindrical_coordinates(symbols=r"R Phi:\Phi Z")
Chart (E^3, (R, Phi, Z))
sage: latex(E.cylindrical_coordinates())
\left(\mathbb{E}^{3},(R, {\Phi}, Z)\right)
```

Note that if the cylindrical coordinates have been already initialized, the argument `symbols` has no effect:

```
sage: E.cylindrical_coordinates(symbols=r"rh:\rho ph:\phi z")
Chart (E^3, (R, Phi, Z))
```

cylindrical_frame()

Return the orthonormal vector frame associated with cylindrical coordinates.

OUTPUT:

- VectorFrame

EXAMPLES:

```
sage: E = EuclideanSpace(3)
sage: E.cylindrical_frame()
Vector frame (E^3, (e_rh,e_ph,e_z))
sage: E.cylindrical_frame()[1]
Vector field e_rh on the Euclidean space E^3
sage: E.cylindrical_frame()[:]
(Vector field e_rh on the Euclidean space E^3,
 Vector field e_ph on the Euclidean space E^3,
 Vector field e_z on the Euclidean space E^3)
```

The cylindrical frame expressed in terms of the Cartesian one:

```
sage: for e in E.cylindrical_frame():
....:     e.display(E.cartesian_frame(), E.cylindrical_coordinates())
e_rh = cos(ph) e_x + sin(ph) e_y
```

(continues on next page)

(continued from previous page)

```
e_ph = -sin(ph) e_x + cos(ph) e_y
e_z = e_z
```

The orthonormal frame (e_r, e_ϕ, e_z) expressed in terms of the coordinate frame $\left(\frac{\partial}{\partial r}, \frac{\partial}{\partial \phi}, \frac{\partial}{\partial z}\right)$:

```
sage: for e in E.cylindrical_frame():
....:     e.display(E.cylindrical_coordinates())
e_rh = ∂/∂rh
e_ph = 1/rh ∂/∂ph
e_z = ∂/∂z
```

scalar_triple_product(*name=None*, *latex_name=None*)

Return the scalar triple product operator, as a 3-form.

The *scalar triple product* (also called *mixed product*) of three vector fields u, v and w defined on an Euclidean space E is the scalar field

$$\epsilon(u, v, w) = u \cdot (v \times w).$$

The scalar triple product operator ϵ is a *3-form*, i.e. a field of fully antisymmetric trilinear forms; it is also called the *volume form* of E or the *Levi-Civita tensor* of E.

INPUT:

- `name` – (default: `None`) string; name given to the scalar triple product operator; if `None`, `'epsilon'` is used
- `latex_name` – (default: `None`) string; LaTeX symbol to denote the scalar triple product; if `None`, it is set to `r'\epsilon'` if `name` is `None` and to `name` otherwise.

OUTPUT:

- the scalar triple product operator ϵ, as an instance of `DiffFormParal`

EXAMPLES:

```
sage: E.<x,y,z> = EuclideanSpace()
sage: triple_product = E.scalar_triple_product()
sage: triple_product
3-form epsilon on the Euclidean space E^3
sage: latex(triple_product)
\epsilon
sage: u = E.vector_field(x, y, z, name='u')
sage: v = E.vector_field(-y, x, 0, name='v')
sage: w = E.vector_field(y*z, x*z, x*y, name='w')
sage: s = triple_product(u, v, w); s
Scalar field epsilon(u,v,w) on the Euclidean space E^3
sage: s.display()
epsilon(u,v,w): E^3 → ℝ
   (x, y, z) ↦ x^3*y + x*y^3 - 2*x*y*z^2
sage: s.expr()
x^3*y + x*y^3 - 2*x*y*z^2
sage: latex(s)
\epsilon\left(u,v,w\right)
sage: s == - triple_product(w, v, u)
True
```

Check of the identity $\epsilon(u,v,w) = u \cdot (v \times w)$:

```
sage: s == u.dot(v.cross(w))
True
```

Customizing the name:

```
sage: E.scalar_triple_product(name='S')
3-form S on the Euclidean space E^3
sage: latex(_)
S
sage: E.scalar_triple_product(name='Omega', latex_name=r'\Omega')
3-form Omega on the Euclidean space E^3
sage: latex(_)
\Omega
```

spherical_coordinates(*symbols=None*, *names=None*)

Return the chart of spherical coordinates, possibly creating it if it does not already exist.

INPUT:

- `symbols` – (default: `None`) string defining the coordinate text symbols and LaTeX symbols, with the same conventions as the argument `coordinates` in `RealDiffChart`; this is used only if the spherical chart has not been already defined; if `None` the symbols are generated as (r, θ, ϕ).
- `names` – (default: `None`) unused argument, except if `symbols` is not provided; it must be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator `<,>` is used)

OUTPUT:

- the chart of spherical coordinates, as an instance of `RealDiffChart`

EXAMPLES:

```
sage: E = EuclideanSpace(3)
sage: E.spherical_coordinates()
Chart (E^3, (r, th, ph))
sage: latex(_)
\left(\mathbb{E}^{3},(r, {\theta}, {\phi})\right)
sage: E.spherical_coordinates().coord_range()
r: (0, +oo); th: (0, pi); ph: [0, 2*pi] (periodic)
```

The relation to Cartesian coordinates is:

```
sage: E.coord_change(E.spherical_coordinates(),
....:                E.cartesian_coordinates()).display()
x = r*cos(ph)*sin(th)
y = r*sin(ph)*sin(th)
z = r*cos(th)
sage: E.coord_change(E.cartesian_coordinates(),
....:                E.spherical_coordinates()).display()
r = sqrt(x^2 + y^2 + z^2)
th = arctan2(sqrt(x^2 + y^2), z)
ph = arctan2(y, x)
```

The coordinate variables are returned by the square bracket operator:

```
sage: E.spherical_coordinates()[1]
r
sage: E.spherical_coordinates()[3]
ph
sage: E.spherical_coordinates()[:]
(r, th, ph)
```

They can also be obtained via the operator <,>:

```
sage: spherical.<r,th,ph> = E.spherical_coordinates()
sage: spherical
Chart (E^3, (r, th, ph))
sage: r, th, ph
(r, th, ph)
```

Actually, `spherical.<r,th,ph> = E.spherical_coordinates()` is a shortcut for:

```
sage: spherical = E.spherical_coordinates()
sage: r, th, ph = spherical[:]
```

The coordinate symbols can be customized:

```
sage: E = EuclideanSpace(3)
sage: E.spherical_coordinates(symbols=r"R T:\Theta F:\Phi")
Chart (E^3, (R, T, F))
sage: latex(E.spherical_coordinates())
\left(\mathbb{E}^{3},(R, {\Theta}, {\Phi})\right)
```

Note that if the spherical coordinates have been already initialized, the argument `symbols` has no effect:

```
sage: E.spherical_coordinates(symbols=r"r th:\theta ph:\phi")
Chart (E^3, (R, T, F))
```

spherical_frame()

Return the orthonormal vector frame associated with spherical coordinates.

OUTPUT:

- VectorFrame

EXAMPLES:

```
sage: E = EuclideanSpace(3)
sage: E.spherical_frame()
Vector frame (E^3, (e_r,e_th,e_ph))
sage: E.spherical_frame()[1]
Vector field e_r on the Euclidean space E^3
sage: E.spherical_frame()[:]
(Vector field e_r on the Euclidean space E^3,
 Vector field e_th on the Euclidean space E^3,
 Vector field e_ph on the Euclidean space E^3)
```

The spherical frame expressed in terms of the Cartesian one:

```
sage: for e in E.spherical_frame():
....:     e.display(E.cartesian_frame(), E.spherical_coordinates())
e_r = cos(ph)*sin(th) e_x + sin(ph)*sin(th) e_y + cos(th) e_z
e_th = cos(ph)*cos(th) e_x + cos(th)*sin(ph) e_y - sin(th) e_z
e_ph = -sin(ph) e_x + cos(ph) e_y
```

The orthonormal frame (e_r, e_θ, e_ϕ) expressed in terms of the coordinate frame $\left(\frac{\partial}{\partial r}, \frac{\partial}{\partial \theta}, \frac{\partial}{\partial \phi}\right)$:

```
sage: for e in E.spherical_frame():
....:     e.display(E.spherical_coordinates())
e_r = ∂/∂r
e_th = 1/r ∂/∂th
e_ph = 1/(r*sin(th)) ∂/∂ph
```

class sage.manifolds.differentiable.examples.euclidean.**EuclideanPlane**(*name=None*, *latex_name=None*, *coordinates='Cartesian'*, *symbols=None*, *metric_name='g'*, *metric_latex_name=None*, *start_index=1*, *base_manifold=None*, *category=None*, *unique_tag=None*)

Bases: *sage.manifolds.differentiable.examples.euclidean.EuclideanSpace*

Euclidean plane.

An *Euclidean plane* is an affine space E, whose associated vector space is a 2-dimensional vector space over $\mathbf{R}$ and is equipped with a positive definite symmetric bilinear form, called the *scalar product* or *dot product*.

The class *EuclideanPlane* inherits from PseudoRiemannianManifold (via *EuclideanSpace*) since an Euclidean plane can be viewed as a Riemannian manifold that is diffeomorphic to $\mathbf{R}^2$ and that has a flat metric g. The Euclidean scalar product is the one defined by the Riemannian metric g.

INPUT:

- name – (default: None) string; name (symbol) given to the Euclidean plane; if None, the name will be set to 'E^2'
- latex_name – (default: None) string; LaTeX symbol to denote the Euclidean plane; if None, it is set to '\mathbb{E}^{2}' if name is None and to name otherwise
- coordinates – (default: 'Cartesian') string describing the type of coordinates to be initialized at the Euclidean plane creation; allowed values are 'Cartesian' (see *cartesian_coordinates()*) and 'polar' (see *polar_coordinates()*)
- symbols – (default: None) string defining the coordinate text symbols and LaTeX symbols, with the same conventions as the argument coordinates in RealDiffChart, namely symbols is a string of coordinate fields separated by a blank space, where each field contains the coordinate's text symbol and possibly the coordinate's LaTeX symbol (when the latter is different from the text symbol), both symbols being separated by a colon (:); if None, the symbols will be automatically generated according to the value of coordinates
- metric_name – (default: 'g') string; name (symbol) given to the Euclidean metric tensor
- metric_latex_name – (default: None) string; LaTeX symbol to denote the Euclidean metric tensor; if none is provided, it is set to metric_name

- `start_index` – (default: 1) integer; lower value of the range of indices used for "indexed objects" in the Euclidean plane, e.g. coordinates of a chart
- `base_manifold` – (default: `None`) if not `None`, must be an Euclidean plane; the created object is then an open subset of `base_manifold`
- `category` – (default: `None`) to specify the category; if `None`, `Manifolds(RR).Smooth() & MetricSpaces().Complete()` is assumed
- `names` – (default: `None`) unused argument, except if `symbols` is not provided; it must then be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator `<,>` is used)
- `init_coord_methods` – (default: `None`) dictionary of methods to initialize the various type of coordinates, with each key being a string describing the type of coordinates; to be used by derived classes only
- `unique_tag` – (default: `None`) tag used to force the construction of a new object when all the other arguments have been used previously (without `unique_tag`, the `UniqueRepresentation` behavior inherited from `PseudoRiemannianManifold` would return the previously constructed object corresponding to these arguments)

EXAMPLES:

One creates an Euclidean plane `E` with:

```
sage: E.<x,y> = EuclideanSpace(); E
Euclidean plane E^2
```

`E` is both a real smooth manifold of dimension 2 and a complete metric space:

```
sage: E.category()
Join of Category of smooth manifolds over Real Field with 53 bits of
 precision and Category of connected manifolds over Real Field with
 53 bits of precision and Category of complete metric spaces
sage: dim(E)
2
```

It is endowed with a default coordinate chart, which is that of Cartesian coordinates (x, y):

```
sage: E.atlas()
[Chart (E^2, (x, y))]
sage: E.default_chart()
Chart (E^2, (x, y))
sage: cartesian = E.cartesian_coordinates()
sage: cartesian is E.default_chart()
True
```

A point of `E`:

```
sage: p = E((3,-2)); p
Point on the Euclidean plane E^2
sage: cartesian(p)
(3, -2)
sage: p in E
True
sage: p.parent() is E
True
```

`E` is endowed with a default metric tensor, which defines the Euclidean scalar product:

```
sage: g = E.metric(); g
Riemannian metric g on the Euclidean plane E^2
sage: g.display()
g = dx⊗dx + dy⊗dy
```

Curvilinear coordinates can be introduced on `E`: see *polar_coordinates()*.

See also:

Example 1: the Euclidean plane

cartesian_coordinates(*symbols=None*, *names=None*)

Return the chart of Cartesian coordinates, possibly creating it if it does not already exist.

INPUT:

- `symbols` – (default: `None`) string defining the coordinate text symbols and LaTeX symbols, with the same conventions as the argument `coordinates` in `RealDiffChart`; this is used only if the Cartesian chart has not been already defined; if `None` the symbols are generated as (x, y).
- `names` – (default: `None`) unused argument, except if `symbols` is not provided; it must be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator `<,>` is used)

OUTPUT:

- the chart of Cartesian coordinates, as an instance of `RealDiffChart`

EXAMPLES:

```
sage: E = EuclideanSpace(2)
sage: E.cartesian_coordinates()
Chart (E^2, (x, y))
sage: E.cartesian_coordinates().coord_range()
x: (-oo, +oo); y: (-oo, +oo)
```

An example where the Cartesian coordinates have not been previously created:

```
sage: E = EuclideanSpace(2, coordinates='polar')
sage: E.atlas()  # only polar coordinates have been initialized
[Chart (E^2, (r, ph))]
sage: E.cartesian_coordinates(symbols='X Y')
Chart (E^2, (X, Y))
sage: E.atlas()  # the Cartesian chart has been added to the atlas
[Chart (E^2, (r, ph)), Chart (E^2, (X, Y))]
```

Note that if the Cartesian coordinates have been already initialized, the argument `symbols` has no effect:

```
sage: E.cartesian_coordinates(symbols='x y')
Chart (E^2, (X, Y))
```

The coordinate variables are returned by the square bracket operator:

```
sage: E.cartesian_coordinates()[1]
X
sage: E.cartesian_coordinates()[2]
Y
sage: E.cartesian_coordinates()[:]
(X, Y)
```

It is also possible to use the operator <,> to set symbolic variable containing the coordinates:

```
sage: E = EuclideanSpace(2, coordinates='polar')
sage: cartesian.<u,v> = E.cartesian_coordinates()
sage: cartesian
Chart (E^2, (u, v))
sage: u,v
(u, v)
```

The command `cartesian.<u,v> = E.cartesian_coordinates()` is actually a shortcut for:

```
sage: cartesian = E.cartesian_coordinates(symbols='u v')
sage: u, v = cartesian[:]
```

polar_coordinates(*symbols=None*, *names=None*)

Return the chart of polar coordinates, possibly creating it if it does not already exist.

INPUT:

- `symbols` – (default: `None`) string defining the coordinate text symbols and LaTeX symbols, with the same conventions as the argument `coordinates` in `RealDiffChart`; this is used only if the polar chart has not been already defined; if `None` the symbols are generated as (r, ϕ).
- `names` – (default: `None`) unused argument, except if `symbols` is not provided; it must be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator <,> is used)

OUTPUT:

- the chart of polar coordinates, as an instance of `RealDiffChart`

EXAMPLES:

```
sage: E = EuclideanSpace(2)
sage: E.polar_coordinates()
Chart (E^2, (r, ph))
sage: latex(_)
\left(\mathbb{E}^{2},(r, {\phi})\right)
sage: E.polar_coordinates().coord_range()
r: (0, +oo); ph: [0, 2*pi] (periodic)
```

The relation to Cartesian coordinates is:

```
sage: E.coord_change(E.polar_coordinates(),
....:                E.cartesian_coordinates()).display()
x = r*cos(ph)
y = r*sin(ph)
sage: E.coord_change(E.cartesian_coordinates(),
....:                E.polar_coordinates()).display()
r = sqrt(x^2 + y^2)
ph = arctan2(y, x)
```

The coordinate variables are returned by the square bracket operator:

```
sage: E.polar_coordinates()[1]
r
sage: E.polar_coordinates()[2]
ph
```

(continues on next page)

(continued from previous page)

```
sage: E.polar_coordinates()[:]
(r, ph)
```

They can also be obtained via the operator <,>:

```
sage: polar.<r,ph> = E.polar_coordinates(); polar
Chart (E^2, (r, ph))
sage: r, ph
(r, ph)
```

Actually, `polar.<r,ph> = E.polar_coordinates()` is a shortcut for:

```
sage: polar = E.polar_coordinates()
sage: r, ph = polar[:]
```

The coordinate symbols can be customized:

```
sage: E = EuclideanSpace(2)
sage: E.polar_coordinates(symbols=r"r th:\theta")
Chart (E^2, (r, th))
sage: latex(E.polar_coordinates())
\left(\mathbb{E}^{2},(r, {\theta})\right)
```

Note that if the polar coordinates have been already initialized, the argument `symbols` has no effect:

```
sage: E.polar_coordinates(symbols=r"R Th:\Theta")
Chart (E^2, (r, th))
```

polar_frame()

Return the orthonormal vector frame associated with polar coordinates.

OUTPUT:

- instance of `VectorFrame`

EXAMPLES:

```
sage: E = EuclideanSpace(2)
sage: E.polar_frame()
Vector frame (E^2, (e_r,e_ph))
sage: E.polar_frame()[1]
Vector field e_r on the Euclidean plane E^2
sage: E.polar_frame()[:]
(Vector field e_r on the Euclidean plane E^2,
 Vector field e_ph on the Euclidean plane E^2)
```

The orthonormal polar frame expressed in terms of the Cartesian one:

```
sage: for e in E.polar_frame():
....:     e.display(E.cartesian_frame(), E.polar_coordinates())
e_r = cos(ph) e_x + sin(ph) e_y
e_ph = -sin(ph) e_x + cos(ph) e_y
```

The orthonormal frame (e_r, e_ϕ) expressed in terms of the coordinate frame $\left(\frac{\partial}{\partial r}, \frac{\partial}{\partial \phi}\right)$:

```
sage: for e in E.polar_frame():
....:     e.display(E.polar_coordinates())
e_r = ∂/∂r
e_ph = 1/r ∂/∂ph
```

class sage.manifolds.differentiable.examples.euclidean.**EuclideanSpace**(*n*, *name=None*, *latex_name=None*, *coordinates='Cartesian'*, *symbols=None*, *metric_name='g'*, *metric_latex_name=None*, *start_index=1*, *base_manifold=None*, *category=None*, *init_coord_methods=None*, *unique_tag=None*)

Bases: sage.manifolds.differentiable.pseudo_riemannian.PseudoRiemannianManifold

Euclidean space.

An *Euclidean space of dimension* n is an affine space E, whose associated vector space is a n-dimensional vector space over $\mathbf{R}$ and is equipped with a positive definite symmetric bilinear form, called the *scalar product* or *dot product*.

Euclidean space of dimension n can be viewed as a Riemannian manifold that is diffeomorphic to $\mathbf{R}^n$ and that has a flat metric g. The Euclidean scalar product is the one defined by the Riemannian metric g.

INPUT:

- `n` – positive integer; dimension of the space over the real field
- `name` – (default: `None`) string; name (symbol) given to the Euclidean space; if `None`, the name will be set to `'E^n'`
- `latex_name` – (default: `None`) string; LaTeX symbol to denote the space; if `None`, it is set to `'\mathbb{E}^{n}'` if `name` is `None` and to `name` otherwise
- `coordinates` – (default: `'Cartesian'`) string describing the type of coordinates to be initialized at the Euclidean space creation; allowed values are
  - `'Cartesian'` (canonical coordinates on $\mathbf{R}^n$)
  - `'polar'` for `n=2` only (see *polar_coordinates()*)
  - `'spherical'` for `n=3` only (see *spherical_coordinates()*)
  - `'cylindrical'` for `n=3` only (see *cylindrical_coordinates()*)
- `symbols` – (default: `None`) string defining the coordinate text symbols and LaTeX symbols, with the same conventions as the argument `coordinates` in RealDiffChart, namely `symbols` is a string of coordinate fields separated by a blank space, where each field contains the coordinate's text symbol and possibly the coordinate's LaTeX symbol (when the latter is different from the text symbol), both symbols being separated by a colon (`:`); if `None`, the symbols will be automatically generated according to the value of `coordinates`
- `metric_name` – (default: `'g'`) string; name (symbol) given to the Euclidean metric tensor
- `metric_latex_name` – (default: `None`) string; LaTeX symbol to denote the Euclidean metric tensor; if none is provided, it is set to `metric_name`

- `start_index` – (default: 1) integer; lower value of the range of indices used for "indexed objects" in the Euclidean space, e.g. coordinates of a chart
- `names` – (default: `None`) unused argument, except if `symbols` is not provided; it must then be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator `<,>` is used)

If `names` is specified, then `n` does not have to be specified.

EXAMPLES:

Constructing a 2-dimensional Euclidean space:

```
sage: E = EuclideanSpace(2); E
Euclidean plane E^2
```

Each call to *EuclideanSpace* creates a different object:

```
sage: E1 = EuclideanSpace(2)
sage: E1 is E
False
sage: E1 == E
False
```

The LaTeX symbol of the Euclidean space is by default $\mathbb{E}^n$, where n is the dimension:

```
sage: latex(E)
\mathbb{E}^{2}
```

But both the name and LaTeX names of the Euclidean space can be customized:

```
sage: F = EuclideanSpace(2, name='F', latex_name=r'\mathcal{F}'); F
Euclidean plane F
sage: latex(F)
\mathcal{F}
```

By default, an Euclidean space is created with a single coordinate chart: that of Cartesian coordinates:

```
sage: E.atlas()
[Chart (E^2, (x, y))]
sage: E.cartesian_coordinates()
Chart (E^2, (x, y))
sage: E.default_chart() is E.cartesian_coordinates()
True
```

The coordinate variables can be initialized, as the Python variables `x` and `y`, by:

```
sage: x, y = E.cartesian_coordinates()[:]
```

However, it is possible to both construct the Euclidean space and initialize the coordinate variables in a single stage, thanks to SageMath operator `<,>`:

```
sage: E.<x,y> = EuclideanSpace()
```

Note that providing the dimension as an argument of `EuclideanSpace` is not necessary in that case, since it can be deduced from the number of coordinates within `<,>`. Besides, the coordinate symbols can be customized:

```
sage: E.<X,Y> = EuclideanSpace()
sage: E.cartesian_coordinates()
Chart (E^2, (X, Y))
```

By default, the LaTeX symbols of the coordinates coincide with the text ones:

```
sage: latex(X+Y)
X + Y
```

However, it is possible to customize them, via the argument `symbols`, which must be a string, usually prefixed by `r` (for *raw* string, in order to allow for the backslash character of LaTeX expressions). This string contains the coordinate fields separated by a blank space; each field contains the coordinate's text symbol and possibly the coordinate's LaTeX symbol (when the latter is different from the text symbol), both symbols being separated by a colon (`:`):

```
sage: E.<xi,ze> = EuclideanSpace(symbols=r"xi:\xi ze:\zeta")
sage: E.cartesian_coordinates()
Chart (E^2, (xi, ze))
sage: latex(xi+ze)
{\xi} + {\zeta}
```

Thanks to the argument `coordinates`, an Euclidean space can be constructed with curvilinear coordinates initialized instead of the Cartesian ones:

```
sage: E.<r,ph> = EuclideanSpace(coordinates='polar')
sage: E.atlas()   # no Cartesian coordinates have been constructed
[Chart (E^2, (r, ph))]
sage: polar = E.polar_coordinates(); polar
Chart (E^2, (r, ph))
sage: E.default_chart() is polar
True
sage: latex(r+ph)
{\phi} + r
```

The Cartesian coordinates, along with the transition maps to and from the curvilinear coordinates, can be constructed at any time by:

```
sage: cartesian.<x,y> = E.cartesian_coordinates()
sage: E.atlas()  # both polar and Cartesian coordinates now exist
[Chart (E^2, (r, ph)), Chart (E^2, (x, y))]
```

The transition maps have been initialized by the command `E.cartesian_coordinates()`:

```
sage: E.coord_change(polar, cartesian).display()
x = r*cos(ph)
y = r*sin(ph)
sage: E.coord_change(cartesian, polar).display()
r = sqrt(x^2 + y^2)
ph = arctan2(y, x)
```

The default name of the Euclidean metric tensor is g:

```
sage: E.metric()
Riemannian metric g on the Euclidean plane E^2
```

(continues on next page)

(continued from previous page)

```
sage: latex(_)
g
```

But this can be customized:

```
sage: E = EuclideanSpace(2, metric_name='h')
sage: E.metric()
Riemannian metric h on the Euclidean plane E^2
sage: latex(_)
h
sage: E = EuclideanSpace(2, metric_latex_name=r'\mathbf{g}')
sage: E.metric()
Riemannian metric g on the Euclidean plane E^2
sage: latex(_)
\mathbf{g}
```

A 4-dimensional Euclidean space:

```
sage: E = EuclideanSpace(4); E
4-dimensional Euclidean space E^4
sage: latex(E)
\mathbb{E}^{4}
```

E is both a real smooth manifold of dimension 4 and a complete metric space:

```
sage: E.category()
Join of Category of smooth manifolds over Real Field with 53 bits of
 precision and Category of connected manifolds over Real Field with
 53 bits of precision and Category of complete metric spaces
sage: dim(E)
4
```

It is endowed with a default coordinate chart, which is that of Cartesian coordinates (x_1, x_2, x_3, x_4):

```
sage: E.atlas()
[Chart (E^4, (x1, x2, x3, x4))]
sage: E.default_chart()
Chart (E^4, (x1, x2, x3, x4))
sage: E.default_chart() is E.cartesian_coordinates()
True
```

E is also endowed with a default metric tensor, which defines the Euclidean scalar product:

```
sage: g = E.metric(); g
Riemannian metric g on the 4-dimensional Euclidean space E^4
sage: g.display()
g = dx1⊗dx1 + dx2⊗dx2 + dx3⊗dx3 + dx4⊗dx4
```

cartesian_coordinates(*symbols=None*, *names=None*)

Return the chart of Cartesian coordinates, possibly creating it if it does not already exist.

INPUT:

- `symbols` – (default: `None`) string defining the coordinate text symbols and LaTeX symbols, with the same conventions as the argument `coordinates` in `RealDiffChart`; this is used only if the Cartesian

chart has not been already defined; if `None` the symbols are generated as $(x_1, \ldots, x_n)$.

- `names` – (default: `None`) unused argument, except if `symbols` is not provided; it must be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator `<,>` is used)

OUTPUT:

- the chart of Cartesian coordinates, as an instance of `RealDiffChart`

EXAMPLES:

```
sage: E = EuclideanSpace(4)
sage: X = E.cartesian_coordinates(); X
Chart (E^4, (x1, x2, x3, x4))
sage: X.coord_range()
x1: (-oo, +oo); x2: (-oo, +oo); x3: (-oo, +oo); x4: (-oo, +oo)
sage: X[2]
x2
sage: X[:]
(x1, x2, x3, x4)
sage: latex(X[:])
\left({x_{1}}, {x_{2}}, {x_{3}}, {x_{4}}\right)
```

cartesian_frame()

Return the orthonormal vector frame associated with Cartesian coordinates.

OUTPUT:

- `CoordFrame`

EXAMPLES:

```
sage: E = EuclideanSpace(2)
sage: E.cartesian_frame()
Coordinate frame (E^2, (e_x,e_y))
sage: E.cartesian_frame()[1]
Vector field e_x on the Euclidean plane E^2
sage: E.cartesian_frame()[:]
(Vector field e_x on the Euclidean plane E^2,
 Vector field e_y on the Euclidean plane E^2)
```

For Cartesian coordinates, the orthonormal frame coincides with the coordinate frame:

```
sage: E.cartesian_frame() is E.cartesian_coordinates().frame()
True
```

dist(p, q)

Euclidean distance between two points.

INPUT:

- `p` – an element of `self`
- `q` – an element of `self`

OUTPUT:

- the Euclidean distance $d(p, q)$

EXAMPLES:

```
sage: E.<x,y> = EuclideanSpace()
sage: p = E((1,0))
sage: q = E((0,2))
sage: E.dist(p, q)
sqrt(5)
sage: p.dist(q)  # indirect doctest
sqrt(5)
```

sphere(*radius=1*, *center=None*, *name=None*, *latex_name=None*, *coordinates='spherical'*, *names=None*)
Return an $(n-1)$-sphere smoothly embedded in `self`.

INPUT:

- `radius` – (default: 1) the radius greater than 1 of the sphere
- `center` – (default: `None`) point on `self` representing the barycenter of the sphere
- `name` – (default: `None`) string; name (symbol) given to the sphere; if `None`, the name will be generated according to the input
- `latex_name` – (default: `None`) string; LaTeX symbol to denote the sphere; if `None`, the symbol will be generated according to the input
- `coordinates` – (default: `'spherical'`) string describing the type of coordinates to be initialized at the sphere's creation; allowed values are
  - `'spherical'` spherical coordinates (see `spherical_coordinates()`)
  - `'stereographic'` stereographic coordinates given by the stereographic projection (see `stereographic_coordinates()`)
- `names` – (default: `None`) must be a tuple containing the coordinate symbols (this guarantees the shortcut operator <,> to function); if `None`, the usual conventions are used (see examples in `Sphere` for details)

EXAMPLES:

Define a 2-sphere with radius 2 centered at $(1, 2, 3)$ in Cartesian coordinates:

```
sage: E3 = EuclideanSpace(3)
sage: c = E3.point((1,2,3), name='c'); c
Point c on the Euclidean space E^3
sage: S2_2 = E3.sphere(radius=2, center=c); S2_2
2-sphere S^2_2(c) of radius 2 smoothly embedded in the Euclidean
 space E^3 centered at the Point c
```

The ambient space is precisely our previously defined Euclidean space:

```
sage: S2_2.ambient() is E3
True
```

The embedding into Euclidean space:

```
sage: S2_2.embedding().display()
iota: S^2_2(c) → E^3
on A: (theta, phi) ↦ (x, y, z) = (2*cos(phi)*sin(theta) + 1,
                                  2*sin(phi)*sin(theta) + 2,
                                  2*cos(theta) + 3)
```

See `Sphere` for more examples.

CHAPTER

TWO

OPERATORS FOR VECTOR CALCULUS

This module defines the following operators for scalar, vector and tensor fields on any pseudo-Riemannian manifold (see pseudo_riemannian), and in particular on Euclidean spaces (see *euclidean*):

- *grad()*: gradient of a scalar field
- *div()*: divergence of a vector field, and more generally of a tensor field
- *curl()*: curl of a vector field (3-dimensional case only)
- *laplacian()*: Laplace-Beltrami operator acting on a scalar field, a vector field, or more generally a tensor field
- *dalembertian()*: d'Alembert operator acting on a scalar field, a vector field, or more generally a tensor field, on a Lorentzian manifold

All these operators are implemented as functions that call the appropriate method on their argument. The purpose is to allow one to use standard mathematical notations, e.g. to write `curl(v)` instead of `v.curl()`.

Note that the norm() operator is defined in the module functional.

See also:

Examples 1 and 2 in *euclidean* for examples involving these operators in the Euclidean plane and in the Euclidean 3-space.

AUTHORS:

- Eric Gourgoulhon (2018): initial version

sage.manifolds.operators.**curl**(*vector*)
Curl operator.

The *curl* of a vector field v on an orientable pseudo-Riemannian manifold (M, g) of dimension 3 is the vector field defined by

$$\mathrm{curl}\, v = (*(\mathrm{d}v^\flat))^\sharp$$

where $v^\flat$ is the 1-form associated to v by the metric g (see down()), $*(\mathrm{d}v^\flat)$ is the Hodge dual with respect to g of the 2-form $\mathrm{d}v^\flat$ (exterior derivative of $v^\flat$) (see hodge_dual()) and $(*(\mathrm{d}v^\flat))^\sharp$ is corresponding vector field by g-duality (see up()).

An alternative expression of the curl is

$$(\mathrm{curl}\, v)^i = \epsilon^{ijk} \nabla_j v_k$$

where ∇ is the Levi-Civita connection of g (cf. LeviCivitaConnection) and ϵ the volume 3-form (Levi-Civita tensor) of g (cf. volume_form())

INPUT:

- `vector` – vector field on an orientable 3-dimensional pseudo-Riemannian manifold, as an instance of `VectorField`

OUTPUT:

- instance of `VectorField` representing the curl of `vector`

EXAMPLES:

Curl of a vector field in the Euclidean 3-space:

```
sage: E.<x,y,z> = EuclideanSpace()
sage: v = E.vector_field(sin(y), sin(x), 0, name='v')
sage: v.display()
v = sin(y) e_x + sin(x) e_y
sage: from sage.manifolds.operators import curl
sage: s = curl(v); s
Vector field curl(v) on the Euclidean space E^3
sage: s.display()
curl(v) = (cos(x) - cos(y)) e_z
sage: s[:]
[0, 0, cos(x) - cos(y)]
```

See the method `curl()` of `VectorField` for more details and examples.

sage.manifolds.operators.**dalembertian**(*field*)

d'Alembert operator.

The *d'Alembert operator* or *d'Alembertian* on a Lorentzian manifold (M, g) is nothing but the Laplace-Beltrami operator:

$$\Box = \nabla_i \nabla^i = g^{ij} \nabla_i \nabla_j$$

where ∇ is the Levi-Civita connection of the metric g (cf. `LeviCivitaConnection`) and $\nabla^i := g^{ij}\nabla_j$

INPUT:

- `field` – a scalar field f (instance of `DiffScalarField`) or a tensor field f (instance of `TensorField`) on a pseudo-Riemannian manifold

OUTPUT:

- $\Box f$, as an instance of `DiffScalarField` or of `TensorField`

EXAMPLES:

d'Alembertian of a scalar field in the 2-dimensional Minkowski spacetime:

```
sage: M = Manifold(2, 'M', structure='Lorentzian')
sage: X.<t,x> = M.chart()
sage: g = M.metric()
sage: g[0,0], g[1,1] = -1, 1
sage: f = M.scalar_field((x-t)^3 + (x+t)^2, name='f')
sage: from sage.manifolds.operators import dalembertian
sage: Df = dalembertian(f); Df
Scalar field Box(f) on the 2-dimensional Lorentzian manifold M
sage: Df.display()
Box(f): M → ℝ
   (t, x) ↦ 0
```

See the method dalembertian() of DiffScalarField and the method dalembertian() of TensorField for more details and examples.

sage.manifolds.operators.**div**(*tensor*)

Divergence operator.

Let t be a tensor field of type $(k, 0)$ with $k \geq 1$ on a pseudo-Riemannian manifold (M, g). The *divergence* of t is the tensor field of type $(k-1, 0)$ defined by

$$(\mathrm{div}\, t)^{a_1 \ldots a_{k-1}} = \nabla_i t^{a_1 \ldots a_{k-1} i} = (\nabla t)^{a_1 \ldots a_{k-1} i}{}_i$$

where ∇ is the Levi-Civita connection of g (cf. LeviCivitaConnection).

Note that the divergence is taken on the *last* index of the tensor field. This definition is extended to tensor fields of type (k, l) with $k \geq 0$ and $l \geq 1$, by raising the last index with the metric g: $\mathrm{div}\, t$ is then the tensor field of type $(k, l-1)$ defined by

$$(\mathrm{div}\, t)^{a_1 \ldots a_k}{}_{b_1 \ldots b_{l-1}} = \nabla_i (g^{ij} t^{a_1 \ldots a_k}{}_{b_1 \ldots b_{l-1} j}) = (\nabla t^\sharp)^{a_1 \ldots a_k i}{}_{b_1 \ldots b_{l-1} i}$$

where $t^\sharp$ is the tensor field deduced from t by raising the last index with the metric g (see up()).

INPUT:

- tensor – tensor field t on a pseudo-Riemannian manifold (M, g), as an instance of TensorField (possibly via one of its derived classes, like VectorField)

OUTPUT:

- the divergence of tensor as an instance of either DiffScalarField if $(k, l) = (1, 0)$ (tensor is a vector field) or $(k, l) = (0, 1)$ (tensor is a 1-form) or of TensorField if $k + l \geq 2$

EXAMPLES:

Divergence of a vector field in the Euclidean plane:

```
sage: E.<x,y> = EuclideanSpace()
sage: v = E.vector_field(cos(x*y), sin(x*y), name='v')
sage: v.display()
v = cos(x*y) e_x + sin(x*y) e_y
sage: from sage.manifolds.operators import div
sage: s = div(v); s
Scalar field div(v) on the Euclidean plane E^2
sage: s.display()
div(v): E^2 → ℝ
   (x, y) ↦ x*cos(x*y) - y*sin(x*y)
sage: s.expr()
x*cos(x*y) - y*sin(x*y)
```

See the method divergence() of TensorField for more details and examples.

sage.manifolds.operators.**grad**(*scalar*)

Gradient operator.

The *gradient* of a scalar field f on a pseudo-Riemannian manifold (M, g) is the vector field $\mathrm{grad}\, f$ whose components in any coordinate frame are

$$(\mathrm{grad}\, f)^i = g^{ij} \frac{\partial F}{\partial x^j}$$

where the x^j's are the coordinates with respect to which the frame is defined and F is the chart function representing f in these coordinates: $f(p) = F(x^1(p), \ldots, x^n(p))$ for any point p in the chart domain. In other words, the gradient of f is the vector field that is the g-dual of the differential of f.

INPUT:

- `scalar` – scalar field f, as an instance of `DiffScalarField`

OUTPUT:

- instance of `VectorField` representing $\mathrm{grad}\, f$

EXAMPLES:

Gradient of a scalar field in the Euclidean plane:

```
sage: E.<x,y> = EuclideanSpace()
sage: f = E.scalar_field(sin(x*y), name='f')
sage: from sage.manifolds.operators import grad
sage: grad(f)
Vector field grad(f) on the Euclidean plane E^2
sage: grad(f).display()
grad(f) = y*cos(x*y) e_x + x*cos(x*y) e_y
sage: grad(f)[:]
[y*cos(x*y), x*cos(x*y)]
```

See the method `gradient()` of `DiffScalarField` for more details and examples.

`sage.manifolds.operators.laplacian`(*field*)
Laplace-Beltrami operator.

The *Laplace-Beltrami operator* on a pseudo-Riemannian manifold (M, g) is the operator

$$\Delta = \nabla_i \nabla^i = g^{ij} \nabla_i \nabla_j$$

where ∇ is the Levi-Civita connection of the metric g (cf. `LeviCivitaConnection`) and $\nabla^i := g^{ij}\nabla_j$

INPUT:

- `field` – a scalar field f (instance of `DiffScalarField`) or a tensor field f (instance of `TensorField`) on a pseudo-Riemannian manifold

OUTPUT:

- Δf, as an instance of `DiffScalarField` or of `TensorField`

EXAMPLES:

Laplacian of a scalar field on the Euclidean plane:

```
sage: E.<x,y> = EuclideanSpace()
sage: f = E.scalar_field(sin(x*y), name='f')
sage: from sage.manifolds.operators import laplacian
sage: Df = laplacian(f); Df
Scalar field Delta(f) on the Euclidean plane E^2
sage: Df.display()
Delta(f): E^2 → ℝ
   (x, y) ↦ -(x^2 + y^2)*sin(x*y)
sage: Df.expr()
-(x^2 + y^2)*sin(x*y)
```

The Laplacian of a scalar field is the divergence of its gradient:

```
sage: from sage.manifolds.operators import div, grad
sage: Df == div(grad(f))
True
```

See the method `laplacian()` of `DiffScalarField` and the method `laplacian()` of `TensorField` for more details and examples.

PYTHON MODULE INDEX

m

INDEX