Differential Geometry of Curves and Surfaces

Release 10.6

The Sage Development Team

Apr 02, 2025

CONTENTS

DIFFERENTIAL GEOMETRY OF PARAMETRIZED SURFACES

AUTHORS:

- Mikhail Malakhaltsev (2010-09-25): initial version

- Joris Vankerschaver (2010-10-25): implementation, doctests

class sage.geometry.riemannian_manifolds.parametrized_surface3d.**ParametrizedSurface3D**(*equation*, *variables*, *name=None*)

Bases: `SageObject`

Class representing a parametrized two-dimensional surface in Euclidian three-space. Provides methods for calculating the main geometrical objects related to such a surface, such as the first and the second fundamental form, the total (Gaussian) and the mean curvature, the geodesic curves, parallel transport, etc.

INPUT:

- surface_equation – a 3-tuple of functions specifying a parametric representation of the surface

- variables – a 2-tuple of intrinsic coordinates (u, v) on the surface, with u and v symbolic variables, or a 2-tuple of triples (u, u_{min}, u_{max}), (v, v_{min}, v_{max}) when the parameter range for the coordinates is known

- name – name of the surface (optional)

> **ⓘ Note**
>
> Throughout the documentation, we use the Einstein summation convention: whenever an index appears twice, once as a subscript, and once as a superscript, summation over that index is implied. For instance, $g_{ij}g^{jk}$ stands for $\sum_j g_{ij}g^{jk}$.

EXAMPLES:

We give several examples of standard surfaces in differential geometry. First, let's construct an elliptic paraboloid by explicitly specifying its parametric equation:

```
sage: u, v = var('u,v', domain='real')
sage: eparaboloid = ParametrizedSurface3D((u, v, u^2 + v^2), (u, v),
....:                                      'elliptic paraboloid'); eparaboloid
Parametrized surface ('elliptic paraboloid') with equation (u, v, u^2 + v^2)
```

```
>>> from sage.all import *
>>> u, v = var('u,v', domain='real')
>>> eparaboloid = ParametrizedSurface3D((u, v, u**Integer(2) + v**Integer(2)), (u,
↪ v),
...                                      'elliptic paraboloid'); eparaboloid
Parametrized surface ('elliptic paraboloid') with equation (u, v, u^2 + v^2)
```

When the ranges for the intrinsic coordinates are known, they can be specified explicitly. This is mainly useful for plotting. Here we construct half of an ellipsoid:

```
sage: u1, u2 = var ('u1, u2', domain='real')
sage: coords = ((u1, -pi/2, pi/2), (u2, 0, pi))
sage: ellipsoid_eq = (cos(u1)*cos(u2), 2*sin(u1)*cos(u2), 3*sin(u2))
sage: ellipsoid = ParametrizedSurface3D(ellipsoid_eq, coords, 'ellipsoid');
↪ellipsoid
Parametrized surface ('ellipsoid') with equation
(cos(u1)*cos(u2), 2*cos(u2)*sin(u1), 3*sin(u2))
sage: ellipsoid.plot()                                                      #
↪needs sage.plot
Graphics3d Object
```

```
>>> from sage.all import *
>>> u1, u2 = var ('u1, u2', domain='real')
>>> coords = ((u1, -pi/Integer(2), pi/Integer(2)), (u2, Integer(0), pi))
>>> ellipsoid_eq = (cos(u1)*cos(u2), Integer(2)*sin(u1)*cos(u2),
↪Integer(3)*sin(u2))
>>> ellipsoid = ParametrizedSurface3D(ellipsoid_eq, coords, 'ellipsoid');
↪ellipsoid
Parametrized surface ('ellipsoid') with equation
(cos(u1)*cos(u2), 2*cos(u2)*sin(u1), 3*sin(u2))
>>> ellipsoid.plot()                                                        #
↪needs sage.plot
Graphics3d Object
```

Standard surfaces can be constructed using the `surfaces` generator:

```
sage: klein = surfaces.Klein(); klein
Parametrized surface ('Klein bottle') with equation
(-(sin(1/2*u)*sin(2*v) - cos(1/2*u)*sin(v) - 1)*cos(u),
 -(sin(1/2*u)*sin(2*v) - cos(1/2*u)*sin(v) - 1)*sin(u),
 cos(1/2*u)*sin(2*v) + sin(1/2*u)*sin(v))
```

```
>>> from sage.all import *
>>> klein = surfaces.Klein(); klein
Parametrized surface ('Klein bottle') with equation
(-(sin(1/2*u)*sin(2*v) - cos(1/2*u)*sin(v) - 1)*cos(u),
 -(sin(1/2*u)*sin(2*v) - cos(1/2*u)*sin(v) - 1)*sin(u),
 cos(1/2*u)*sin(2*v) + sin(1/2*u)*sin(v))
```

Latex representation of the surfaces:

```
sage: u, v = var('u, v', domain='real')
sage: sphere = ParametrizedSurface3D((cos(u)*cos(v), sin(u)*cos(v), sin(v)), (u,
```

(continues on next page)

```
↪v),
....:                                    'sphere')
sage: print(latex(sphere))
\left(\cos\left(u\right) \cos\left(v\right), \cos\left(v\right) \sin\left(u\
↪right), \sin\left(v\right)\right)
sage: print(sphere._latex_())
\left(\cos\left(u\right) \cos\left(v\right), \cos\left(v\right) \sin\left(u\
↪right), \sin\left(v\right)\right)
sage: print(sphere)
Parametrized surface ('sphere') with equation (cos(u)*cos(v), cos(v)*sin(u),␣
↪sin(v))
```

```
>>> from sage.all import *
>>> u, v = var('u, v', domain='real')
>>> sphere = ParametrizedSurface3D((cos(u)*cos(v), sin(u)*cos(v), sin(v)), (u, v),
...                                    'sphere')
>>> print(latex(sphere))
\left(\cos\left(u\right) \cos\left(v\right), \cos\left(v\right) \sin\left(u\
↪right), \sin\left(v\right)\right)
>>> print(sphere._latex_())
\left(\cos\left(u\right) \cos\left(v\right), \cos\left(v\right) \sin\left(u\
↪right), \sin\left(v\right)\right)
>>> print(sphere)
Parametrized surface ('sphere') with equation (cos(u)*cos(v), cos(v)*sin(u),␣
↪sin(v))
```

To plot a parametric surface, use the *plot()* member function:

```
sage: enneper = surfaces.Enneper(); enneper
Parametrized surface ('Enneper's surface') with equation
(-1/9*(u^2 - 3*v^2 - 3)*u, -1/9*(3*u^2 - v^2 + 3)*v, 1/3*u^2 - 1/3*v^2)
sage: enneper.plot(aspect_ratio='automatic')                                   #␣
↪needs sage.plot
Graphics3d Object
```

```
>>> from sage.all import *
>>> enneper = surfaces.Enneper(); enneper
Parametrized surface ('Enneper's surface') with equation
(-1/9*(u^2 - 3*v^2 - 3)*u, -1/9*(3*u^2 - v^2 + 3)*v, 1/3*u^2 - 1/3*v^2)
>>> enneper.plot(aspect_ratio='automatic')                                     #␣
↪needs sage.plot
Graphics3d Object
```

We construct an ellipsoid whose axes are given by symbolic variables a, b and c, and find the natural frame of tangent vectors, expressed in intrinsic coordinates. Note that the result is a dictionary of vector fields:

```
sage: a, b, c = var('a, b, c', domain='real')
sage: u1, u2 = var('u1, u2', domain='real')
sage: ellipsoid_eq = (a*cos(u1)*cos(u2), b*sin(u1)*cos(u2), c*sin(u2))
sage: ellipsoid = ParametrizedSurface3D(ellipsoid_eq, (u1, u2),
....:                                    'Symbolic ellipsoid'); ellipsoid
Parametrized surface ('Symbolic ellipsoid') with equation
```

```
(a*cos(u1)*cos(u2), b*cos(u2)*sin(u1), c*sin(u2))

sage: ellipsoid.natural_frame()
{1: (-a*cos(u2)*sin(u1), b*cos(u1)*cos(u2), 0),
 2: (-a*cos(u1)*sin(u2), -b*sin(u1)*sin(u2), c*cos(u2))}
```

```
>>> from sage.all import *
>>> a, b, c = var('a, b, c', domain='real')
>>> u1, u2 = var('u1, u2', domain='real')
>>> ellipsoid_eq = (a*cos(u1)*cos(u2), b*sin(u1)*cos(u2), c*sin(u2))
>>> ellipsoid = ParametrizedSurface3D(ellipsoid_eq, (u1, u2),
...                                     'Symbolic ellipsoid'); ellipsoid
Parametrized surface ('Symbolic ellipsoid') with equation
(a*cos(u1)*cos(u2), b*cos(u2)*sin(u1), c*sin(u2))

>>> ellipsoid.natural_frame()
{1: (-a*cos(u2)*sin(u1), b*cos(u1)*cos(u2), 0),
 2: (-a*cos(u1)*sin(u2), -b*sin(u1)*sin(u2), c*cos(u2))}
```

We find the normal vector field to the surface. The normal vector field is the vector product of the vectors of the natural frame, and is given by:

```
sage: ellipsoid.normal_vector()
(b*c*cos(u1)*cos(u2)^2, a*c*cos(u2)^2*sin(u1), a*b*cos(u2)*sin(u2))
```

```
>>> from sage.all import *
>>> ellipsoid.normal_vector()
(b*c*cos(u1)*cos(u2)^2, a*c*cos(u2)^2*sin(u1), a*b*cos(u2)*sin(u2))
```

By default, the normal vector field is not normalized. To obtain the unit normal vector field of the elliptic paraboloid, we put:

```
sage: u, v = var('u,v', domain='real')
sage: eparaboloid = ParametrizedSurface3D([u, v, u^2 + v^2], [u,v],
....:                                     'elliptic paraboloid')
sage: eparaboloid.normal_vector(normalized=True)
(-2*u/sqrt(4*u^2 + 4*v^2 + 1), -2*v/sqrt(4*u^2 + 4*v^2 + 1), 1/sqrt(4*u^2 + 4*v^2
↪+ 1))
```

```
>>> from sage.all import *
>>> u, v = var('u,v', domain='real')
>>> eparaboloid = ParametrizedSurface3D([u, v, u**Integer(2) + v**Integer(2)], [u,
↪v],
...                                     'elliptic paraboloid')
>>> eparaboloid.normal_vector(normalized=True)
(-2*u/sqrt(4*u^2 + 4*v^2 + 1), -2*v/sqrt(4*u^2 + 4*v^2 + 1), 1/sqrt(4*u^2 + 4*v^2
↪+ 1))
```

Now let us compute the coefficients of the first fundamental form of the torus:

```
sage: u, v = var('u, v', domain='real')
sage: a, b = var('a, b', domain='real')
```

```
sage: torus = ParametrizedSurface3D(((a + b*cos(u))*cos(v),
....:                                  (a + b*cos(u))*sin(v),
....:                                  b*sin(u)), [u,v], 'torus')
sage: torus.first_fundamental_form_coefficients()
{(1, 1): b^2, (1, 2): 0, (2, 1): 0, (2, 2): b^2*cos(u)^2 + 2*a*b*cos(u) + a^2}
```

```
>>> from sage.all import *
>>> u, v = var('u, v', domain='real')
>>> a, b = var('a, b', domain='real')
>>> torus = ParametrizedSurface3D(((a + b*cos(u))*cos(v),
...                                  (a + b*cos(u))*sin(v),
...                                  b*sin(u)), [u,v], 'torus')
>>> torus.first_fundamental_form_coefficients()
{(1, 1): b^2, (1, 2): 0, (2, 1): 0, (2, 2): b^2*cos(u)^2 + 2*a*b*cos(u) + a^2}
```

The first fundamental form can be used to compute the length of a curve on the surface. For example, let us find the length of the curve $u^1 = t$, $u^2 = t$, $t \in [0, 2\pi]$, on the ellipsoid with axes $a = 1$, $b = 1.5$ and $c = 1$. So we take the curve:

```
sage: t = var('t', domain='real')
sage: u1 = t
sage: u2 = t
```

```
>>> from sage.all import *
>>> t = var('t', domain='real')
>>> u1 = t
>>> u2 = t
```

Then find the tangent vector:

```
sage: du1 = diff(u1,t)
sage: du2 = diff(u2,t)
sage: du = vector([du1, du2]); du
(1, 1)
```

```
>>> from sage.all import *
>>> du1 = diff(u1,t)
>>> du2 = diff(u2,t)
>>> du = vector([du1, du2]); du
(1, 1)
```

Once we specify numerical values for the axes of the ellipsoid, we can determine the numerical value of the length integral:

```
sage: L = sqrt(ellipsoid.first_fundamental_form(du, du).substitute(u1=u1, u2=u2))
sage: numerical_integral(L.substitute(a=2, b=1.5, c=1),0,1)[0] # rel tol 1e-11
2.00127905972
```

```
>>> from sage.all import *
>>> L = sqrt(ellipsoid.first_fundamental_form(du, du).substitute(u1=u1, u2=u2))
>>> numerical_integral(L.substitute(a=Integer(2), b=RealNumber('1.5'),␣
```

```
→c=Integer(1)),Integer(0),Integer(1))[Integer(0)] # rel tol 1e-11
2.00127905972
```

We find the area of the sphere of radius R:

```
sage: R = var('R', domain='real')
sage: u, v = var('u,v', domain='real')
sage: assume(R>0)
sage: assume(cos(v)>0)
sage: sphere = ParametrizedSurface3D([R*cos(u)*cos(v), R*sin(u)*cos(v), R*sin(v)],
....:                                 [u,v], 'sphere')
sage: integral(integral(sphere.area_form(), u, 0, 2*pi), v, -pi/2, pi/2)
4*pi*R^2
```

```
>>> from sage.all import *
>>> R = var('R', domain='real')
>>> u, v = var('u,v', domain='real')
>>> assume(R>Integer(0))
>>> assume(cos(v)>Integer(0))
>>> sphere = ParametrizedSurface3D([R*cos(u)*cos(v), R*sin(u)*cos(v), R*sin(v)],
...                                 [u,v], 'sphere')
>>> integral(integral(sphere.area_form(), u, Integer(0), Integer(2)*pi), v, -pi/
→Integer(2), pi/Integer(2))
4*pi*R^2
```

We can find an orthonormal frame field $\{e_1, e_2\}$ of a surface and calculate its structure functions. Let us first determine the orthonormal frame field for the elliptic paraboloid:

```
sage: u, v = var('u,v', domain='real')
sage: eparaboloid = ParametrizedSurface3D([u, v, u^2 + v^2], [u,v],
....:                                       'elliptic paraboloid')
sage: eparaboloid.orthonormal_frame()
{1: (1/sqrt(4*u^2 + 1), 0, 2*u/sqrt(4*u^2 + 1)),
 2: (-4*u*v/(sqrt(4*u^2 + 4*v^2 + 1)*sqrt(4*u^2 + 1)),
     sqrt(4*u^2 + 1)/sqrt(4*u^2 + 4*v^2 + 1),
     2*v/(sqrt(4*u^2 + 4*v^2 + 1)*sqrt(4*u^2 + 1)))}
```

```
>>> from sage.all import *
>>> u, v = var('u,v', domain='real')
>>> eparaboloid = ParametrizedSurface3D([u, v, u**Integer(2) + v**Integer(2)], [u,
→v],
...                                       'elliptic paraboloid')
>>> eparaboloid.orthonormal_frame()
{1: (1/sqrt(4*u^2 + 1), 0, 2*u/sqrt(4*u^2 + 1)),
 2: (-4*u*v/(sqrt(4*u^2 + 4*v^2 + 1)*sqrt(4*u^2 + 1)),
     sqrt(4*u^2 + 1)/sqrt(4*u^2 + 4*v^2 + 1),
     2*v/(sqrt(4*u^2 + 4*v^2 + 1)*sqrt(4*u^2 + 1)))}
```

We can express the orthogonal frame field both in exterior coordinates (i.e. expressed as vector field fields in the ambient space $\mathbf{R}^3$, the default) or in intrinsic coordinates (with respect to the natural frame). Here we use intrinsic coordinates:

```
sage: eparaboloid.orthonormal_frame(coordinates='int')
{1: (1/sqrt(4*u^2 + 1), 0),
 2: (-4*u*v/(sqrt(4*u^2 + 4*v^2 + 1)*sqrt(4*u^2 + 1)),
     sqrt(4*u^2 + 1)/sqrt(4*u^2 + 4*v^2 + 1))}
```

```
>>> from sage.all import *
>>> eparaboloid.orthonormal_frame(coordinates='int')
{1: (1/sqrt(4*u^2 + 1), 0),
 2: (-4*u*v/(sqrt(4*u^2 + 4*v^2 + 1)*sqrt(4*u^2 + 1)),
     sqrt(4*u^2 + 1)/sqrt(4*u^2 + 4*v^2 + 1))}
```

Using the orthonormal frame in interior coordinates, we can calculate the structure functions c_{ij}^k of the surface, defined by $[e_i, e_j] = c_{ij}^k e_k$, where $[e_i, e_j]$ represents the Lie bracket of two frame vector fields e_i, e_j. For the elliptic paraboloid, we get:

```
sage: EE = eparaboloid.orthonormal_frame(coordinates='int')
sage: E1 = EE[1]; E2 = EE[2]
sage: CC = eparaboloid.frame_structure_functions(E1,E2)
sage: CC[1,2,1].simplify_full()
4*sqrt(4*u^2 + 4*v^2 + 1)*v/((16*u^4 + 4*(4*u^2 + 1)*v^2 + 8*u^2 + 1)*sqrt(4*u^2␣
↪+ 1))
```

```
>>> from sage.all import *
>>> EE = eparaboloid.orthonormal_frame(coordinates='int')
>>> E1 = EE[Integer(1)]; E2 = EE[Integer(2)]
>>> CC = eparaboloid.frame_structure_functions(E1,E2)
>>> CC[Integer(1),Integer(2),Integer(1)].simplify_full()
4*sqrt(4*u^2 + 4*v^2 + 1)*v/((16*u^4 + 4*(4*u^2 + 1)*v^2 + 8*u^2 + 1)*sqrt(4*u^2␣
↪+ 1))
```

We compute the Gaussian and mean curvatures of the sphere:

```
sage: sphere = surfaces.Sphere(); sphere
Parametrized surface ('Sphere') with equation (cos(u)*cos(v), cos(v)*sin(u),␣
↪sin(v))
sage: K = sphere.gauss_curvature(); K  # Not tested -- see trac 12737
1
sage: H = sphere.mean_curvature(); H  # Not tested -- see trac 12737
-1
```

```
>>> from sage.all import *
>>> sphere = surfaces.Sphere(); sphere
Parametrized surface ('Sphere') with equation (cos(u)*cos(v), cos(v)*sin(u),␣
↪sin(v))
>>> K = sphere.gauss_curvature(); K  # Not tested -- see trac 12737
1
>>> H = sphere.mean_curvature(); H  # Not tested -- see trac 12737
-1
```

We can easily generate a color plot of the Gaussian curvature of a surface. Here we deal with the ellipsoid:

```
sage: # needs numpy
sage: u1, u2 = var('u1,u2', domain='real')
sage: u = [u1,u2]
sage: ellipsoid_equation(u1,u2) = [2*cos(u1)*cos(u2),1.5*cos(u1)*sin(u2),sin(u1)]
sage: ellipsoid = ParametrizedSurface3D(ellipsoid_equation(u1,u2), [u1, u2],
→'ellipsoid')
sage: # set intervals for variables and the number of division points
sage: u1min, u1max = -1.5, 1.5
sage: u2min, u2max = 0, 6.28
sage: u1num, u2num = 10, 20
sage: # make the arguments array
sage: from numpy import linspace
sage: u1_array = linspace(u1min, u1max, u1num)
sage: u2_array = linspace(u2min, u2max, u2num)
sage: u_array = [(uu1,uu2) for uu1 in u1_array for uu2 in u2_array]
sage: # Find the gaussian curvature
sage: K(u1,u2) = ellipsoid.gauss_curvature()
sage: # Make array of K values
sage: K_array = [K(uu[0],uu[1]) for uu in u_array]
sage: # Find minimum and max of the Gauss curvature
sage: K_max = max(K_array)
sage: K_min = min(K_array)
sage: # Make the array of color coefficients
sage: cc_array = [(ccc - K_min)/(K_max - K_min) for ccc in K_array]
sage: points_array = [ellipsoid_equation(u_array[counter][0],
....:                             u_array[counter][1])
....:              for counter in range(len(u_array))]
sage: curvature_ellipsoid_plot = sum(point([xx                          #␣
→needs sage.plot
....:                                   for xx in points_array[counter]],
....:                                color=hue(cc_array[counter]/2))
....:                            for counter in range(len(u_array)))
sage: curvature_ellipsoid_plot.show(aspect_ratio=1)                     #␣
→needs sage.plot
```

```
>>> from sage.all import *
>>> # needs numpy
>>> u1, u2 = var('u1,u2', domain='real')
>>> u = [u1,u2]
>>> __tmp__=var("u1,u2"); ellipsoid_equation = symbolic_
→expression([Integer(2)*cos(u1)*cos(u2),RealNumber('1.5')*cos(u1)*sin(u2),
→sin(u1)]).function(u1,u2)
>>> ellipsoid = ParametrizedSurface3D(ellipsoid_equation(u1,u2), [u1, u2],
→'ellipsoid')
>>> # set intervals for variables and the number of division points
>>> u1min, u1max = -RealNumber('1.5'), RealNumber('1.5')
>>> u2min, u2max = Integer(0), RealNumber('6.28')
>>> u1num, u2num = Integer(10), Integer(20)
>>> # make the arguments array
>>> from numpy import linspace
>>> u1_array = linspace(u1min, u1max, u1num)
>>> u2_array = linspace(u2min, u2max, u2num)
```

(continues on next page)

```
>>> u_array = [(uu1,uu2) for uu1 in u1_array for uu2 in u2_array]
>>> # Find the gaussian curvature
>>> __tmp__=var("u1,u2"); K = symbolic_expression(ellipsoid.gauss_curvature()).
↪function(u1,u2)
>>> # Make array of K values
>>> K_array = [K(uu[Integer(0)],uu[Integer(1)]) for uu in u_array]
>>> # Find minimum and max of the Gauss curvature
>>> K_max = max(K_array)
>>> K_min = min(K_array)
>>> # Make the array of color coefficients
>>> cc_array = [(ccc - K_min)/(K_max - K_min) for ccc in K_array]
>>> points_array = [ellipsoid_equation(u_array[counter][Integer(0)],
...                                     u_array[counter][Integer(1)])
...              for counter in range(len(u_array))]
>>> curvature_ellipsoid_plot = sum(point([xx                              #␣
↪needs sage.plot
...                                 for xx in points_array[counter]],
...                               color=hue(cc_array[counter]/Integer(2)))
...                           for counter in range(len(u_array)))
>>> curvature_ellipsoid_plot.show(aspect_ratio=Integer(1))                    ␣
↪      # needs sage.plot
```

We can find the principal curvatures and principal directions of the elliptic paraboloid:

```
sage: u, v = var('u, v', domain='real')
sage: eparaboloid = ParametrizedSurface3D([u, v, u^2+v^2], [u, v],
....:                                      'elliptic paraboloid')
sage: pd = eparaboloid.principal_directions(); pd
[(2*sqrt(4*u^2 + 4*v^2 + 1)/(16*u^4 + 16*v^4 + 8*(4*u^2 + 1)*v^2 + 8*u^2 + 1),␣
↪[(1, v/u)], 1),
 (2/sqrt(4*u^2 + 4*v^2 + 1), [(1, -u/v)], 1)]
```

```
>>> from sage.all import *
>>> u, v = var('u, v', domain='real')
>>> eparaboloid = ParametrizedSurface3D([u, v, u**Integer(2)+v**Integer(2)], [u,␣
↪v],
...                                      'elliptic paraboloid')
>>> pd = eparaboloid.principal_directions(); pd
[(2*sqrt(4*u^2 + 4*v^2 + 1)/(16*u^4 + 16*v^4 + 8*(4*u^2 + 1)*v^2 + 8*u^2 + 1),␣
↪[(1, v/u)], 1),
 (2/sqrt(4*u^2 + 4*v^2 + 1), [(1, -u/v)], 1)]
```

We extract the principal curvatures:

```
sage: k1 = pd[0][0].simplify_full(); k1
2*sqrt(4*u^2 + 4*v^2 + 1)/(16*u^4 + 16*v^4 + 8*(4*u^2 + 1)*v^2 + 8*u^2 + 1)
sage: k2 = pd[1][0].simplify_full(); k2
2/sqrt(4*u^2 + 4*v^2 + 1)
```

```
>>> from sage.all import *
>>> k1 = pd[Integer(0)][Integer(0)].simplify_full(); k1
2*sqrt(4*u^2 + 4*v^2 + 1)/(16*u^4 + 16*v^4 + 8*(4*u^2 + 1)*v^2 + 8*u^2 + 1)
```

```
>>> k2 = pd[Integer(1)][Integer(0)].simplify_full(); k2
2/sqrt(4*u^2 + 4*v^2 + 1)
```

and check them by comparison with the Gaussian and mean curvature expressed in terms of the principal curvatures:

```
sage: K = eparaboloid.gauss_curvature().simplify_full(); K
4/(16*u^4 + 16*v^4 + 8*(4*u^2 + 1)*v^2 + 8*u^2 + 1)
sage: H = eparaboloid.mean_curvature().simplify_full(); H
2*(2*u^2 + 2*v^2 + 1)/(4*u^2 + 4*v^2 + 1)^(3/2)
sage: (K - k1*k2).simplify_full()
0
sage: (2*H - k1 - k2).simplify_full()
0
```

```
>>> from sage.all import *
>>> K = eparaboloid.gauss_curvature().simplify_full(); K
4/(16*u^4 + 16*v^4 + 8*(4*u^2 + 1)*v^2 + 8*u^2 + 1)
>>> H = eparaboloid.mean_curvature().simplify_full(); H
2*(2*u^2 + 2*v^2 + 1)/(4*u^2 + 4*v^2 + 1)^(3/2)
>>> (K - k1*k2).simplify_full()
0
>>> (Integer(2)*H - k1 - k2).simplify_full()
0
```

We can find the intrinsic (local coordinates) of the principal directions:

```
sage: pd[0][1]
[(1, v/u)]
sage: pd[1][1]
[(1, -u/v)]
```

```
>>> from sage.all import *
>>> pd[Integer(0)][Integer(1)]
[(1, v/u)]
>>> pd[Integer(1)][Integer(1)]
[(1, -u/v)]
```

The ParametrizedSurface3D class also contains functionality to compute the coefficients of the second fundamental form, the shape operator, the rotation on the surface at a given angle, the connection coefficients. One can also calculate numerically the geodesics and the parallel translation along a curve.

Here we compute a number of geodesics on the sphere emanating from the point (1, 0, 0), in various directions. The geodesics intersect again in the antipodal point (-1, 0, 0), indicating that these points are conjugate:

```
sage: S = surfaces.Sphere()
sage: g1 = [c[-1] for c in S.geodesics_numerical((0,0), (1,0), (0,2*pi,100))]
sage: g2 = [c[-1] for c in S.geodesics_numerical((0,0),
....:                                            (cos(pi/3),sin(pi/3)),
....:                                            (0,2*pi,100))]
sage: g3 = [c[-1] for c in S.geodesics_numerical((0,0),
....:                                            (cos(2*pi/3),sin(2*pi/3)),
....:                                            (0,2*pi,100))]
sage: (S.plot(opacity=0.3) + line3d(g1, color='red')                        #␣
```

```
→needs sage.plot
....:        + line3d(g2, color='red') + line3d(g3, color='red')).show()
```

```
>>> from sage.all import *
>>> S = surfaces.Sphere()
>>> g1 = [c[-Integer(1)] for c in S.geodesics_numerical((Integer(0),Integer(0)),␣
→(Integer(1),Integer(0)), (Integer(0),Integer(2)*pi,Integer(100)))]
>>> g2 = [c[-Integer(1)] for c in S.geodesics_numerical((Integer(0),Integer(0)),
...                                              (cos(pi/Integer(3)),sin(pi/
→Integer(3))),
...                                              (Integer(0),Integer(2)*pi,
→Integer(100)))]
>>> g3 = [c[-Integer(1)] for c in S.geodesics_numerical((Integer(0),Integer(0)),
...                                              (cos(Integer(2)*pi/Integer(3)),
→sin(Integer(2)*pi/Integer(3))),
...                                              (Integer(0),Integer(2)*pi,
→Integer(100)))]
>>> (S.plot(opacity=RealNumber('0.3')) + line3d(g1, color='red')                ␣
→             # needs sage.plot
...        + line3d(g2, color='red') + line3d(g3, color='red')).show()
```

area_form()

Return the coefficient of the area form on the surface. In terms of the coefficients g_{ij} (where $i, j = 1, 2$) of the first fundamental form, the coefficient of the area form is given by $A = \sqrt{g_{11}g_{22} - g_{12}^2}$.

See also *area_form_squared()*.

OUTPUT: coefficient of the area form

EXAMPLES:

```
sage: u, v = var('u,v', domain='real')
sage: sphere = ParametrizedSurface3D([cos(u)*cos(v),sin(u)*cos(v),sin(v)],[u,
→v],'sphere')
sage: sphere.area_form()
cos(v)
```

```
>>> from sage.all import *
>>> u, v = var('u,v', domain='real')
>>> sphere = ParametrizedSurface3D([cos(u)*cos(v),sin(u)*cos(v),sin(v)],[u,v],
→'sphere')
>>> sphere.area_form()
cos(v)
```

area_form_squared()

Return the square of the coefficient of the area form on the surface. In terms of the coefficients g_{ij} (where $i, j = 1, 2$) of the first fundamental form, this invariant is given by $A^2 = g_{11}g_{22} - g_{12}^2$.

See also *area_form()*.

OUTPUT: square of the area form

EXAMPLES:

```
sage: u, v = var('u, v', domain='real')
sage: sphere = ParametrizedSurface3D([cos(u)*cos(v),sin(u)*cos(v),sin(v)],[u,
↪v],'sphere')
sage: sphere.area_form_squared()
cos(v)^2
```

```
>>> from sage.all import *
>>> u, v = var('u, v', domain='real')
>>> sphere = ParametrizedSurface3D([cos(u)*cos(v),sin(u)*cos(v),sin(v)],[u,v],
↪'sphere')
>>> sphere.area_form_squared()
cos(v)^2
```

connection_coefficients()

> Compute the connection coefficients or Christoffel symbols Γ_{ij}^k of the surface. If the coefficients of the first fundamental form are given by g_{ij} (where $i, j = 1, 2$), then $\Gamma_{ij}^k = \frac{1}{2} g^{kl} \left(\frac{\partial g_{li}}{\partial x^j} - \frac{\partial g_{ij}}{\partial x^l} + \frac{\partial g_{lj}}{\partial x^i} \right)$. Here, (g^{kl}) is the inverse of the matrix (g_{ij}), with $i, j = 1, 2$.
>
> OUTPUT:
>
> Dictionary of connection coefficients, where the keys are 3-tuples (i, j, k) and the values are the corresponding coefficients Γ_{ij}^k.
>
> EXAMPLES:

```
sage: r = var('r')
sage: assume(r > 0)
sage: u, v = var('u,v', domain='real')
sage: assume(cos(v)>0)
sage: sphere = ParametrizedSurface3D([r*cos(u)*cos(v),r*sin(u)*cos(v),
↪r*sin(v)],[u,v],'sphere')
sage: sphere.connection_coefficients()
{(1, 1, 1): 0,
 (1, 1, 2): cos(v)*sin(v),
 (1, 2, 1): -sin(v)/cos(v),
 (1, 2, 2): 0,
 (2, 1, 1): -sin(v)/cos(v),
 (2, 1, 2): 0,
 (2, 2, 1): 0,
 (2, 2, 2): 0}
```

```
>>> from sage.all import *
>>> r = var('r')
>>> assume(r > Integer(0))
>>> u, v = var('u,v', domain='real')
>>> assume(cos(v)>Integer(0))
>>> sphere = ParametrizedSurface3D([r*cos(u)*cos(v),r*sin(u)*cos(v),r*sin(v)],
↪[u,v],'sphere')
>>> sphere.connection_coefficients()
{(1, 1, 1): 0,
 (1, 1, 2): cos(v)*sin(v),
 (1, 2, 1): -sin(v)/cos(v),
 (1, 2, 2): 0,
```

(continues on next page)

```
(2, 1, 1): -sin(v)/cos(v),
(2, 1, 2): 0,
(2, 2, 1): 0,
(2, 2, 2): 0}
```

first_fundamental_form(*vector1*, *vector2*)

Evaluate the first fundamental form on two vectors expressed with respect to the natural coordinate frame on the surface. In other words, if the vectors are $v = (v^1, v^2)$ and $w = (w^1, w^2)$, calculate $g_{11}v^1w^1 + g_{12}(v^1w^2 + v^2w^1) + g_{22}v^2w^2$, with g_{ij} the coefficients of the first fundamental form.

INPUT:

- `vector1`, `vector2` – vectors on the surface

OUTPUT: first fundamental form evaluated on the input vectors

EXAMPLES:

```
sage: u, v = var('u, v', domain='real')
sage: v1, v2, w1, w2 = var('v1, v2, w1, w2', domain='real')
sage: sphere = ParametrizedSurface3D((cos(u)*cos(v), sin(u)*cos(v), sin(v)),␣
↪(u, v),'sphere')
sage: sphere.first_fundamental_form(vector([v1,v2]),vector([w1,w2]))
v1*w1*cos(v)^2 + v2*w2

sage: vv = vector([1,2])
sage: sphere.first_fundamental_form(vv,vv)
cos(v)^2 + 4

sage: sphere.first_fundamental_form([1,1],[2,1])
2*cos(v)^2 + 1
```

```
>>> from sage.all import *
>>> u, v = var('u, v', domain='real')
>>> v1, v2, w1, w2 = var('v1, v2, w1, w2', domain='real')
>>> sphere = ParametrizedSurface3D((cos(u)*cos(v), sin(u)*cos(v), sin(v)), (u,
↪ v),'sphere')
>>> sphere.first_fundamental_form(vector([v1,v2]),vector([w1,w2]))
v1*w1*cos(v)^2 + v2*w2

>>> vv = vector([Integer(1),Integer(2)])
>>> sphere.first_fundamental_form(vv,vv)
cos(v)^2 + 4

>>> sphere.first_fundamental_form([Integer(1),Integer(1)],[Integer(2),
↪Integer(1)])
2*cos(v)^2 + 1
```

first_fundamental_form_coefficient(*index*)

Compute a single component g_{ij} of the first fundamental form. If the parametric representation of the surface is given by the vector function $\vec{r}(u^i)$, where u^i, $i = 1, 2$ are curvilinear coordinates, then $g_{ij} = \frac{\partial \vec{r}}{\partial u^i} \cdot \frac{\partial \vec{r}}{\partial u^j}$.

INPUT:

- `index` – tuple `(i, j)` specifying the index of the component g_{ij}

OUTPUT: component g_{ij} of the first fundamental form

EXAMPLES:

```
sage: u, v = var('u, v', domain='real')
sage: eparaboloid = ParametrizedSurface3D((u, v, u^2+v^2), (u, v))
sage: eparaboloid.first_fundamental_form_coefficient((1,2))
4*u*v
```

```
>>> from sage.all import *
>>> u, v = var('u, v', domain='real')
>>> eparaboloid = ParametrizedSurface3D((u, v, u**Integer(2)+v**Integer(2)),
→(u, v))
>>> eparaboloid.first_fundamental_form_coefficient((Integer(1),Integer(2)))
4*u*v
```

When the index is invalid, an error is raised:

```
sage: u, v = var('u, v', domain='real')
sage: eparaboloid = ParametrizedSurface3D((u, v, u^2+v^2), (u, v))
sage: eparaboloid.first_fundamental_form_coefficient((1,5))
Traceback (most recent call last):
...
ValueError: Index (1, 5) out of bounds.
```

```
>>> from sage.all import *
>>> u, v = var('u, v', domain='real')
>>> eparaboloid = ParametrizedSurface3D((u, v, u**Integer(2)+v**Integer(2)),
→(u, v))
>>> eparaboloid.first_fundamental_form_coefficient((Integer(1),Integer(5)))
Traceback (most recent call last):
...
ValueError: Index (1, 5) out of bounds.
```

first_fundamental_form_coefficients()

Return the coefficients of the first fundamental form as a dictionary. The keys are tuples (i, j), where i and j range over $1, 2$, while the values are the corresponding coefficients g_{ij}.

OUTPUT: dictionary of first fundamental form coefficients

EXAMPLES:

```
sage: u, v = var('u,v', domain='real')
sage: sphere = ParametrizedSurface3D((cos(u)*cos(v), sin(u)*cos(v), sin(v)),
→(u, v), 'sphere')
sage: sphere.first_fundamental_form_coefficients()
{(1, 1): cos(v)^2, (1, 2): 0, (2, 1): 0, (2, 2): 1}
```

```
>>> from sage.all import *
>>> u, v = var('u,v', domain='real')
>>> sphere = ParametrizedSurface3D((cos(u)*cos(v), sin(u)*cos(v), sin(v)), (u,
→ v), 'sphere')
>>> sphere.first_fundamental_form_coefficients()
{(1, 1): cos(v)^2, (1, 2): 0, (2, 1): 0, (2, 2): 1}
```

first_fundamental_form_inverse_coefficient(*index*)

Return a specific component g^{ij} of the inverse of the fundamental form.

INPUT:

- index – tuple (i, j) specifying the index of the component g^{ij}

OUTPUT: component of the inverse of the fundamental form

EXAMPLES:

```
sage: u, v = var('u, v', domain='real')
sage: sphere = ParametrizedSurface3D([cos(u)*cos(v),sin(u)*cos(v),sin(v)],[u,
→v],'sphere')
sage: sphere.first_fundamental_form_inverse_coefficient((1, 2))
0
sage: sphere.first_fundamental_form_inverse_coefficient((1, 1))
cos(v)^(-2)
```

```
>>> from sage.all import *
>>> u, v = var('u, v', domain='real')
>>> sphere = ParametrizedSurface3D([cos(u)*cos(v),sin(u)*cos(v),sin(v)],[u,v],
→'sphere')
>>> sphere.first_fundamental_form_inverse_coefficient((Integer(1),␣
→Integer(2)))
0
>>> sphere.first_fundamental_form_inverse_coefficient((Integer(1),␣
→Integer(1)))
cos(v)^(-2)
```

first_fundamental_form_inverse_coefficients()

Return the coefficients g^{ij} of the inverse of the fundamental form, as a dictionary. The inverse coefficients are defined by $g^{ij}g_{jk} = \delta^i_k$ with δ^i_k the Kronecker delta.

OUTPUT: dictionary of the inverse coefficients

EXAMPLES:

```
sage: u, v = var('u, v', domain='real')
sage: sphere = ParametrizedSurface3D([cos(u)*cos(v),sin(u)*cos(v),sin(v)],[u,
→v],'sphere')
sage: sphere.first_fundamental_form_inverse_coefficients()
{(1, 1): cos(v)^(-2), (1, 2): 0, (2, 1): 0, (2, 2): 1}
```

```
>>> from sage.all import *
>>> u, v = var('u, v', domain='real')
>>> sphere = ParametrizedSurface3D([cos(u)*cos(v),sin(u)*cos(v),sin(v)],[u,v],
→'sphere')
>>> sphere.first_fundamental_form_inverse_coefficients()
{(1, 1): cos(v)^(-2), (1, 2): 0, (2, 1): 0, (2, 2): 1}
```

frame_structure_functions(*e1, e2*)

Return the structure functions c^k_{ij} for a frame field e_1, e_2, i.e. a pair of vector fields on the surface which are linearly independent at each point. The structure functions are defined using the Lie bracket by $[e_i, e_j] = c^k_{ij}e_k$.

INPUT:

- `e1`, `e2` – vector fields in intrinsic coordinates on the surface, expressed as pairs of functions, or as vectors of length 2

OUTPUT:

Dictionary of structure functions, where the key (`i`, `j`, `k`) refers to the structure function $c_{i,j}^k$.

EXAMPLES:

```
sage: u, v = var('u, v', domain='real')
sage: assume(cos(v) > 0)
sage: sphere = ParametrizedSurface3D([cos(u)*cos(v), sin(u)*cos(v), sin(v)],␣
→[u, v], 'sphere')
sage: sphere.frame_structure_functions([u, v], [-v, u])
{(1, 1, 1): 0,
 (1, 1, 2): 0,
 (1, 2, 1): 0,
 (1, 2, 2): 0,
 (2, 1, 1): 0,
 (2, 1, 2): 0,
 (2, 2, 1): 0,
 (2, 2, 2): 0}
```

```
>>> from sage.all import *
>>> u, v = var('u, v', domain='real')
>>> assume(cos(v) > Integer(0))
>>> sphere = ParametrizedSurface3D([cos(u)*cos(v), sin(u)*cos(v), sin(v)], [u,
→ v], 'sphere')
>>> sphere.frame_structure_functions([u, v], [-v, u])
{(1, 1, 1): 0,
 (1, 1, 2): 0,
 (1, 2, 1): 0,
 (1, 2, 2): 0,
 (2, 1, 1): 0,
 (2, 1, 2): 0,
 (2, 2, 1): 0,
 (2, 2, 2): 0}
```

We construct the structure functions of the orthonormal frame on the surface:

```
sage: EE_int = sphere.orthonormal_frame(coordinates='int')
sage: CC = sphere.frame_structure_functions(EE_int[1],EE_int[2]); CC
{(1, 1, 1): 0,
 (1, 1, 2): 0,
 (1, 2, 1): sin(v)/cos(v),
 (1, 2, 2): 0,
 (2, 1, 1): -sin(v)/cos(v),
 (2, 1, 2): 0,
 (2, 2, 1): 0,
 (2, 2, 2): 0}
sage: sphere.lie_bracket(EE_int[1],EE_int[2]) - CC[(1,2,1)]*EE_int[1] - CC[(1,
→2,2)]*EE_int[2]
(0, 0)
```

```
>>> from sage.all import *
>>> EE_int = sphere.orthonormal_frame(coordinates='int')
>>> CC = sphere.frame_structure_functions(EE_int[Integer(1)],EE_
→int[Integer(2)]); CC
{(1, 1, 1): 0,
 (1, 1, 2): 0,
 (1, 2, 1): sin(v)/cos(v),
 (1, 2, 2): 0,
 (2, 1, 1): -sin(v)/cos(v),
 (2, 1, 2): 0,
 (2, 2, 1): 0,
 (2, 2, 2): 0}
>>> sphere.lie_bracket(EE_int[Integer(1)],EE_int[Integer(2)]) -␣
→CC[(Integer(1),Integer(2),Integer(1))]*EE_int[Integer(1)] - CC[(Integer(1),
→Integer(2),Integer(2))]*EE_int[Integer(2)]
(0, 0)
```

gauss_curvature()

Finds the gaussian curvature of the surface, given by $K = \frac{h_{11}h_{22} - h_{12}^2}{g_{11}g_{22} - g_{12}^2}$, where g_{ij} and h_{ij} are the coefficients of the first and second fundamental form, respectively.

OUTPUT: Gaussian curvature of the surface.

EXAMPLES:

```
sage: R = var('R')
sage: assume(R>0)
sage: u, v = var('u,v', domain='real')
sage: assume(cos(v)>0)
sage: sphere = ParametrizedSurface3D([R*cos(u)*cos(v),R*sin(u)*cos(v),
→R*sin(v)],[u,v],'sphere')
sage: sphere.gauss_curvature()
R^(-2)
```

```
>>> from sage.all import *
>>> R = var('R')
>>> assume(R>Integer(0))
>>> u, v = var('u,v', domain='real')
>>> assume(cos(v)>Integer(0))
>>> sphere = ParametrizedSurface3D([R*cos(u)*cos(v),R*sin(u)*cos(v),R*sin(v)],
→[u,v],'sphere')
>>> sphere.gauss_curvature()
R^(-2)
```

geodesics_numerical(*p0*, *v0*, *tinterval*)

Numerical integration of the geodesic equations. Explicitly, the geodesic equations are given by $\frac{d^2 u^i}{dt^2} + \Gamma^i_{jk} \frac{du^j}{dt} \frac{du^k}{dt} = 0$.

Solving these equations gives the coordinates (u^1, u^2) of the geodesic on the surface. The coordinates in space can then be found by substituting (u^1, u^2) into the vector $\vec{r}(u^1, u^2)$ representing the surface.

ALGORITHM:

The geodesic equations are integrated forward in time using the ode solvers from sage.calculus.ode.

See the member function `_create_geodesic_ode_system` for more details.

INPUT:

- `p0` – 2-tuple with coordinates of the initial point

- `v0` – 2-tuple with components of the initial tangent vector to the geodesic

- `tinterval` – list `[a, b, M]`, where `(a,b)` is the domain of the geodesic and `M` is the number of subdivision points used when returning the solution

OUTPUT:

List of lists `[t, [u1(t), u2(t)], [v1(t), v2(t)], [x1(t), x2(t), x3(t)]]`, where

- `t` – a subdivision point;

- `[u1(t), u2(t)]` are the intrinsic coordinates of the geodesic point;

- `[v1(t), v2(t)]` are the intrinsic coordinates of the tangent vector to the geodesic;

- `[x1(t), x2(t), x3(t)]` are the coordinates of the geodesic point in the three-dimensional space.

EXAMPLES:

```
sage: p, q = var('p,q', domain='real')
sage: assume(cos(q)>0)
sage: sphere = ParametrizedSurface3D([cos(q)*cos(p),sin(q)*cos(p),sin(p)],[p,
↪q],'sphere')
sage: geodesic = sphere.geodesics_numerical([0.0,0.0],[1.0,1.0],[0,2*pi,5])
sage: times, points, tangent_vectors, ext_points = zip(*geodesic)

sage: round4 = lambda vec: [N(x, digits=4) for x in vec]   # helper function␣
↪to round to 4 digits
sage: round4(times)
[0.0000, 1.257, 2.513, 3.770, 5.027, 6.283]
sage: [round4(p) for p in points]
[[0.0000, 0.0000], [0.7644, 1.859], [-0.2876, 3.442], [-0.6137, 5.502], [0.
↪5464, 6.937], [0.3714, 9.025]]
sage: [round4(p) for p in ext_points]
[[1.000, 0.0000, 0.0000], [-0.2049, 0.6921, 0.6921], [-0.9160, -0.2836, -0.
↪2836], [0.5803, -0.5759, -0.5759], [0.6782, 0.5196, 0.5196], [-0.8582, 0.
↪3629, 0.3629]]
```

```
>>> from sage.all import *
>>> p, q = var('p,q', domain='real')
>>> assume(cos(q)>Integer(0))
>>> sphere = ParametrizedSurface3D([cos(q)*cos(p),sin(q)*cos(p),sin(p)],[p,q],
↪'sphere')
>>> geodesic = sphere.geodesics_numerical([RealNumber('0.0'),RealNumber('0.0
↪')],[RealNumber('1.0'),RealNumber('1.0')],[Integer(0),Integer(2)*pi,
↪Integer(5)])
>>> times, points, tangent_vectors, ext_points = zip(*geodesic)

>>> round4 = lambda vec: [N(x, digits=Integer(4)) for x in vec]   # helper␣
↪function to round to 4 digits
>>> round4(times)
[0.0000, 1.257, 2.513, 3.770, 5.027, 6.283]
>>> [round4(p) for p in points]
```

```
[[0.0000, 0.0000], [0.7644, 1.859], [-0.2876, 3.442], [-0.6137, 5.502], [0.
↪5464, 6.937], [0.3714, 9.025]]
>>> [round4(p) for p in ext_points]
[[1.000, 0.0000, 0.0000], [-0.2049, 0.6921, 0.6921], [-0.9160, -0.2836, -0.
↪2836], [0.5803, -0.5759, -0.5759], [0.6782, 0.5196, 0.5196], [-0.8582, 0.
↪3629, 0.3629]]
```

lie_bracket(*v*, *w*)

Return the Lie bracket of two vector fields that are tangent to the surface. The vector fields should be given in intrinsic coordinates, i.e. with respect to the natural frame.

INPUT:

- v, w – vector fields on the surface, expressed as pairs of functions or as vectors of length 2

OUTPUT: the Lie bracket $[v, w]$

EXAMPLES:

```
sage: u, v = var('u, v', domain='real')
sage: assume(cos(v)>0)
sage: sphere = ParametrizedSurface3D([cos(u)*cos(v),sin(u)*cos(v),sin(v)],[u,
↪v],'sphere')
sage: sphere.lie_bracket([u,v],[-v,u])
(0, 0)

sage: EE_int = sphere.orthonormal_frame(coordinates='int')
sage: sphere.lie_bracket(EE_int[1],EE_int[2])
(sin(v)/cos(v)^2, 0)
```

```
>>> from sage.all import *
>>> u, v = var('u, v', domain='real')
>>> assume(cos(v)>Integer(0))
>>> sphere = ParametrizedSurface3D([cos(u)*cos(v),sin(u)*cos(v),sin(v)],[u,v],
↪'sphere')
>>> sphere.lie_bracket([u,v],[-v,u])
(0, 0)

>>> EE_int = sphere.orthonormal_frame(coordinates='int')
>>> sphere.lie_bracket(EE_int[Integer(1)],EE_int[Integer(2)])
(sin(v)/cos(v)^2, 0)
```

mean_curvature()

Finds the mean curvature of the surface, given by $H = \frac{1}{2}\frac{g_{22}h_{11}-2g_{12}h_{12}+g_{11}h_{22}}{g_{11}g_{22}-g_{12}^2}$, where g_{ij} and h_{ij} are the components of the first and second fundamental forms, respectively.

OUTPUT: mean curvature of the surface

EXAMPLES:

```
sage: R = var('R')
sage: assume(R>0)
sage: u, v = var('u,v', domain='real')
sage: assume(cos(v)>0)
```

```
sage: sphere = ParametrizedSurface3D([R*cos(u)*cos(v),R*sin(u)*cos(v),
↪R*sin(v)],[u,v],'sphere')
sage: sphere.mean_curvature()
-1/R
```

```
>>> from sage.all import *
>>> R = var('R')
>>> assume(R>Integer(0))
>>> u, v = var('u,v', domain='real')
>>> assume(cos(v)>Integer(0))
>>> sphere = ParametrizedSurface3D([R*cos(u)*cos(v),R*sin(u)*cos(v),R*sin(v)],
↪[u,v],'sphere')
>>> sphere.mean_curvature()
-1/R
```

natural_frame()

> Return the natural tangent frame on the parametrized surface. The vectors of this frame are tangent to the coordinate lines on the surface.
>
> OUTPUT: the natural frame as a dictionary
>
> EXAMPLES:

```
sage: u, v = var('u, v', domain='real')
sage: eparaboloid = ParametrizedSurface3D((u, v, u^2+v^2), (u, v),
....:                                 'elliptic paraboloid')
sage: eparaboloid.natural_frame()
{1: (1, 0, 2*u), 2: (0, 1, 2*v)}
```

```
>>> from sage.all import *
>>> u, v = var('u, v', domain='real')
>>> eparaboloid = ParametrizedSurface3D((u, v, u**Integer(2)+v**Integer(2)),
↪(u, v),
...                                 'elliptic paraboloid')
>>> eparaboloid.natural_frame()
{1: (1, 0, 2*u), 2: (0, 1, 2*v)}
```

normal_vector(*normalized=False*)

> Return the normal vector field of the parametrized surface.
>
> INPUT:
>
> - `normalized` – boolean (default: `False`); specifies whether the normal vector should be normalized
>
> OUTPUT: normal vector field
>
> EXAMPLES:

```
sage: u, v = var('u, v', domain='real')
sage: eparaboloid = ParametrizedSurface3D((u, v, u^2 + v^2), (u, v),
....:                                 'elliptic paraboloid')
sage: eparaboloid.normal_vector(normalized=False)
(-2*u, -2*v, 1)
sage: eparaboloid.normal_vector(normalized=True)
```

```
(-2*u/sqrt(4*u^2 + 4*v^2 + 1),
 -2*v/sqrt(4*u^2 + 4*v^2 + 1),
 1/sqrt(4*u^2 + 4*v^2 + 1))
```

```
>>> from sage.all import *
>>> u, v = var('u, v', domain='real')
>>> eparaboloid = ParametrizedSurface3D((u, v, u**Integer(2) + v**Integer(2)),
↪ (u, v),
...                                    'elliptic paraboloid')
>>> eparaboloid.normal_vector(normalized=False)
(-2*u, -2*v, 1)
>>> eparaboloid.normal_vector(normalized=True)
(-2*u/sqrt(4*u^2 + 4*v^2 + 1),
 -2*v/sqrt(4*u^2 + 4*v^2 + 1),
 1/sqrt(4*u^2 + 4*v^2 + 1))
```

orthonormal_frame(*coordinates='ext'*)

Return the orthonormal frame field on the surface, expressed either in exterior coordinates (i.e. expressed as vector fields in the ambient space $\mathbb{R}^3$, the default) or interior coordinates (with respect to the natural frame)

INPUT:

- `coordinates` – either `ext` (default) or `int`

OUTPUT: orthogonal frame field as a dictionary

ALGORITHM:

We normalize the first vector $\vec{e}_1$ of the natural frame and then get the second frame vector as $\vec{e}_2 = [\vec{n}, \vec{e}_1]$, where $\vec{n}$ is the unit normal to the surface.

EXAMPLES:

```
sage: u, v = var('u,v', domain='real')
sage: assume(cos(v)>0)
sage: sphere = ParametrizedSurface3D([cos(u)*cos(v), sin(u)*cos(v), sin(v)],
↪[u, v],'sphere')
sage: frame = sphere.orthonormal_frame(); frame
{1: (-sin(u), cos(u), 0), 2: (-cos(u)*sin(v), -sin(u)*sin(v), cos(v))}
sage: (frame[1]*frame[1]).simplify_full()
1
sage: (frame[1]*frame[2]).simplify_full()
0
sage: frame[1] == sphere.orthonormal_frame_vector(1)
True
```

```
>>> from sage.all import *
>>> u, v = var('u,v', domain='real')
>>> assume(cos(v)>Integer(0))
>>> sphere = ParametrizedSurface3D([cos(u)*cos(v), sin(u)*cos(v), sin(v)], [u,
↪ v],'sphere')
>>> frame = sphere.orthonormal_frame(); frame
{1: (-sin(u), cos(u), 0), 2: (-cos(u)*sin(v), -sin(u)*sin(v), cos(v))}
>>> (frame[Integer(1)]*frame[Integer(1)]).simplify_full()
```

```
1
>>> (frame[Integer(1)]*frame[Integer(2)]).simplify_full()
0
>>> frame[Integer(1)] == sphere.orthonormal_frame_vector(Integer(1))
True
```

We compute the orthonormal frame with respect to the natural frame on the surface:

```
sage: frame_int = sphere.orthonormal_frame(coordinates='int'); frame_int
{1: (1/cos(v), 0), 2: (0, 1)}
sage: sphere.first_fundamental_form(frame_int[1], frame_int[1])
1
sage: sphere.first_fundamental_form(frame_int[1], frame_int[2])
0
sage: sphere.first_fundamental_form(frame_int[2], frame_int[2])
1
```

```
>>> from sage.all import *
>>> frame_int = sphere.orthonormal_frame(coordinates='int'); frame_int
{1: (1/cos(v), 0), 2: (0, 1)}
>>> sphere.first_fundamental_form(frame_int[Integer(1)], frame_
 ↪int[Integer(1)])
1
>>> sphere.first_fundamental_form(frame_int[Integer(1)], frame_
 ↪int[Integer(2)])
0
>>> sphere.first_fundamental_form(frame_int[Integer(2)], frame_
 ↪int[Integer(2)])
1
```

orthonormal_frame_vector(*index*, *coordinates='ext'*)

Return a specific basis vector field of the orthonormal frame field on the surface, expressed in exterior or interior coordinates. See `orthogonal_frame()` for more details.

INPUT:

- `index` – index of the basis vector

- `coordinates` – either `ext` (default) or `int`

OUTPUT: orthonormal frame vector field

EXAMPLES:

```
sage: u, v = var('u, v', domain='real')
sage: assume(cos(v)>0)
sage: sphere = ParametrizedSurface3D([cos(u)*cos(v),sin(u)*cos(v),sin(v)],[u,
 ↪v],'sphere')
sage: V1 = sphere.orthonormal_frame_vector(1); V1
(-sin(u), cos(u), 0)
sage: V2 = sphere.orthonormal_frame_vector(2); V2
(-cos(u)*sin(v), -sin(u)*sin(v), cos(v))
sage: (V1*V1).simplify_full()
1
```

```
sage: (V1*V2).simplify_full()
0

sage: n = sphere.normal_vector(normalized=True)
sage: (V1.cross_product(V2) - n).simplify_full()
(0, 0, 0)
```

```
>>> from sage.all import *
>>> u, v = var('u, v', domain='real')
>>> assume(cos(v)>Integer(0))
>>> sphere = ParametrizedSurface3D([cos(u)*cos(v),sin(u)*cos(v),sin(v)],[u,v],
↪'sphere')
>>> V1 = sphere.orthonormal_frame_vector(Integer(1)); V1
(-sin(u), cos(u), 0)
>>> V2 = sphere.orthonormal_frame_vector(Integer(2)); V2
(-cos(u)*sin(v), -sin(u)*sin(v), cos(v))
>>> (V1*V1).simplify_full()
1
>>> (V1*V2).simplify_full()
0

>>> n = sphere.normal_vector(normalized=True)
>>> (V1.cross_product(V2) - n).simplify_full()
(0, 0, 0)
```

parallel_translation_numerical(*curve*, *t*, *v0*, *tinterval*)

Numerically solve the equations for parallel translation of a vector along a curve on the surface. Explicitly, the equations for parallel translation are given by $\frac{du^i}{dt} + u^j \frac{dc^k}{dt} \Gamma^i_{jk} = 0$, where Γ^i_{jk} are the connection coefficients of the surface, the vector to be transported has components u^j and the curve along which to transport has components c^k.

ALGORITHM:

The parallel transport equations are integrated forward in time using the ode solvers from `sage.calculus.ode`. See `_create_pt_ode_system()` for more details.

INPUT:

- `curve` – 2-tuple of functions which determine the curve with respect to the local coordinate system

- `t` – symbolic variable denoting the curve parameter

- `v0` – 2-tuple representing the initial vector

- `tinterval` – list `[a, b, N]`, where `(a, b)` is the domain of the curve and `N` is the number of subdivision points

OUTPUT:

The list consisting of lists `[t, [v1(t), v2(t)]]`, where

- `t` – a subdivision point;

- `[v1(t), v2(t)]` is the list of coordinates of the vector parallel translated along the curve.

EXAMPLES:

```
sage: p, q = var('p,q', domain='real')
sage: v = [p,q]
sage: assume(cos(q)>0)
sage: sphere = ParametrizedSurface3D([cos(q)*cos(p),sin(q)*cos(p),sin(p)],v,
↪'sphere')
sage: s = var('s')
sage: vector_field = sphere.parallel_translation_numerical([s,s],s,[1.0,1.0],
↪[0.0, pi/4, 5])
sage: times, components = zip(*vector_field)

sage: round4 = lambda vec: [N(x, digits=4) for x in vec]   # helper function
↪to round to 4 digits
sage: round4(times)
[0.0000, 0.1571, 0.3142, 0.4712, 0.6283, 0.7854]
sage: [round4(v) for v in components]
[[1.000, 1.000], [0.9876, 1.025], [0.9499, 1.102], [0.8853, 1.238], [0.7920,
↪1.448], [0.6687, 1.762]]
```

```
>>> from sage.all import *
>>> p, q = var('p,q', domain='real')
>>> v = [p,q]
>>> assume(cos(q)>Integer(0))
>>> sphere = ParametrizedSurface3D([cos(q)*cos(p),sin(q)*cos(p),sin(p)],v,
↪'sphere')
>>> s = var('s')
>>> vector_field = sphere.parallel_translation_numerical([s,s],s,[RealNumber(
↪'1.0'),RealNumber('1.0')],[RealNumber('0.0'), pi/Integer(4), Integer(5)])
>>> times, components = zip(*vector_field)

>>> round4 = lambda vec: [N(x, digits=Integer(4)) for x in vec]   # helper
↪function to round to 4 digits
>>> round4(times)
[0.0000, 0.1571, 0.3142, 0.4712, 0.6283, 0.7854]
>>> [round4(v) for v in components]
[[1.000, 1.000], [0.9876, 1.025], [0.9499, 1.102], [0.8853, 1.238], [0.7920,
↪1.448], [0.6687, 1.762]]
```

plot(*urange=None*, *vrange=None*, ***kwds*)

Enable easy plotting directly from the surface class.

The optional keywords urange and vrange specify the range for the surface parameters u and v. If either of these parameters is None, the method checks whether a parameter range was specified when the surface was created. If not, the default of $(0, 2\pi)$ is used.

INPUT:

- urange – 2-tuple specifying the parameter range for u

- vrange – 2-tuple specifying the parameter range for v

EXAMPLES:

```
sage: u, v = var('u, v', domain='real')
sage: eq = (3*u + 3*u*v^2 - u^3, 3*v + 3*u^2*v - v^3, 3*(u^2-v^2))
sage: enneper = ParametrizedSurface3D(eq, (u, v), 'Enneper Surface')
```

(continues on next page)

```
sage: enneper.plot((-5, 5), (-5, 5))                                           #␣
→needs sage.plot
Graphics3d Object
```

```
>>> from sage.all import *
>>> u, v = var('u, v', domain='real')
>>> eq = (Integer(3)*u + Integer(3)*u*v**Integer(2) - u**Integer(3),␣
→Integer(3)*v + Integer(3)*u**Integer(2)*v - v**Integer(3),␣
→Integer(3)*(u**Integer(2)-v**Integer(2)))
>>> enneper = ParametrizedSurface3D(eq, (u, v), 'Enneper Surface')
>>> enneper.plot((-Integer(5), Integer(5)), (-Integer(5), Integer(5)))         ␣
→                                       # needs sage.plot
Graphics3d Object
```

point(*coords*)

> Return a point on the surface given its intrinsic coordinates.
>
> INPUT:
>
> - coords – 2-tuple specifying the intrinsic coordinates (u, v) of the point
>
> OUTPUT: 3-vector specifying the coordinates in $\mathbf{R}^3$ of the point
>
> EXAMPLES:

```
sage: u, v = var('u, v', domain='real')
sage: torus = ParametrizedSurface3D(((2 + cos(u))*cos(v),
....:                                 (2 + cos(u))*sin(v),
....:                                 sin(u)), [u,v], 'torus')
sage: torus.point((0, pi/2))
(0, 3, 0)
sage: torus.point((pi/2, pi))
(-2, 0, 1)
sage: torus.point((pi, pi/2))
(0, 1, 0)
```

```
>>> from sage.all import *
>>> u, v = var('u, v', domain='real')
>>> torus = ParametrizedSurface3D(((Integer(2) + cos(u))*cos(v),
...                                 (Integer(2) + cos(u))*sin(v),
...                                 sin(u)), [u,v], 'torus')
>>> torus.point((Integer(0), pi/Integer(2)))
(0, 3, 0)
>>> torus.point((pi/Integer(2), pi))
(-2, 0, 1)
>>> torus.point((pi, pi/Integer(2)))
(0, 1, 0)
```

principal_directions()

> Finds the principal curvatures and principal directions of the surface.
>
> OUTPUT:
>
> For each principal curvature, returns a list of the form (ρ, V, n), where ρ is the principal curvature, V is the corresponding principal direction, and n is the multiplicity.

EXAMPLES:

```
sage: u, v = var('u, v', domain='real')
sage: R, r = var('R,r', domain='real')
sage: assume(R>r,r>0)
sage: torus = ParametrizedSurface3D([(R+r*cos(v))*cos(u),(R+r*cos(v))*sin(u),
→r*sin(v)],[u,v],'torus')
sage: torus.principal_directions()
[(-cos(v)/(r*cos(v) + R), [(1, 0)], 1), (-1/r, [(0, 1)], 1)]
```

```
>>> from sage.all import *
>>> u, v = var('u, v', domain='real')
>>> R, r = var('R,r', domain='real')
>>> assume(R>r,r>Integer(0))
>>> torus = ParametrizedSurface3D([(R+r*cos(v))*cos(u),(R+r*cos(v))*sin(u),
→r*sin(v)],[u,v],'torus')
>>> torus.principal_directions()
[(-cos(v)/(r*cos(v) + R), [(1, 0)], 1), (-1/r, [(0, 1)], 1)]
```

```
sage: u, v = var('u, v', domain='real')
sage: V = vector([u*cos(u+v), u*sin(u+v), u+v])
sage: helicoid = ParametrizedSurface3D(V, (u, v))
sage: helicoid.principal_directions()
[(-1/(u^2 + 1), [(1, -(u^2 - sqrt(u^2 + 1) + 1)/(u^2 + 1))], 1),
(1/(u^2 + 1), [(1, -(u^2 + sqrt(u^2 + 1) + 1)/(u^2 + 1))], 1)]
```

```
>>> from sage.all import *
>>> u, v = var('u, v', domain='real')
>>> V = vector([u*cos(u+v), u*sin(u+v), u+v])
>>> helicoid = ParametrizedSurface3D(V, (u, v))
>>> helicoid.principal_directions()
[(-1/(u^2 + 1), [(1, -(u^2 - sqrt(u^2 + 1) + 1)/(u^2 + 1))], 1),
(1/(u^2 + 1), [(1, -(u^2 + sqrt(u^2 + 1) + 1)/(u^2 + 1))], 1)]
```

rotation(*theta*)

Give the matrix of the rotation operator over a given angle θ with respect to the natural frame.

INPUT:

- `theta` – rotation angle

OUTPUT: rotation matrix with respect to the natural frame

ALGORITHM:

The operator of rotation over $\pi/2$ is $J_j^i = g^{ik}\omega_{jk}$, where ω is the area form. The operator of rotation over an angle θ is $\cos(\theta)I + \sin(\theta)J$.

EXAMPLES:

```
sage: u, v = var('u, v', domain='real')
sage: assume(cos(v)>0)
sage: sphere = ParametrizedSurface3D([cos(u)*cos(v),sin(u)*cos(v),sin(v)],[u,
→v],'sphere')
```

```
>>> from sage.all import *
>>> u, v = var('u, v', domain='real')
>>> assume(cos(v)>Integer(0))
>>> sphere = ParametrizedSurface3D([cos(u)*cos(v),sin(u)*cos(v),sin(v)],[u,v],
↪'sphere')
```

We first compute the matrix of rotation over $\pi/3$:

```
sage: rotation = sphere.rotation(pi/3); rotation
[              1/2 -1/2*sqrt(3)/cos(v)]
[ 1/2*sqrt(3)*cos(v)                1/2]
```

```
>>> from sage.all import *
>>> rotation = sphere.rotation(pi/Integer(3)); rotation
[              1/2 -1/2*sqrt(3)/cos(v)]
[ 1/2*sqrt(3)*cos(v)                1/2]
```

We verify that three successive rotations over $\pi/3$ yield minus the identity:

```
sage: rotation^3
[-1  0]
[ 0 -1]
```

```
>>> from sage.all import *
>>> rotation**Integer(3)
[-1  0]
[ 0 -1]
```

second_fundamental_form(*vector1*, *vector2*)

Evaluates the second fundamental form on two vectors on the surface. If the vectors are given by $v = (v^1, v^2)$ and $w = (w^1, w^2)$, the result of this function is $h_{11}v^1w^1 + h_{12}(v^1w^2 + v^2w^1) + h_{22}v^2w^2$.

INPUT:

- `vector1`, `vector2` – 2-tuples representing the input vectors

OUTPUT: value of the second fundamental form evaluated on the given vectors

EXAMPLES:

We evaluate the second fundamental form on two symbolic vectors:

```
sage: u, v = var('u, v', domain='real')
sage: v1, v2, w1, w2 = var('v1, v2, w1, w2', domain='real')
sage: assume(cos(v) > 0)
sage: sphere = ParametrizedSurface3D([cos(u)*cos(v),sin(u)*cos(v),sin(v)],[u,
↪v],'sphere')
sage: sphere.second_fundamental_form(vector([v1, v2]), vector([w1, w2]))
-v1*w1*cos(v)^2 - v2*w2
```

```
>>> from sage.all import *
>>> u, v = var('u, v', domain='real')
>>> v1, v2, w1, w2 = var('v1, v2, w1, w2', domain='real')
>>> assume(cos(v) > Integer(0))
>>> sphere = ParametrizedSurface3D([cos(u)*cos(v),sin(u)*cos(v),sin(v)],[u,v],
```

```
↪'sphere')
>>> sphere.second_fundamental_form(vector([v1, v2]), vector([w1, w2]))
-v1*w1*cos(v)^2 - v2*w2
```

We evaluate the second fundamental form on vectors with numerical components:

```
sage: vect = vector([1,2])
sage: sphere.second_fundamental_form(vect, vect)
-cos(v)^2 - 4
sage: sphere.second_fundamental_form([1,1], [2,1])
-2*cos(v)^2 - 1
```

```
>>> from sage.all import *
>>> vect = vector([Integer(1),Integer(2)])
>>> sphere.second_fundamental_form(vect, vect)
-cos(v)^2 - 4
>>> sphere.second_fundamental_form([Integer(1),Integer(1)], [Integer(2),
↪Integer(1)])
-2*cos(v)^2 - 1
```

second_fundamental_form_coefficient(*index*)

Return the coefficient h_{ij} of the second fundamental form corresponding to the index (i, j). If the equation of the surface is $\vec{r}(u^1, u^2)$, then $h_{ij} = \vec{r}_{u^i u^j} \cdot \vec{n}$, where $\vec{n}$ is the unit normal.

INPUT:

- index – a 2-tuple (i, j)

OUTPUT: component h_{ij} of the second fundamental form

EXAMPLES:

```
sage: u, v = var('u,v', domain='real')
sage: assume(cos(v)>0)
sage: sphere = ParametrizedSurface3D([cos(u)*cos(v),sin(u)*cos(v),sin(v)],[u,
↪v],'sphere')
sage: sphere.second_fundamental_form_coefficient((1, 1))
-cos(v)^2
sage: sphere.second_fundamental_form_coefficient((2, 1))
0
```

```
>>> from sage.all import *
>>> u, v = var('u,v', domain='real')
>>> assume(cos(v)>Integer(0))
>>> sphere = ParametrizedSurface3D([cos(u)*cos(v),sin(u)*cos(v),sin(v)],[u,v],
↪'sphere')
>>> sphere.second_fundamental_form_coefficient((Integer(1), Integer(1)))
-cos(v)^2
>>> sphere.second_fundamental_form_coefficient((Integer(2), Integer(1)))
0
```

second_fundamental_form_coefficients()

Return the coefficients h_{ij} of the second fundamental form as a dictionary, where the keys are the indices (i, j) and the values are the corresponding components h_{ij}.

When only one component is needed, consider instead the function *second_fundamental_form_coefficient()*.

OUTPUT: dictionary of second fundamental form coefficients

EXAMPLES:

```
sage: u, v = var('u, v', domain='real')
sage: assume(cos(v)>0)
sage: sphere = ParametrizedSurface3D([cos(u)*cos(v),sin(u)*cos(v),sin(v)],[u,
↪v],'sphere')
sage: sphere.second_fundamental_form_coefficients()
{(1, 1): -cos(v)^2, (1, 2): 0, (2, 1): 0, (2, 2): -1}
```

```
>>> from sage.all import *
>>> u, v = var('u, v', domain='real')
>>> assume(cos(v)>Integer(0))
>>> sphere = ParametrizedSurface3D([cos(u)*cos(v),sin(u)*cos(v),sin(v)],[u,v],
↪'sphere')
>>> sphere.second_fundamental_form_coefficients()
{(1, 1): -cos(v)^2, (1, 2): 0, (2, 1): 0, (2, 2): -1}
```

second_order_natural_frame()

Return the second-order frame of the surface, i.e. computes the second-order derivatives (with respect to the parameters on the surface) of the parametric expression $\vec{r} = \vec{r}(u^1, u^2)$ of the surface.

OUTPUT:

- Dictionary where the keys are 2-tuples (i, j) and the values are the corresponding derivatives r_{ij}.

EXAMPLES:

We compute the second-order natural frame of the sphere:

```
sage: u, v = var('u, v', domain='real')
sage: sphere = ParametrizedSurface3D([cos(u)*cos(v),sin(u)*cos(v),sin(v)],[u,
↪v],'sphere')
sage: sphere.second_order_natural_frame()
{(1, 1): (-cos(u)*cos(v), -cos(v)*sin(u), 0),
 (1, 2): (sin(u)*sin(v), -cos(u)*sin(v), 0),
 (2, 1): (sin(u)*sin(v), -cos(u)*sin(v), 0),
 (2, 2): (-cos(u)*cos(v), -cos(v)*sin(u), -sin(v))}
```

```
>>> from sage.all import *
>>> u, v = var('u, v', domain='real')
>>> sphere = ParametrizedSurface3D([cos(u)*cos(v),sin(u)*cos(v),sin(v)],[u,v],
↪'sphere')
>>> sphere.second_order_natural_frame()
{(1, 1): (-cos(u)*cos(v), -cos(v)*sin(u), 0),
 (1, 2): (sin(u)*sin(v), -cos(u)*sin(v), 0),
 (2, 1): (sin(u)*sin(v), -cos(u)*sin(v), 0),
 (2, 2): (-cos(u)*cos(v), -cos(v)*sin(u), -sin(v))}
```

second_order_natural_frame_element(*index*)

Return a vector in the second-order frame of the surface, i.e. computes the second-order derivatives of the parametric expression $\vec{r}$ of the surface with respect to the parameters listed in the argument.

INPUT:

- `index` – a 2-tuple (`i, j`) specifying the element of the second-order frame

OUTPUT: the second-order derivative r_{ij}

EXAMPLES:

```
sage: u, v = var('u, v', domain='real')
sage: sphere = ParametrizedSurface3D([cos(u)*cos(v),sin(u)*cos(v),sin(v)],[u,
→v],'sphere')
sage: sphere.second_order_natural_frame_element((1, 2))
(sin(u)*sin(v), -cos(u)*sin(v), 0)
```

```
>>> from sage.all import *
>>> u, v = var('u, v', domain='real')
>>> sphere = ParametrizedSurface3D([cos(u)*cos(v),sin(u)*cos(v),sin(v)],[u,v],
→'sphere')
>>> sphere.second_order_natural_frame_element((Integer(1), Integer(2)))
(sin(u)*sin(v), -cos(u)*sin(v), 0)
```

shape_operator()

Return the shape operator of the surface as a matrix. The shape operator is defined as the derivative of the Gauss map, and is computed here in terms of the first and second fundamental form by means of the Weingarten equations.

OUTPUT: matrix of the shape operator

EXAMPLES:

```
sage: R = var('R')
sage: assume(R>0)
sage: u, v = var('u,v', domain='real')
sage: assume(cos(v)>0)
sage: sphere = ParametrizedSurface3D([R*cos(u)*cos(v),R*sin(u)*cos(v),
→R*sin(v)],[u,v],'sphere')
sage: S = sphere.shape_operator(); S
[-1/R    0]
[   0 -1/R]
```

```
>>> from sage.all import *
>>> R = var('R')
>>> assume(R>Integer(0))
>>> u, v = var('u,v', domain='real')
>>> assume(cos(v)>Integer(0))
>>> sphere = ParametrizedSurface3D([R*cos(u)*cos(v),R*sin(u)*cos(v),R*sin(v)],
→[u,v],'sphere')
>>> S = sphere.shape_operator(); S
[-1/R    0]
[   0 -1/R]
```

The eigenvalues of the shape operator are the principal curvatures of the surface:

```
sage: u, v = var('u,v', domain='real')
sage: paraboloid = ParametrizedSurface3D([u, v, u^2+v^2], [u, v], 'paraboloid
```

```
↪')
sage: S = paraboloid.shape_operator(); S
[2*(4*v^2 + 1)/(4*u^2 + 4*v^2 + 1)^(3/2)           -8*u*v/(4*u^2 + 4*v^2 + 1)^(3/
↪2)]
[         -8*u*v/(4*u^2 + 4*v^2 + 1)^(3/2) 2*(4*u^2 + 1)/(4*u^2 + 4*v^2 + 1)^(3/
↪2)]
sage: S.eigenvalues()
[2*sqrt(4*u^2 + 4*v^2 + 1)/(16*u^4 + 16*v^4 + 8*(4*u^2 + 1)*v^2 + 8*u^2 + 1),␣
↪2/sqrt(4*u^2 + 4*v^2 + 1)]
```

```
>>> from sage.all import *
>>> u, v = var('u,v', domain='real')
>>> paraboloid = ParametrizedSurface3D([u, v, u**Integer(2)+v**Integer(2)],␣
↪[u, v], 'paraboloid')
>>> S = paraboloid.shape_operator(); S
[2*(4*v^2 + 1)/(4*u^2 + 4*v^2 + 1)^(3/2)           -8*u*v/(4*u^2 + 4*v^2 + 1)^(3/
↪2)]
[         -8*u*v/(4*u^2 + 4*v^2 + 1)^(3/2) 2*(4*u^2 + 1)/(4*u^2 + 4*v^2 + 1)^(3/
↪2)]
>>> S.eigenvalues()
[2*sqrt(4*u^2 + 4*v^2 + 1)/(16*u^4 + 16*v^4 + 8*(4*u^2 + 1)*v^2 + 8*u^2 + 1),␣
↪2/sqrt(4*u^2 + 4*v^2 + 1)]
```

shape_operator_coefficients()

Return the components of the shape operator of the surface as a dictionary. See `shape_operator` for more information.

OUTPUT:

Dictionary where the keys are two-tuples (i, j), with values the corresponding component of the shape operator.

EXAMPLES:

```
sage: R = var('R')
sage: u, v = var('u,v', domain='real')
sage: assume(cos(v)>0)
sage: sphere = ParametrizedSurface3D([R*cos(u)*cos(v),R*sin(u)*cos(v),
↪R*sin(v)],[u,v],'sphere')
sage: sphere.shape_operator_coefficients()
{(1, 1): -1/R, (1, 2): 0, (2, 1): 0, (2, 2): -1/R}
```

```
>>> from sage.all import *
>>> R = var('R')
>>> u, v = var('u,v', domain='real')
>>> assume(cos(v)>Integer(0))
>>> sphere = ParametrizedSurface3D([R*cos(u)*cos(v),R*sin(u)*cos(v),R*sin(v)],
↪[u,v],'sphere')
>>> sphere.shape_operator_coefficients()
{(1, 1): -1/R, (1, 2): 0, (2, 1): 0, (2, 2): -1/R}
```

tangent_vector(*coords*, *components*)

Return the components of a tangent vector given the intrinsic coordinates of the base point and the components of the vector in the intrinsic frame.

INPUT:

- `coords` – 2-tuple specifying the intrinsic coordinates (`u, v`) of the point
- `components` – 2-tuple specifying the components of the tangent vector in the intrinsic coordinate frame

OUTPUT: 3-vector specifying the components in $\mathbf{R}^3$ of the vector

EXAMPLES:

We compute two tangent vectors to Enneper's surface along the coordinate lines and check that their cross product gives the normal vector:

```
sage: u, v = var('u,v', domain='real')
sage: eq = (3*u + 3*u*v^2 - u^3, 3*v + 3*u^2*v - v^3, 3*(u^2-v^2))
sage: e = ParametrizedSurface3D(eq, (u, v), 'Enneper Surface')

sage: w1 = e.tangent_vector((1, 2), (1, 0)); w1
(12, 12, 6)
sage: w2 = e.tangent_vector((1, 2), (0, 1)); w2
(12, -6, -12)
sage: w1.cross_product(w2)
(-108, 216, -216)

sage: n = e.normal_vector().subs({u: 1, v: 2}); n
(-108, 216, -216)
sage: n == w1.cross_product(w2)
True
```

```
>>> from sage.all import *
>>> u, v = var('u,v', domain='real')
>>> eq = (Integer(3)*u + Integer(3)*u*v**Integer(2) - u**Integer(3),␣
→Integer(3)*v + Integer(3)*u**Integer(2)*v - v**Integer(3),␣
→Integer(3)*(u**Integer(2)-v**Integer(2)))
>>> e = ParametrizedSurface3D(eq, (u, v), 'Enneper Surface')

>>> w1 = e.tangent_vector((Integer(1), Integer(2)), (Integer(1), Integer(0)));
→ w1
(12, 12, 6)
>>> w2 = e.tangent_vector((Integer(1), Integer(2)), (Integer(0), Integer(1)));
→ w2
(12, -6, -12)
>>> w1.cross_product(w2)
(-108, 216, -216)

>>> n = e.normal_vector().subs({u: Integer(1), v: Integer(2)}); n
(-108, 216, -216)
>>> n == w1.cross_product(w2)
True
```

COMMON PARAMETRIZED SURFACES IN 3D.

AUTHORS:

```
- Joris Vankerschaver (2012-06-16)
```

class sage.geometry.riemannian_manifolds.surface3d_generators.**SurfaceGenerators**

Bases: object

A class consisting of generators for several common parametrized surfaces in 3D.

static Catenoid(*c=1*, *name='Catenoid'*)

Return a catenoid surface, with parametric representation.

$$
\begin{aligned}
x(u,v) &= c\cosh(v/c)\cos(u); \\
y(u,v) &= c\cosh(v/c)\sin(u); \\
z(u,v) &= v.
\end{aligned}
$$

INPUT:

- c – surface parameter

- name – string; name of the surface

For more information, see Wikipedia article Catenoid.

EXAMPLES:

```
sage: cat = surfaces.Catenoid(); cat
Parametrized surface ('Catenoid') with equation (cos(u)*cosh(v),␣
↪cosh(v)*sin(u), v)
sage: cat.plot()                                                    #␣
↪needs sage.plot
Graphics3d Object
```

```
>>> from sage.all import *
>>> cat = surfaces.Catenoid(); cat
Parametrized surface ('Catenoid') with equation (cos(u)*cosh(v),␣
↪cosh(v)*sin(u), v)
>>> cat.plot()                                                      #␣
↪needs sage.plot
Graphics3d Object
```

static Crosscap(*r=1*, *name='Crosscap'*)

    Return a crosscap surface, with parametrization.

$$x(u,v) = r(1 + \cos(v))\cos(u);$$
$$y(u,v) = r(1 + \cos(v))\sin(u);$$
$$z(u,v) = -r\tanh(u - \pi)\sin(v).$$

    INPUT:

- `r` – surface parameter

- `name` – string; name of the surface

    For more information, see Wikipedia article Cross-cap.

    EXAMPLES:

```
sage: crosscap = surfaces.Crosscap(); crosscap
Parametrized surface ('Crosscap') with equation ((cos(v) + 1)*cos(u), (cos(v)⌑
↪+ 1)*sin(u), -sin(v)*tanh(-pi + u))
sage: crosscap.plot()                                                      #⌑
↪needs sage.plot
Graphics3d Object
```

```
>>> from sage.all import *
>>> crosscap = surfaces.Crosscap(); crosscap
Parametrized surface ('Crosscap') with equation ((cos(v) + 1)*cos(u), (cos(v)⌑
↪+ 1)*sin(u), -sin(v)*tanh(-pi + u))
>>> crosscap.plot()                                                        #⌑
↪needs sage.plot
Graphics3d Object
```

static Dini(*a=1*, *b=1*, *name="Dini's surface"*)

    Return Dini's surface, with parametrization.

$$x(u,v) = a\cos(u)\sin(v);$$
$$y(u,v) = a\sin(u)\sin(v);$$
$$z(u,v) = u + \log(\tan(v/2)) + \cos(v).$$

    INPUT:

- `a`, `b` – surface parameters

- `name` – string; name of the surface

    For more information, see Wikipedia article Dini%27s_surface.

    EXAMPLES:

```
sage: dini = surfaces.Dini(a=3, b=4); dini
Parametrized surface ('Dini's surface') with equation (3*cos(u)*sin(v),⌑
↪3*sin(u)*sin(v), 4*u + 3*cos(v) + 3*log(tan(1/2*v)))
sage: dini.plot()                                                          #⌑
↪needs sage.plot
Graphics3d Object
```

```
>>> from sage.all import *
>>> dini = surfaces.Dini(a=Integer(3), b=Integer(4)); dini
Parametrized surface ('Dini's surface') with equation (3*cos(u)*sin(v),␣
↪3*sin(u)*sin(v), 4*u + 3*cos(v) + 3*log(tan(1/2*v)))
>>> dini.plot()                                                          #␣
↪needs sage.plot
Graphics3d Object
```

static Ellipsoid(*center=(0, 0, 0)*, *axes=(1, 1, 1)*, *name='Ellipsoid'*)

Return an ellipsoid centered at `center` whose semi-principal axes have lengths given by the components of `axes`. The parametrization of the ellipsoid is given by

$$x(u, v) = x_0 + a\cos(u)\cos(v);$$
$$y(u, v) = y_0 + b\sin(u)\cos(v);$$
$$z(u, v) = z_0 + c\sin(v).$$

INPUT:

- `center` – 3-tuple; coordinates of the center of the ellipsoid

- `axes` – 3-tuple; lengths of the semi-principal axes

- `name` – string; name of the ellipsoid

For more information, see Wikipedia article Ellipsoid.

EXAMPLES:

```
sage: ell = surfaces.Ellipsoid(axes=(1, 2, 3)); ell
Parametrized surface ('Ellipsoid') with equation (cos(u)*cos(v),␣
↪2*cos(v)*sin(u), 3*sin(v))
sage: ell.plot()                                                        #␣
↪needs sage.plot
Graphics3d Object
```

```
>>> from sage.all import *
>>> ell = surfaces.Ellipsoid(axes=(Integer(1), Integer(2), Integer(3))); ell
Parametrized surface ('Ellipsoid') with equation (cos(u)*cos(v),␣
↪2*cos(v)*sin(u), 3*sin(v))
>>> ell.plot()                                                          #␣
↪needs sage.plot
Graphics3d Object
```

static Enneper(*name="Enneper's surface"*)

Return Enneper's surface, with parametrization.

$$x(u, v) = u(1 - u^2/3 + v^2)/3;$$
$$y(u, v) = -v(1 - v^2/3 + u^2)/3;$$
$$z(u, v) = (u^2 - v^2)/3.$$

INPUT:

- `name` – string; name of the surface

For more information, see Wikipedia article Enneper_surface.

EXAMPLES:

```
sage: enn = surfaces.Enneper(); enn
Parametrized surface ('Enneper's surface') with equation (-1/9*(u^2 - 3*v^2 -␣
→3)*u, -1/9*(3*u^2 - v^2 + 3)*v, 1/3*u^2 - 1/3*v^2)
sage: enn.plot()                                                               #␣
→needs sage.plot
Graphics3d Object
```

```
>>> from sage.all import *
>>> enn = surfaces.Enneper(); enn
Parametrized surface ('Enneper's surface') with equation (-1/9*(u^2 - 3*v^2 -␣
→3)*u, -1/9*(3*u^2 - v^2 + 3)*v, 1/3*u^2 - 1/3*v^2)
>>> enn.plot()                                                                 #␣
→needs sage.plot
Graphics3d Object
```

static Helicoid(*h=1*, *name='Helicoid'*)

Return a helicoid surface, with parametrization.

$$x(\rho, \theta) = \rho \cos(\theta);$$
$$y(\rho, \theta) = \rho \sin(\theta);$$
$$z(\rho, \theta) = h\theta/(2\pi).$$

INPUT:

- h – distance along the z-axis between two successive turns of the helicoid

- name – string; name of the surface

For more information, see Wikipedia article Helicoid.

EXAMPLES:

```
sage: helicoid = surfaces.Helicoid(h=2); helicoid
Parametrized surface ('Helicoid') with equation (rho*cos(theta),␣
→rho*sin(theta), theta/pi)
sage: helicoid.plot()                                                          #␣
→needs sage.plot
Graphics3d Object
```

```
>>> from sage.all import *
>>> helicoid = surfaces.Helicoid(h=Integer(2)); helicoid
Parametrized surface ('Helicoid') with equation (rho*cos(theta),␣
→rho*sin(theta), theta/pi)
>>> helicoid.plot()                                                            #␣
→needs sage.plot
Graphics3d Object
```

static Klein(*r=1*, *name='Klein bottle'*)

Return the Klein bottle, in the figure-8 parametrization given by

$$x(u, v) = (r + \cos(u/2)\cos(v) - \sin(u/2)\sin(2v))\cos(u);$$
$$y(u, v) = (r + \cos(u/2)\cos(v) - \sin(u/2)\sin(2v))\sin(u);$$
$$z(u, v) = \sin(u/2)\cos(v) + \cos(u/2)\sin(2v).$$

INPUT:

- `r` – radius of the "figure-8" circle

- `name` – string; name of the surface

For more information, see Wikipedia article Klein_bottle.

EXAMPLES:

```
sage: klein = surfaces.Klein(); klein
Parametrized surface ('Klein bottle') with equation (-(sin(1/2*u)*sin(2*v) -
→cos(1/2*u)*sin(v) - 1)*cos(u), -(sin(1/2*u)*sin(2*v) - cos(1/2*u)*sin(v) -
→1)*sin(u), cos(1/2*u)*sin(2*v) + sin(1/2*u)*sin(v))
sage: klein.plot()                                                        #
→needs sage.plot
Graphics3d Object
```

```
>>> from sage.all import *
>>> klein = surfaces.Klein(); klein
Parametrized surface ('Klein bottle') with equation (-(sin(1/2*u)*sin(2*v) -
→cos(1/2*u)*sin(v) - 1)*cos(u), -(sin(1/2*u)*sin(2*v) - cos(1/2*u)*sin(v) -
→1)*sin(u), cos(1/2*u)*sin(2*v) + sin(1/2*u)*sin(v))
>>> klein.plot()                                                          #
→needs sage.plot
Graphics3d Object
```

static **MonkeySaddle**(*name='Monkey saddle'*)

Return a monkey saddle surface, with equation

$$z = x^3 - 3xy^2.$$

INPUT:

- `name` – string; name of the surface

For more information, see Wikipedia article Monkey_saddle.

EXAMPLES:

```
sage: saddle = surfaces.MonkeySaddle(); saddle
Parametrized surface ('Monkey saddle') with equation (u, v, u^3 - 3*u*v^2)
sage: saddle.plot()                                                       #
→needs sage.plot
Graphics3d Object
```

```
>>> from sage.all import *
>>> saddle = surfaces.MonkeySaddle(); saddle
Parametrized surface ('Monkey saddle') with equation (u, v, u^3 - 3*u*v^2)
>>> saddle.plot()                                                         #
→needs sage.plot
Graphics3d Object
```

static **Paraboloid**(*a=1*, *b=1*, *c=1*, *elliptic=True*, *name=None*)

Return a paraboloid with equation.

$$\frac{z}{c} = \pm\frac{x^2}{a^2} + \frac{y^2}{b^2}$$

When the plus sign is selected, the paraboloid is elliptic. Otherwise the surface is a hyperbolic paraboloid.

INPUT:

- a, b, c – surface parameters

- elliptic – boolean (default: True); whether to create an elliptic or hyperbolic paraboloid

- name – string; name of the surface

For more information, see Wikipedia article Paraboloid.

EXAMPLES:

```
sage: epar = surfaces.Paraboloid(1, 3, 2); epar
Parametrized surface ('Elliptic paraboloid') with equation (u, v, 2*u^2 + 2/
↪9*v^2)
sage: epar.plot()                                                          #␣
↪needs sage.plot
Graphics3d Object

sage: hpar = surfaces.Paraboloid(2, 3, 1, elliptic=False); hpar
Parametrized surface ('Hyperbolic paraboloid') with equation (u, v, -1/4*u^2␣
↪+ 1/9*v^2)
sage: hpar.plot()                                                          #␣
↪needs sage.plot
Graphics3d Object
```

```
>>> from sage.all import *
>>> epar = surfaces.Paraboloid(Integer(1), Integer(3), Integer(2)); epar
Parametrized surface ('Elliptic paraboloid') with equation (u, v, 2*u^2 + 2/
↪9*v^2)
>>> epar.plot()                                                            #␣
↪needs sage.plot
Graphics3d Object

>>> hpar = surfaces.Paraboloid(Integer(2), Integer(3), Integer(1),␣
↪elliptic=False); hpar
Parametrized surface ('Hyperbolic paraboloid') with equation (u, v, -1/4*u^2␣
↪+ 1/9*v^2)
>>> hpar.plot()                                                            #␣
↪needs sage.plot
Graphics3d Object
```

static Sphere(*center=(0, 0, 0)*, *R=1*, *name='Sphere'*)

Return a sphere of radius R centered at center.

INPUT:

- center – 3-tuple; center of the sphere

- R – radius of the sphere

- name – string; name of the surface

For more information, see Wikipedia article Sphere.

EXAMPLES:

```
sage: sphere = surfaces.Sphere(center=(0, 1, -1), R=2); sphere
Parametrized surface ('Sphere') with equation (2*cos(u)*cos(v),␣
→2*cos(v)*sin(u) + 1, 2*sin(v) - 1)
sage: sphere.plot()                                                        #␣
→needs sage.plot
Graphics3d Object
```

```
>>> from sage.all import *
>>> sphere = surfaces.Sphere(center=(Integer(0), Integer(1), -Integer(1)),␣
→R=Integer(2)); sphere
Parametrized surface ('Sphere') with equation (2*cos(u)*cos(v),␣
→2*cos(v)*sin(u) + 1, 2*sin(v) - 1)
>>> sphere.plot()                                                          #␣
→needs sage.plot
Graphics3d Object
```

Note that the radius of the sphere can be negative. The surface thus obtained is equal to the sphere (or part thereof) with positive radius, whose coordinate functions have been multiplied by -1. Compare for instant the first octant of the unit sphere with positive radius:

```
sage: octant1 = surfaces.Sphere(R=1); octant1
Parametrized surface ('Sphere') with equation (cos(u)*cos(v), cos(v)*sin(u),␣
→sin(v))
sage: octant1.plot((0, pi/2), (0, pi/2))                                   #␣
→needs sage.plot
Graphics3d Object
```

```
>>> from sage.all import *
>>> octant1 = surfaces.Sphere(R=Integer(1)); octant1
Parametrized surface ('Sphere') with equation (cos(u)*cos(v), cos(v)*sin(u),␣
→sin(v))
>>> octant1.plot((Integer(0), pi/Integer(2)), (Integer(0), pi/Integer(2)))   ␣
→                                    # needs sage.plot
Graphics3d Object
```

with the first octant of the unit sphere with negative radius:

```
sage: octant2 = surfaces.Sphere(R=-1); octant2
Parametrized surface ('Sphere') with equation (-cos(u)*cos(v), -cos(v)*sin(u),
→ -sin(v))
sage: octant2.plot((0, pi/2), (0, pi/2))                                   #␣
→needs sage.plot
Graphics3d Object
```

```
>>> from sage.all import *
>>> octant2 = surfaces.Sphere(R=-Integer(1)); octant2
Parametrized surface ('Sphere') with equation (-cos(u)*cos(v), -cos(v)*sin(u),
→ -sin(v))
>>> octant2.plot((Integer(0), pi/Integer(2)), (Integer(0), pi/Integer(2)))   ␣
→                                    # needs sage.plot
Graphics3d Object
```

static Torus(*r=2*, *R=3*, *name='Torus'*)

Return a torus obtained by revolving a circle of radius `r` around a coplanar axis `R` units away from the center of the circle. The parametrization used is

$$x(u,v) = (R + r\cos(v))\cos(u);$$
$$y(u,v) = (R + r\cos(v))\sin(u);$$
$$z(u,v) = r\sin(v).$$

INPUT:

- `r`, `R` – minor and major radius of the torus

- `name` – string; name of the surface

For more information, see Wikipedia article Torus.

EXAMPLES:

```
sage: torus = surfaces.Torus(); torus
Parametrized surface ('Torus') with equation ((2*cos(v) + 3)*cos(u),
→(2*cos(v) + 3)*sin(u), 2*sin(v))
sage: torus.plot()                                                    #
→needs sage.plot
Graphics3d Object
```

```
>>> from sage.all import *
>>> torus = surfaces.Torus(); torus
Parametrized surface ('Torus') with equation ((2*cos(v) + 3)*cos(u),
→(2*cos(v) + 3)*sin(u), 2*sin(v))
>>> torus.plot()                                                      #
→needs sage.plot
Graphics3d Object
```

static **WhitneyUmbrella**(*name="Whitney's umbrella"*)

Return Whitney's umbrella, with parametric representation

$$x(u,v) = uv, \quad y(u,v) = u, \quad z(u,v) = v^2.$$

INPUT:

- `name` – string; name of the surface

For more information, see Wikipedia article Whitney_umbrella.

EXAMPLES:

```
sage: whitney = surfaces.WhitneyUmbrella(); whitney
Parametrized surface ('Whitney's umbrella') with equation (u*v, u, v^2)
sage: whitney.plot()                                                  #
→needs sage.plot
Graphics3d Object
```

```
>>> from sage.all import *
>>> whitney = surfaces.WhitneyUmbrella(); whitney
Parametrized surface ('Whitney's umbrella') with equation (u*v, u, v^2)
>>> whitney.plot()                                                    #
→needs sage.plot
Graphics3d Object
```

INDICES AND TABLES

- Index
- Module Index
- Search Page

PYTHON MODULE INDEX

g

INDEX