geometer Documentation

Release 0.2.0

Jan Müller

May 27, 2019
## Contents:

1 Installation 3
   1.1 Quickstart .................................................. 3
   1.2 API Reference ............................................... 5
   1.3 Changelog .................................................... 24

2 Indices and tables 25
   2.1 References .................................................. 25

Python Module Index 27
Geometer is a geometry library for Python 3 that uses projective geometry and numpy for fast geometric computation. In projective geometry every point in 2D is represented by a three-dimensional vector and every point in 3D is represented by a four-dimensional vector. This has the following advantages:

- There are points at infinity that can be treated just like normal points.
- Projective transformations are described by matrices but they can also represent affine transformations i.e. also translations.
- Every two lines have a unique point of intersection if they lie in the same plane. Parallel lines have a point of intersection at infinity.
- Points of intersection, planes or lines through given points can be calculated using simple cross products or tensor diagrams.
- Special complex points at infinity and cross ratios can be used to calculate angles and to construct perpendicular geometric structures.

Most of the computation in the library done via tensor diagrams (using numpy.einsum).

The source code of the package can be found on GitHub and the documentation on Read the Docs.
You can install the package directly from PyPI:

```
pip install geometer
```

# 1.1 Quickstart

## 1.1.1 Geometry in two dimensions

The elementary objects of projective geometry, points and lines can be created using the `Point` and `Line` classes without having to specify the dimension:

```
from geometer import *

p = Point(2, 4)
q = Point(3, 5)
l = Line(p, q)
m = Line(0, 1, 0)
```

Here we specified a line once by two base points and once using the homogeneous coordinates of the line. The two elementary operations `meet` and `join` can also be called called exactly as one would expect:

```
l = join(p, q)
r = meet(l, m)
```

Geometer can also construct parallel and perpendicular lines:

```
m = l.parallel(through=Point(1, 1))
n = l.perpendicular(through=Point(1, 1))
is_perpendicular(m, n)
```

The function `is_perpendicular()` returns `True` when two lines are perpendicular. Other angles an distances can also be calculated:
a = angle(1, Point(1, 0))
dist(l, p)

import numpy as np
p + 2*dist(p, q)*Point(np.cos(a), np.sin(a))

Projective transformations can be easily created using the methods rotation and translation or by supplying a matrix to the Transformation class:

t1 = translation(0, -1)
t2 = rotation(-np.pi)
t3 = Transformation([[0, 1, 0],
                      [1, 0, 0],
                      [0, 0, 1]])
t1*t2*p

Geometer also includes tools to work with conics. They can be created using the classes Conic or Circle:

a = Point(-1, 0)
b = Point(0, 3)
c = Point(1, 2)
d = Point(2, 1)
e = Point(0, -1)
conic = Conic.from_points(a, b, c, d, e)

To calculate cross ratios of points or lines, the function crossratio() can be used:

t = rotation(np.pi/16)
crossratio(q, t*q, t**2 * q, t**3 * q, p)

Other interesting operators are harmonic_set(), angle_bisectors(), is_cocircular() and is_collinear().

1.1.2 Geometry in three dimensions

Creating points and lines in 3D works the same as in the two dimensional case:

p1 = Point(1, 1, 0)
p2 = Point(2, 1, 0)
p3 = Point(3, 4, 0)
l = Line(p1, p2)

In addition to points and lines, in three dimensions we can use the Plane class or the join operation to create planes:

A = join(l, p3)
A.project(Point(3, 4, 5))

Points can be projected onto planes and lines. The result is still a point in 3D but now lying on the plane.

The point of intersection of a plane and a line can be calculated with the meet operation:

l = Line(Point(1, 2, 3), Point(3, 4, 5))
A.meet(l)

All other operations such as angles, distances, perpendicular lines, cross ratios are also compatible with objects in 3D. For information on their usage, you can look at the two dimensional example.
1.2 API Reference

1.2.1 geometer package

geometer.utils package

geometer.utils.math.

null_space

(\( A \))

Constructs an orthonormal basis for the null space of a \( A \) using SVD.

Parameters

\( A \) (array_like) – The input matrix.

Returns

Orthonormal basis for the null space of \( A \) (as column vectors in the returned matrix).

Return type

numpy.ndarray

geometer.utils.polynomial.

np_array_to_poly

(\( c \), symbols)

Converts an array of coefficients into a sympy polynomial.

If \( c \) an array of shape \((m, \ldots, m)\) and variables \( x_1, \ldots, x_n \) are given as \( \text{symbols} \), the resulting polynomial will be given by

\[
\sum_{0 \leq i_1, \ldots, i_n \leq m-1} c[i_1, \ldots, i_n] x_1^{i_1} \cdots x_n^{i_n}.
\]

Parameters

• \( c \) (array_like) – Array of coefficients where the different axis correspond to different variables with the degree in each axis given by the corresponding index.

• \( \text{symbols} \) (list of sympy.Symbol) – The symbols to be used as variables of the polynomial.

Returns

The resulting polynomial as described above.

Return type

sympy.Poly

geometer.utils.polynomial.

poly_to_np_array

(\( p \), symbols)

Convert a sympy polynomial to an array of coefficients as required by numpy.

If \( p \) is a multivariate polynomial with the variables \( x_1, \ldots, x_n \) specified in \( \text{symbols} \), the resulting array will be such that

\[
p = \sum_{0 \leq i_1, \ldots, i_n \leq m-1} c[i_1, \ldots, i_n] x_1^{i_1} \cdots x_n^{i_n}.
\]

Parameters

• \( \text{p} \) (sympy.Expr) – The sympy expression that represents the polynomial.

• \( \text{symbols} \) (list of sympy.Symbol) – The variables used in the polynomial expression.

Returns

The coefficients of the polynomial as described above.

Return type

numpy.ndarray

geometer.utils.polynomial.

polyval

(x, c)

Evaluate a multivariate polynomial at a specified point.

If \( x \) is an array of length \( n \) and \( c \) an array of shape \((m, \ldots, m)\), the result is

\[
\sum_{0 \leq i_1, \ldots, i_n \leq m-1} c[i_1, \ldots, i_n] x[1]^{i_1} \cdots x[n]^{i_n}.
\]
Parameters

- **x** (*array_like*) – The point to evaluate the polynomial at.
- **c** (*array_like*) – Array of coefficients where the different axis correspond to different variables with the degree in each axis given by the corresponding index.

**Returns** The result of the evaluation as described above.

**Return type** float or complex

### Modules

**geometer.base module**

**class** `geometer.base.KroneckerDelta(n, p=1)`

Bases: `geometer.base.Tensor`

This class can be used to construct a (p, p)-tensor representing the Kronecker delta tensor.

**Parameters**

- **n** (*int*) – The dimension of the tensor.
- **p** (*int*, optional) – The number of covariant and contravariant indices of the tensor, default is 1.

**References**

**class** `geometer.base.LeviCivitaTensor(size, covariant=True)`

Bases: `geometer.base.Tensor`

This class can be used to construct a tensor representing the Levi-Civita symbol.

**Parameters**

- **size** (*int*) – The number of indices of the tensor.
- **covariant** (*bool*, optional) – If true, the tensor will only have covariant indices. Default: True

**References**

**class** `geometer.base.ProjectiveElement(*args, covariant=True)`

Bases: `geometer.base.Tensor, abc.ABC`

Base class for all projective tensors, i.e. all objects that identify scalar multiples.

**dim**

The dimension of the tensor.

**Type** int

**class** `geometer.base.Tensor(*args, covariant=True)`

Bases: `object`

Wrapper class around a numpy array that keeps track of covariant and contravariant indices.

**Parameters**

- ***args** – Either a single iterable or multiple coordinate numbers.
• **covariant** (bool or list of int, optional) – If False, all indices are contravariant. If a list of indices indices is supplied, the specified indices of the array will be covariant indices and all others contravariant indices. By default all indices are covariant.

**array**
The underlying numpy array.

*Type* numpy.ndarray

**copy()**

**tensor_product**(other)
Return a new tensor that is the tensor product of this and the other tensor.

*Parameters* other (Tensor) – The other tensor.

*Returns* The tensor product.

*Return type* Tensor

**tensor_shape**
The shape or type of the tensor, the first number is the number of covariant indices, the second the number of contravariant indices.

*Type* tuple of int

**transpose**(perm=None)
Permute the indices of the tensor.

*Parameters* perm (tuple of int, optional) – A list of permuted indices or a shorter list representing a permutation in cycle notation. By default, the indices are reversed.

*Returns* The tensor with permuted indices.

*Return type* Tensor

---

**geometer.base.TensorDiagram(*edges)**

**Bases:** object

A class used to specify and calculate tensor diagrams (also called Penrose Graphical Notation).

Each edge in the diagram represents a contraction of two indices of the tensors connected by that edge. In Einstein-notation that would mean that an edge from tensor $A$ to tensor $B$ is equivalent to the expression $A_{ij}B^{ik}$, where $i, k, l$ are free indices. The indices to contract are chosen from back to front from contravariant and covariant indices of the tensors that are connected by an edge.

*Parameters* *edges* – Variable number of tuples, that represent the edge from one tensor to another.

**References**

**add_edge**(source, target)
Add an edge to the diagram.

*Parameters*

• **source** (Tensor) – The source tensor of the edge in the diagram.

• **target** (Tensor) – The target tensor of the edge in the diagram.

**calculate()**
Calculates the result of the diagram.

*Returns* The tensor resulting from the specified tensor diagram.
Return type  \textit{Tensor}

gemeter.curve module

class \texttt{geometer.curve.AlgebraicCurve}(\textit{poly}, \textit{symbols}=\text{None})
Bases: \texttt{geometer.curve.AlgebraicHypersurface}

A plane algebraic curve, defined by the zero set of a homogeneous polynomial in 3 variables.

degree
The degree of the curve, i.e. the homogeneous order of the defining polynomial.

Type  \texttt{int}

\texttt{is\_tangent}(\textit{line})
Tests if a given line is tangent to the curve.

The method compares the number of intersections to the degree of the algebraic curve. If the line is tangent to the curve it will have at least a double intersection and using Bezout’s theorem we know that otherwise the number of intersections (counted without multiplicity) is equal to the degree of the curve.

\textbf{Parameters} \textit{line}(\texttt{Line}) – The line to test.

\textbf{Returns} True if the given line is tangent to the algebraic curve.

\textbf{Return type} \texttt{bool}

class \texttt{geometer.curve.AlgebraicHypersurface}(\textit{poly}, \textit{symbols}=\text{None})
Bases: \texttt{geometer.base.ProjectiveElement}

A general algebraic hypersurface, i.e. a hypersurface defined by the zero set of a homogeneous polynomial.

\textbf{Parameters}

\begin{itemize}
  \item \texttt{poly}(\texttt{sympy.Expr} or \texttt{numpy.ndarray}) – The polynomial defining the hypersurface. It is automatically homogenized.
  \item \texttt{symbols}(\texttt{tuple} of \texttt{sympy.Symbol}, optional) – The symbols that are used in the polynomial. By default the symbols (x1, …, xn) will be used.
\end{itemize}

\texttt{symbols}
The symbols used in the polynomial defining the hypersurface.

Type  \texttt{tuple} of \texttt{sympy.Symbol}

\texttt{contains}(\textit{pt})
Tests if a given point lies on the hypersurface.

\textbf{Parameters} \textit{pt}(\texttt{Point}) – The point to test.

\textbf{Returns} True if the curve contains the point.

\textbf{Return type} \texttt{bool}

\texttt{intersect}(\textit{other})
Calculates points of intersection with the hypersurface.

\textbf{Parameters} \textit{other}(\texttt{Line} or \texttt{AlgebraicCurve}) – The object to intersect this surface with.

\textbf{Returns} The points of intersection.

\textbf{Return type} \texttt{list} of \texttt{Point}

\texttt{polynomial}
The polynomial defining this hypersurface.
**tangent** (at)
Calculates the tangent space of the surface at a given point.

**Parameters**
- **at** (*Point*) – The point to calculate the tangent space at.

**Returns**
The tangent space.

**Return type**
Line or Plane

```python
class geometer.curve.Circle(center=Point(0.0, 0.0), radius=1)
```

**Bases**
geometer.curve.Ellipse

A circle in 2D.

**Parameters**
- **center** (*Point*, *optional*) – The center point of the circle, default is Point(0, 0).
- **radius** (*float*, *optional*) – The radius of the circle, default is 1.

```python
area()
```
Calculate the area of the circle.

**Returns**
The area of the circle.

**Return type**
float

```python
center
```
The center of the circle.

**Type**
Point

```python
intersection_angle(other)
```
Calculates the angle of intersection of two circles using its Lie coordinates.

**Parameters**
- **other** (*Circle*) – The circle to intersect this circle with.

**Returns**
The angle of intersection.

**Return type**
float

**References**

```python
lie_coordinates
```
The Lie coordinates of the circle as point in RP4.

**Type**
Point

```python
radius
```
The radius of the circle.

**Type**
float

```python
class geometer.curve.Conic(matrix, is_dual=False)
```

**Bases**
geometer.curve.Quadric, geometer.curve.AlgebraicCurve

A two-dimensional conic.

**Parameters**
- **matrix** (*array_like or Tensor*) – A two dimensional array that defines the symmetric 3x3 matrix of the conic.
• **is_dual**(bool, optional) – If true, the conic represents a dual conic, i.e. all lines tangent to the non-dual conic.

**components**

The components of a degenerate conic.

  Type list of ProjectiveElement

**dual**

The dual conic.

  Type Conic

**foci**

The foci of the conic.

  Type tuple of Point

**classmethod from_crossratio**(cr, a, b, c, d)

Construct a conic from a cross ratio and four other points.

This method relies on the fact that a point lies on a conic with five other points, if and only of the cross ratio seen from this point is the same as the cross ratio of four of the other points seen from the fifth point.

  Parameters

  • cr (float) – The crossratio of the other points that defines the conic.
  • a (Point) –
  • b (Point) –
  • c (Point) –
  • d (Point) –

  Returns The resulting conic.

  Return type Conic

**References**

**classmethod from_foci**(f1, f2, bound)

Construct a conic with the given focal points that passes through the boundary point.

  Parameters

  • f1 (Point) – A focal point.
  • f2 (Point) – A focal point.
  • bound (Point) – A boundary point that lies on the conic.

  Returns The resulting conic.

  Return type Conic

**classmethod from_lines**(g, h)

Construct a degenerate conic from two lines.

  Parameters

  • g (Line) –
  • h (Line) –

  Returns The resulting conic.
Return type \textit{Conic}

\textbf{classmethod from_points\,(a, b, c, d, e)}
Construct a conic through five points.

\begin{itemize}
  \item \textit{a} (Point)
  \item \textit{b} (Point)
  \item \textit{c} (Point)
  \item \textit{d} (Point)
  \item \textit{e} (Point)
\end{itemize}

\textbf{Returns} The resulting conic.

\textbf{Return type \textit{Conic}}

\textbf{classmethod from_tangent\,(tangent, a, b, c, d)}
Construct a conic through four points and tangent to a line.

\begin{itemize}
  \item \textit{tangent} (Line)
  \item \textit{a} (Point)
  \item \textit{b} (Point)
  \item \textit{c} (Point)
  \item \textit{d} (Point)
\end{itemize}

\textbf{Returns} The resulting conic.

\textbf{Return type \textit{Conic}}

\textbf{intersect\,(other)}
Calculates points of intersection with the conic.

\textbf{Parameters} \textit{other} (Line or Conic) – The object to intersect this curve with.

\textbf{Returns} The points of intersection

\textbf{Return type} list of Point

\textbf{polar\,(pt)}
Calculates the polar line of the conic at a given point.

\textbf{Parameters} \textit{pt} (Point) – The point to calculate the polar at.

\textbf{Returns} The polar line.

\textbf{Return type \textit{Line}}

\textbf{tangent\,(at)}
Calculates the tangent line at a given point or the tangent lines between a point and the conic.

\textbf{Parameters} \textit{at} (Point) – The point to calculate the tangent at.

\textbf{Returns} The tangent line(s).

\textbf{Return type \textit{Line} or tuple of Line}
class geometer.curve.Ellipse(center=Point(0.0, 0.0), hradius=1, vradius=1)
Bases: geometer.curve.Conic

Represents an ellipse in 2D.

Parameters

- center (Point, optional) – The center of the ellipse, default is Point(0, 0).
- hradius (float, optional) – The horizontal radius (along the x-axis), default is 1.
- vradius (float, optional) – The vertical radius (along the y-axis), default is 1.

class geometer.curve.Quadric(matrix)
Bases: geometer.curve.AlgebraicHypersurface

Represents a quadric, i.e. the zero set of a polynomial of degree 2, in any dimension.

The quadric is defined by a symmetric matrix of size n+1 where n is the dimension of the space.

Parameters matrix (array_like or Tensor) – The array defining a (n+1)x(n+1) symmetric matrix.

contains (pt)
Tests if a given point lies on the quadric.

Parameters pt (Point) – The point to test.

Returns True if the quadric contains the point.

Return type bool

is_degenerate
True if the quadric is degenerate.

Type bool

polynomial
The polynomial defining this quadric.

Type sympy.Poly

tangent (at)
Returns the hyperplane defining the tangent space at a given point.

Parameters at (Point) – The point at which the tangent space is calculated.

Returns The tangent plane at the given point.

Return type Plane

class geometer.curve.Sphere(center=Point(0.0, 0.0, 0.0), radius=1)
Bases: geometer.curve.Quadric

A sphere in any dimension.

Parameters

- center (Point, optional) – The center of the sphere, default is Point(0, 0, 0).
- radius (float, optional) – The radius of the sphere, default is 1.

area ()
Calculate the surface area of the sphere.

Returns The surface area of the sphere.

Return type float
center
The center of the sphere.

Type Point

radius
The radius of the sphere.

Type float

volume()
Calculate the volume of the sphere.

Returns The volume of the sphere.

Return type float

gemeter.exceptions module

gemeter.exceptions.GeometryException
exception
gemeter.exceptions.GeometryException
Bases: Exception
A general geometric error occurred.

gemeter.exceptions.IncidenceError
exception
gemeter.exceptions.IncidenceError
Bases: gemeter.exceptions.GeometryException, ValueError
The given objects were not incident to each other.

gemeter.exceptions.LinearDependenceError
exception
gemeter.exceptions.LinearDependenceError
Bases: gemeter.exceptions.GeometryException, ValueError
The given values were linearly dependent, making the computation impossible.

gemeter.exceptions.NotCollinear
exception
gemeter.exceptions.NotCollinear
Bases: gemeter.exceptions.GeometryException, ValueError
The given values are not collinear.

gemeter.exceptions.NotCoplanar
exception
gemeter.exceptions.NotCoplanar
Bases: gemeter.exceptions.GeometryException, ValueError
The given values are not coplanar.

gemeter.exceptions.TensorComputationError
exception
gemeter.exceptions.TensorComputationError
Bases: gemeter.exceptions.GeometryException
An error during a tensor computation occurred.

gemeter.operators module

gemeter.operators.angle(*args)
Calculates the (oriented) angle between given points, lines or planes.

The function uses the Laguerre formula to calculate angles in two or three dimensional projective space using cross ratios. To calculate the angle between two planes, two additional planes tangent to the absolute conic are constructed (see [1]).

Parameters *args – The objects between which the function calculates the angle. This can be 2 or 3 points, 2 lines or 2 planes.

Returns The oriented angle between the given objects.
Return type  float

References

geometer.operators.angle_bisectors(l, m)
Constructs the angle bisectors of two given lines.

Parameters
• l (Line) – A line in any dimension.
• m (Line) – A line in any dimension.

Returns  The two angle bisectors.

Return type  tuple of Line

geometer.operators.crossratio(a, b, c, d, from_point=None)
Calculates the cross ratio of points or lines.

Parameters
• a (Point, Line or Plane) – A point, line or plane in any dimension.
• b (Point, Line or Plane) – A point, line or plane in any dimension.
• c (Point, Line or Plane) – A point, line or plane in any dimension.
• d (Point, Line or Plane) – A point, line or plane in any dimension.
• from_point (Point, optional) – A 2D point, only possible if the other arguments are also 2D points.

Returns  The cross ratio of the given objects.

Return type  float or complex

geometer.operators.dist(p, q)
Calculates the (euclidean) distance between two objects.

Parameters
• p (Point, Line or Plane) – A point, line or plane to calculate the distance to.
• q (Point, Line or Plane) – A point, line or plane to calculate the distance to.

Returns  The distance between the given objects.

Return type  float

geometer.operators.harmonic_set(a, b, c)
Constructs a fourth point that forms a harmonic set with the given points.

The three given points must be collinear.

If the returned point is d, the points \{a, b\}, \{c, d\} will be in harmonic position.

Parameters
• a (Point) – A point in any dimension.
• b (Point) – A point in any dimension.
• c (Point) – A point in any dimension.

Returns  The point that forms a harmonic set with the given points.

Return type  Point
geometer.operators.is_cocircular\( (a, b, c, d) \)
Tests whether four points lie on a circle.

**Parameters**

- \( a \) (Point) – A point in RP2 or CP1.
- \( b \) (Point) – A point in RP2 or CP1.
- \( c \) (Point) – A point in RP2 or CP1.
- \( d \) (Point) – A point in RP2 or CP1.

**Returns** True if the four points lie on a circle.

**Return type** bool

geometer.operators.is_collinear\(*args\)
Tests whether the given points or lines are collinear, coplanar or concurrent. Works in any dimension.

Due to line point duality this function has dual versions \( is\_collinear, is\_collinear \) and \( is\_concurrent \).

**Parameters** *args – The points or lines to test.

**Returns** True if the given points are coplanar (in 3D) or collinear (in 2D) or if the given lines are concurrent.

**Return type** bool

geometer.operators.is_concurrent\(*args\)
Tests whether the given points or lines are collinear, coplanar or concurrent. Works in any dimension.

Due to line point duality this function has dual versions \( is\_collinear, is\_collinear \) and \( is\_concurrent \).

**Parameters** *args – The points or lines to test.

**Returns** True if the given points are coplanar (in 3D) or collinear (in 2D) or if the given lines are concurrent.

**Return type** bool

geometer.operators.is_coplanar\(*args\)
Tests whether the given points or lines are collinear, coplanar or concurrent. Works in any dimension.

Due to line point duality this function has dual versions \( is\_collinear, is\_collinear \) and \( is\_concurrent \).

**Parameters** *args – The points or lines to test.

**Returns** True if the given points are coplanar (in 3D) or collinear (in 2D) or if the given lines are concurrent.

**Return type** bool

geometer.operators.is_perpendicular\( (l, m) \)
Tests whether two lines/planes are perpendicular.

**Parameters**

- \( l \) (Line or Plane) – A line in any dimension or a plane in 3D.
- \( m \) (Line or Plane) – A second line or a plane in 3D.

**Returns** True if the two lines/planes are perpendicular.

**Return type** bool
geometer.point module

class geometer.point.Line(*args)
    Bases: geometer.point.Subspace

    Represents a line in a projective space of arbitrary dimension.

    Parameters *args – Two points or the coordinates of the line. Instead of all coordinates separately,
                       a single iterable can also be supplied.

    base_point
        A base point for the line, arbitrarily chosen.
        Type Point

    basis_matrix
        A matrix with orthonormal basis vectors as rows.
        Type numpy.ndarray

    contravariant_tensor
        The contravariant version of a line in 3D.
        Type Line

    covariant_tensor
        The covariant version of a line in 3D.
        Type Line

    direction
        The direction of the line (not normalized).
        Type Point

    is_coplanar(other)
        Tests whether another line lies in the same plane as this line, i.e. whether two lines intersect.
        Parameters other (Line) – A line in 3D to test.
        Returns True if the two lines intersect (i.e. they lie in the same plane).
        Return type bool

    References

    lie_coordinates
        The Lie coordinates of a line in 2D.
        Type Point

    mirror(pt)
        Construct the reflection of a point at this line.
        Parameters pt (Point) – The point to reflect.
        Returns The mirror point.
        Return type Point

    16 Chapter 1. Installation
References

**perpendicular (through)**
Construct the perpendicular line though a point.

- **Parameters** through (**Point**) – The point through which the perpendicular is constructed.
- **Returns** The perpendicular line.
- **Return type** **Line**

**project (pt)**
The orthogonal projection of a point onto the line.

- **Parameters** pt (**Point**) – The point to project.
- **Returns** The projected point.
- **Return type** **Point**

**class geometer.point.Plane (**args**)**
**Bases:** **geometer.point.Subspace**

- **basis_matrix**
  A matrix with orthonormal basis vectors as rows.
  - **Type** numpy.ndarray

- **mirror (pt)**
  Construct the reflection of a point at this plane.
  Only works in 3D.
  - **Parameters** pt (**Point**) – The point to reflect.
  - **Returns** The mirror point.
  - **Return type** **Point**

- **perpendicular (through)**
  Construct the perpendicular line though a point.
  Only works in 3D.
  - **Parameters** through (**Point**) – The point through which the perpendicular is constructed.
  - **Returns** The perpendicular line.
  - **Return type** **Line**

- **polynomial**
The polynomial defining this hyperplane.
  - **Type** sympy.Poly

- **project (pt)**
The orthogonal projection of a point onto the plane.
  Only works in 3D.
  - **Parameters** pt (**Point**) – The point to project.
  - **Returns** The projected point.
  - **Return type** **Point**
class geometer.point.Point(*args)
Bases: geometer.base.ProjectiveElement

Represents points in a projective space of arbitrary dimension.

The number of supplied coordinates determines the dimension of the space that the vector lives in. If the coordinates are given as arguments (not in a single iterable), the coordinates will automatically be transformed into homogeneous coordinates, i.e. a one added as an additional coordinate.

**Parameters**

*args – A single iterable object or tensor or multiple (affine) coordinates.

join(*others)

Execute the join of this point with other objects.

**Parameters**

*others – The objects to join the point with.

**Returns**

The result of the join operation.

**Return type**

Line or Plane

See also:

join()

lie_coordinates

The Lie coordinates of a point in 2D.

**Type**

Point

normalized_array

The normalized coordinates as array.

**Type**
	numpy.ndarray

class geometer.point.Subspace(*args)
Bases: geometer.base.ProjectiveElement

Represents a general subspace of a projective space. Line and Plane are subclasses.

**Parameters**

*args – The coordinates of the subspace. Instead of all coordinates separately, a single iterable can also be supplied.

basis_matrix

A matrix with orthonormal basis vectors as rows.

**Type**

numpy.ndarray

contains(other)

Tests whether a given point or line lies in the subspace.

**Parameters**

other (Point or Line) – The object to test.

**Returns**

True, if the given point/line lies in the subspace.

**Return type**

bool

general_point

A point in general position i.e. not in the subspace, to be used in geometric constructions.

**Type**

Point

is_parallel(other)

Tests whether a given subspace is parallel to this subspace.

**Parameters**

other (Subspace) – The other space to test.

**Returns**

True, if the two spaces are parallel.
Return type  bool

join (*others)
  Execute the join of the subspace with other objects.

  Parameters  *others – The objects to join the subspace with.

  Returns  The result of the join operation.

  Return type  Subspace

See also:

  join ()

meet (*others)
  Intersect the subspace with other objects.

  Parameters  *others – The objects to intersect the subspace with.

  Returns  The result of the meet operation.

  Return type  Subspace or Point

See also:

  meet ()

parallel (through)
  Returns the subspace through a given point that is parallel to this subspace.

  Parameters  through (Point) – The point through which the parallel subspace is to be con-structed.

  Returns  The parallel subspace.

  Return type  Subspace

polynomials (symbols=None)
  Returns a list of polynomials, to use for symbolic calculations.

  Parameters  symbols (list of sympy.Symbol, optional) – The symbols used in the result-
ing polynomial. By default “x1”, …, “xn” will be used.

  Returns  The polynomials describing the subspace.

  Return type  list of sympy.Poly

geometer.point .infty_hyperplane (dimension)

geometer.point .join (*args)
  Joins a number of objects to form a line or plane.

  Parameters  *args – Objects to join, e.g. 2 points, lines, a point and a line or 3 points.

  Returns  The resulting plane or line.

  Return type  Line or Plane

geometer.point .meet (*args)
  Intersects a number of given objects.

  Parameters  *args – Objects to intersect, e.g. two lines, planes, a plane and a line or 3 planes.

  Returns  The resulting point or line.

  Return type  Point or Line
geometer Documentation, Release 0.2.0

geometer.shapes module

class geometer.shapes.Cuboid(a, b, c, d)
Bases: geometer.shapes.Polyhedron

A class that can be used to construct a cuboid/box or a cube.

Parameters

• a (Point) – The base point of the cuboid.
• b (Point) – The vertex that determines the first direction of the edges.
• b – The vertex that determines the second direction of the edges.
• b – The vertex that determines the third direction of the edges.

class geometer.shapes.Polygon (*args)
Bases: geometer.shapes.Polytope

A flat polygon with vertices in any dimension.

Parameters *args – The coplanar points that are the vertices of the polygon. They will be connected sequentially by line segments.

angles
The interior angles of the polygon.

Type list of float

area
The area of the polygon.

Type float

contains (other)
Tests whether a point is contained in the polygon.

Parameters other (Point) – The point to test.

Returns True if the point is contained in the polygon.

Return type bool

edges

intersect (other)
Intersect the polygon with another object.

Parameters other (Line or Segment) –

Returns The points of intersection.

Return type list of Point

class geometer.shapes.Polyhedron (*args)
Bases: geometer.shapes.Polytope

A class representing polyhedra (3-polytopes).

area
The surface area of the polyhedron.

Type float

edges

faces
**intersect**(other)

Intersect the polyhedron with another object.

Parameters **other**(Line or Segment) -

Returns The points of intersection.

Return type list of Point

class geometer.shapes.Polytope(*args)

Bases: object

A class representing polytopes in arbitrary dimension. A (n+1)-polytope is a collection of n-polytopes that have some (n-1)-polytopes in common, where 3-polytopes are polyhedra, 2-polytopes are polygons and 1-polytopes are line segments.

Parameters ***args** – The polytopes defining the facets of the polytope.

dim

The dimension of the space that the polytope lives in.

Type int

facets

The facets of the polytope.

Type list of Polytope

vertices

The vertices of the polytope.

Type list of Point

class geometer.shapes.Rectangle(a, b, c, d)

Bases: geometer.shapes.Polygon

A class representing rectangles.

Parameters

- a(Point) -
- b(Point) -
- c(Point) -
- d(Point) -

class geometer.shapes.RegularPolygon(center, radius, n, axis=None)

Bases: geometer.shapes.Polygon

A class that can be used to construct regular polygon from a radius and a center point.

Parameters

- center(Point) – The center of the polygon.
- radius(float) – The distance from the center to the vertices of the polygon.
- axis(Point, optional) – If constructed in higher-dimensional spaces, an axis vector is required to orient the polygon.

class geometer.shapes.Segment(p, q)

Bases: geometer.shapes.Polytope

Represents a line segment in an arbitrary projective space.
As a (real) projective line is homeomorphic to a circle, there are two line segments that connect two points. An instance of this class will represent the finite segment connecting the two points, if there is one, and the segment in the direction of the infinite point otherwise (identifying only scalar multiples by positive scalar factors). When both points are at infinity, the points will be considered in the oriented projective space to define the segment between them.

Segments with one point at infinity represent rays/half-lines in a traditional sense.

Parameters

- \( p \) (Point) – The start of the line segment.
- \( q \) (Point) – The end point of the segment.

contains \( (other) \)
Tests whether a point is contained in the segment.

Parameters \( other \) (Point) – The point to test.

Returns True if the point is contained in the segment.

Return type bool

intersect \( (other) \)
Intersect the line segment with another object.

Parameters \( other \) (Line, Plane, Segment, Polygon or Polyhedron) –

Returns The points of intersection.

Return type list of Point

length
The length of the segment.

Type float

midpoint
The midpoint of the segment.

Type Point

vertices
The vertices of the polytope.

Type list of Point

class geometer.shapes.Simplex(*args)
Bases: geometer.shapes.Polytope
Represents a simplex in any dimension, i.e. a k-polytope with k+1 vertices where k is the dimension.

The simplex determined by k+1 points is given by the convex hull of these points.

Parameters *args – The points that are the vertices of the simplex.

volume
The volume of the simplex, calculated using the Cayley–Menger determinant.

Type float

class geometer.shapes.Triangle \((a, b, c)\)
Bases: geometer.shapes.Polygon, geometer.shapes.Simplex
A class representing triangles.

Parameters
• a (Point)
• b (Point)
• c (Point)

geometer.transformation module

class geometer.transformation.Transformation(*args)
    Bases: geometer.base.ProjectiveElement

    Represents a projective transformation in an arbitrary projective space.
    The underlying array of the transformation is the matrix representation of the projective transformation. The
    transformation can be applied to points by simple multiplication.

    Parameters *args – The array that defines the matrix representing the transformation.

classmethod from_points(*args)
    Constructs a projective transformation in n-dimensional projective space from the image of n + 2 points.
    For two dimensional transformations, 4 pairs of points are required. For three dimensional transformations,
    5 pairs of points are required.

    Parameters *args – Pairs of points, where in each pair one point is mapped to the other.
    Returns The transformation mapping each of the given points to the specified points.
    Return type Transformation

inverse()
    Calculates the inverse projective transformation.

    Returns The inverse transformation.
    Return type Transformation

geometer.transformation.rotation(angle, axis=None)
    Returns a projective transformation that represents a rotation by the specified angle (and axis).

    Parameters

    • angle (float) – The angle to rotate by.
    • axis (Point, optional) – The axis to rotate around when rotating points in 3D.

    Returns The rotation.
    Return type Transformation

geometer.transformation.translation(*coordinates)
    Returns a projective transformation that represents a translation by the given coordinates.

    Parameters *coordinates – The coordinates by which points are translated when applying the
    resulting transformation.
    Returns The translation.
    Return type Transformation
1.3 Changelog

1.3.1 0.2 - unreleased

1.3.2 New Features

- New shapes module that implements line segments, polygons and general polytopes
- Tensor has a new tensor_product method to calculate the tensor product with another tensor
- AlgebraicCurve is now a subclass of the new class AlgebraicHypersurface
- A new sphere class (a subclass of Quadric) that works in any dimension
- Ellipse class that constructs a conic from center and radius
- Added Conic.foci and Conic.polar
- Construct a conic from its focal points, using a tangent line or a cross ratio
- Refactored & documented the code for calculation of tensor diagrams
- New KroneckerDelta tensor
- Added Circle.intersection_angle to calculate the angle of intersection of two circles
- is_perpendicular now works with two planes

1.3.3 Bug fixes

- Plane.perpendicular now also works for points that lie on the plane

1.3.4 0.1.2 - released (24.2.2019)

1.3.5 New Features

- Optimized performance of Conic, LeviCivitaTensor and TensorDiagram
- More operations are now compatible with higher-dimensional objects
- New Subspace class that can be used to represent subspaces of any dimension
- New repr and copy methods of Tensor
- scipy is no longer a dependency

1.3.6 Bug fixes

- Rotation in 3D now returns the correct transformation if the axis is not a normalized vector
- Line.perpendicular now also works for points that lie on the line

1.3.7 0.1.1 - released (2.2.2019)
CHAPTER 2

Indices and tables

• genindex
• modindex
• search

2.1 References

Many of the algorithms and formulas implemented in the package are taken from the following books and papers:

• Jürgen Richter-Gebert, Perspectives on Projective Geometry
• Jürgen Richter-Gebert and Thorsten Orendt, Geometriekalküle
• Olivier Faugeras, Three-Dimensional Computer Vision
• Jim Blinn, Lines in Space: The 4D Cross Product
• Jim Blinn, Lines in Space: The Line Formulation
• Jim Blinn, Lines in Space: The Two Matrices
• Jim Blinn, Lines in Space: Back to the Diagrams
• Jim Blinn, Lines in Space: A Tale of Two Lines
• Jim Blinn, Lines in Space: Our Friend the Hyperbolic Paraboloid
• Jim Blinn, Lines in Space: The Algebra of Tinkertoys
• Jim Blinn, Lines in Space: Line(s) through Four Lines
geometer.base, 6
geometer.curve, 8
geometer.exceptions, 13
geometer.operators, 13
geometer.point, 16
geometer.shapes, 20
geometer.transformation, 23
geometer.utils.math, 5
geometer.utils.polynomial, 5
Index

A
add_edge() (geometer.base.TensorDiagram method), 7
AlgebraicCurve (class in geometer.curve), 8
AlgebraicHypersurface (class in geometer.curve), 8
angle() (in module geometer.operators), 8
angles (geometer.shapes.Polygon attribute), 20
area (geometer.shapes.Polygon attribute), 20
area (geometer.shapes.Polyhedron attribute), 20
area() (geometer.curve.Circle method), 9
area() (geometer.curve.Sphere method), 12
array (geometer.base.Tensor attribute), 7
copy() (geometer.base.Tensor method), 7
covariant_tensor (geometer.point.Line attribute), 16
crossratio() (in module geometer.operators), 14
Cuboid (class in geometer.shapes), 20

D
degree (geometer.curve.AlgebraicCurve attribute), 8
dim (geometer.base.ProjectiveElement attribute), 6
dim (geometer.shapes.Polytope attribute), 21
direction (geometer.point.Line attribute), 16
dist() (in module geometer.operators), 14
dual (geometer.curve.Conic attribute), 10

E
dual (geometer.curve.Conic attribute), 10
edges (geometer.shapes.Polygon attribute), 20
edges (geometer.shapes.Polyhedron attribute), 20
Ellipse (class in geometer.curve), 11

F
faces (geometer.shapes.Polyhedron attribute), 20
facets (geometer.shapes.Polytope attribute), 21
from_crossratio() (geometer.curve.Conic class method), 10
from_foci() (geometer.curve.Conic class method), 10
from_lines() (geometer.curve.Conic class method), 10
from_points() (geometer.curve.Conic class method), 11
from_points() (geometer transformation.Transformation class method), 23
from_tangent() (geometer.curve.Conic class method), 11

G
general_point (geometer.point.Subspace attribute), 18
geometer.base (module), 6
geometer.curve (module), 8
geometer.exceptions (module), 13
geometer.operators (module), 13
geometer.point (module), 16
geometer.shapes (module), 20
geometer.transformation (module), 23
geometer.utils.math (module), 5
geometer.utils.polynomial (module), 5
GeometryException, 13
HarmonicException, 13

HarmonicException (in module geometer.operators), 14
H
HarmonicException, 13

IncidenceError, 13
I
IncidenceError, 13

Interceptor (geometer.curve.AlgebraicHypersurface method), 8
Intersector (geometer.curve.Conic method), 11
Intersector (geometer.shapes.Polygon method), 20
Intersector (geometer.shapes.Polyhedron method), 20
Intersector (geometer.shapes.Segment method), 22
Intersection_angle (geometer.curve.Circle method), 9
Inverse (geometer.transformation.Transformation method), 23
Is_cocircular (in module geometer.operators), 14
Is_collinear (in module geometer.operators), 15
Is_concurrent (in module geometer.operators), 15
Is_coplanar (geometer.point.Line method), 16
Is_coplanar (in module geometer.operators), 15
Is_degenerate (geometer.curve.Quadric attribute), 12
Is_parallel (geometer.point.Subspace method), 18
Is_perpendicular (in module geometer.operators), 15
Is_tangent (geometer.curve.AlgebraicCurve method), 8
J
Join (geometer.point.Point method), 18
Join (geometer.point.Subspace method), 19
Join (in module geometer.point), 19
K
KroneckerDelta (class in geometer.base), 6
L
Length (geometer.shapes.Segment attribute), 22
LeviCivitaTensor (class in geometer.base), 6
L
Lie_coordinates (geometer.curve.Circle attribute), 9
Lie_coordinates (geometer.point.Line attribute), 16
Lie_coordinates (geometer.point.Point attribute), 18
Line (class in geometer.point), 16
LinearDependenceError, 13
M
Meet (geometer.point.Subspace method), 19
Meet (in module geometer.point), 19
Midpoint (geometer.shapes.Segment attribute), 22
Mirror (geometer.point.Line method), 16
Mirror (geometer.point.Plane method), 17
Normalized_array (geometer.point.Point attribute), 18
NotCollinear, 13
NotCoplanar, 13
N
Np_array_to_poly (in module geometer.utils.polynomial), 5
Null_space (in module geometer.utils.math), 5
Polytope (class in geometer.shapes), 21
Polyval (in module geometer.utils.polynomial), 5
Project (geometer.point.Line method), 17
Project (geometer.point.Plane method), 17
ProjectiveElement (class in geometer.base), 6
Q
Quadric (class in geometer.curve), 12
R
radius (geometer.curve.Circle attribute), 9
radius (geometer.curve.Sphere attribute), 13
Rectangle (class in geometer.shapes), 21
RegularPolygon (class in geometer.shapes), 21
rotation() (in module geometer.transformation), 23

S
Segment (class in geometer.shapes), 21
Simplex (class in geometer.shapes), 22
Sphere (class in geometer.curve), 12
Subspace (class in geometer.point), 18
symbols (geometer.curve.AlgebraicHypersurface attribute), 8

T
tangent() (geometer.curve.AlgebraicHypersurface method), 9
tangent() (geometer.curve.Conic method), 11
tangent() (geometer.curve.Quadric method), 12
Tensor (class in geometer.base), 6
tensor_product() (geometer.base.Tensor method), 7
tensor_shape (geometer.base.Tensor attribute), 7
TensorComputationError, 13
TensorDiagram (class in geometer.base), 7
Transformation (class in geometer.transformation), 23
translation() (in module geometer.transformation), 23
transpose() (geometer.base.Tensor method), 7
Triangle (class in geometer.shapes), 22

V
vertices (geometer.shapes.Polytope attribute), 21
vertices (geometer.shapes.Segment attribute), 22
volume (geometer.shapes.Simplex attribute), 22
volume() (geometer.curve.Sphere method), 13