property centroid

The centroid of the polygon.

Returns

centroid : Point

Examples

```
>>> from sympy import Point, Polygon
>>> p1, p2, p3, p4 = map(Point, [(0, 0), (1, 0), (5, 1), (0, 1)])
>>> poly = Polygon(p1, p2, p3, p4)
>>> poly.centroid
Point2D(31/18, 11/18)
```

See also:

sympy.geometry.point.Point (page 2202), *sympy.geometry.util.centroid* (page 2200)

cut_section(*line*)

Returns a tuple of two polygon segments that lie above and below the intersecting line respectively.

Parameters

line: Line object of geometry module

line which cuts the Polygon. The part of the Polygon that lies above and below this line is returned.

Returns

upper_polygon, lower_polygon: Polygon objects or None

upper_polygon is the polygon that lies above the given line. lower_polygon is the polygon that lies below the given line. upper_polygon and lower polygon are None when no polygon exists above the line or below the line.

Raises

ValueError: When the line does not intersect the polygon

Examples

```
>>> from sympy import Polygon, Line
>>> a, b = 20, 10
>>> p1, p2, p3, p4 = [(0, b), (0, 0), (a, 0), (a, b)]
>>> rectangle = Polygon(p1, p2, p3, p4)
>>> t = rectangle.cut_section(Line((0, 5), slope=0))
>>> t
(Polygon(Point2D(0, 10), Point2D(0, 5), Point2D(20, 5), Point2D(20,␣
↪10)),
Polygon(Point2D(0, 5), Point2D(0, 0), Point2D(20, 0), Point2D(20, 5)))
>>> upper_segment, lower_segment = t
>>> upper_segment.area
100
```

(continues on next page)

(continued from previous page)

```
>>> upper_segment.centroid
Point2D(10, 15/2)
>>> lower_segment.centroid
Point2D(10, 5/2)
```

References

[R511]

distance(*o*)

Returns the shortest distance between self and o.

If o is a point, then self does not need to be convex. If o is another polygon self and o must be convex.

Examples

```
>>> from sympy import Point, Polygon, RegularPolygon
>>> p1, p2 = map(Point, [(0, 0), (7, 5)])
>>> poly = Polygon(*RegularPolygon(p1, 1, 3).vertices)
>>> poly.distance(p2)
sqrt(61)
```

encloses_point(*p*)

Return True if p is enclosed by (is inside of) self.

Parameters

p : Point

Returns

encloses_point : True, False or None

Notes

Being on the border of self is considered False.

Examples

```
>>> from sympy import Polygon, Point
>>> p = Polygon((0, 0), (4, 0), (4, 4))
>>> p.encloses_point(Point(2, 1))
True
>>> p.encloses_point(Point(2, 2))
False
>>> p.encloses_point(Point(5, 5))
False
```

See also:

sympy.geometry.point.Point (page 2202), *sympy.geometry.ellipse.Ellipse.encloses_point* (page 2257)

References

[R512]

first_moment_of_area(*point=None*)

Returns the first moment of area of a two-dimensional polygon with respect to a certain point of interest.

First moment of area is a measure of the distribution of the area of a polygon in relation to an axis. The first moment of area of the entire polygon about its own centroid is always zero. Therefore, here it is calculated for an area, above or below a certain point of interest, that makes up a smaller portion of the polygon. This area is bounded by the point of interest and the extreme end (top or bottom) of the polygon. The first moment for this area is is then determined about the centroidal axis of the initial polygon.

Parameters

point: Point, two-tuple of sympifyable objects, or None (default=None)

point is the point above or below which the area of interest lies If `point=None` then the centroid acts as the point of interest.

Returns

Q_x, Q_y: number or SymPy expressions

Q_x is the first moment of area about the x-axis Q_y is the first moment of area about the y-axis A negative sign indicates that the section modulus is determined for a section below (or left of) the centroidal axis

Examples

```
>>> from sympy import Point, Polygon
>>> a, b = 50, 10
>>> p1, p2, p3, p4 = [(0, b), (0, 0), (a, 0), (a, b)]
>>> p = Polygon(p1, p2, p3, p4)
>>> p.first_moment_of_area()
(625, 3125)
>>> p.first_moment_of_area(point=Point(30, 7))
(525, 3000)
```

References

[R513], [R514]

intersection(*o*)

The intersection of polygon and geometry entity.

The intersection may be empty and can contain individual Points and complete Line Segments.

Parameters

other: GeometryEntity

Returns

intersection : list

The list of Segments and Points

Examples

```
>>> from sympy import Point, Polygon, Line
>>> p1, p2, p3, p4 = map(Point, [(0, 0), (1, 0), (5, 1), (0, 1)])
>>> poly1 = Polygon(p1, p2, p3, p4)
>>> p5, p6, p7 = map(Point, [(3, 2), (1, -1), (0, 2)])
>>> poly2 = Polygon(p5, p6, p7)
>>> poly1.intersection(poly2)
[Point2D(1/3, 1), Point2D(2/3, 0), Point2D(9/5, 1/5), Point2D(7/3, 1)]
>>> poly1.intersection(Line(p1, p2))
[Segment2D(Point2D(0, 0), Point2D(1, 0))]
>>> poly1.intersection(p1)
[Point2D(0, 0)]
```

See also:

sympy.geometry.point.Point (page 2202), *sympy.geometry.line.Segment* (page 2233)

is_convex()

Is the polygon convex?

A polygon is convex if all its interior angles are less than 180 degrees and there are no intersections between sides.

Returns

is_convex : boolean

True if this polygon is convex, False otherwise.

Examples

```
>>> from sympy import Point, Polygon
>>> p1, p2, p3, p4 = map(Point, [(0, 0), (1, 0), (5, 1), (0, 1)])
>>> poly = Polygon(p1, p2, p3, p4)
>>> poly.is_convex()
True
```

See also:

sympy.geometry.util.convex_hull (page 2198)

property perimeter

The perimeter of the polygon.

Returns

perimeter : number or Basic instance

Examples

```
>>> from sympy import Point, Polygon
>>> p1, p2, p3, p4 = map(Point, [(0, 0), (1, 0), (5, 1), (0, 1)])
>>> poly = Polygon(p1, p2, p3, p4)
>>> poly.perimeter
sqrt(17) + 7
```

See also:

sympy.geometry.line.Segment.length (page 2235)

plot_interval(*parameter='t'*)

The plot interval for the default geometric plot of the polygon.

Parameters

parameter : str, optional

Default value is 't'.

Returns

plot_interval : list (plot interval)

[parameter, lower_bound, upper_bound]

Examples

```
>>> from sympy import Polygon
>>> p = Polygon((0, 0), (1, 0), (1, 1))
>>> p.plot_interval()
[t, 0, 1]
```

polar_second_moment_of_area()

Returns the polar modulus of a two-dimensional polygon

It is a constituent of the second moment of area, linked through the perpendicular axis theorem. While the planar second moment of area describes an object's resistance to deflection (bending) when subjected to a force applied to a plane parallel to the central axis, the polar second moment of area describes an object's resistance to deflection when subjected to a moment applied in a plane perpendicular to the object's central axis (i.e. parallel to the cross-section)

Examples

```
>>> from sympy import Polygon, symbols
>>> a, b = symbols('a, b')
>>> rectangle = Polygon((0, 0), (a, 0), (a, b), (0, b))
>>> rectangle.polar_second_moment_of_area()
a**3*b/12 + a*b**3/12
```

References

[R515]

second_moment_of_area(*point=None*)

Returns the second moment and product moment of area of a two dimensional polygon.

Parameters

point : Point, two-tuple of sympifyable objects, or None(default=None)

point is the point about which second moment of area is to be found. If "point=None" it will be calculated about the axis passing through the centroid of the polygon.

Returns

I_xx, I_yy, I_xy : number or SymPy expression

I_xx, I_yy are second moment of area of a two dimensional polygon. I_xy is product moment of area of a two dimensional polygon.

Examples

```
>>> from sympy import Polygon, symbols
>>> a, b = symbols('a, b')
>>> p1, p2, p3, p4, p5 = [(0, 0), (a, 0), (a, b), (0, b), (a/3, b/3)]
>>> rectangle = Polygon(p1, p2, p3, p4)
>>> rectangle.second_moment_of_area()
(a*b**3/12, a**3*b/12, 0)
>>> rectangle.second_moment_of_area(p5)
(a*b**3/9, a**3*b/9, a**2*b**2/36)
```

References

[R516]

section_modulus(*point=None*)

Returns a tuple with the section modulus of a two-dimensional polygon.

Section modulus is a geometric property of a polygon defined as the ratio of second moment of area to the distance of the extreme end of the polygon from the centroidal axis.

Parameters

point : Point, two-tuple of sympifyable objects, or None(default=None)

point is the point at which section modulus is to be found. If "point=None" it will be calculated for the point farthest from the centroidal axis of the polygon.

Returns

S_x, S_y: numbers or SymPy expressions

S_x is the section modulus with respect to the x-axis S_y is the section modulus with respect to the y-axis A negative sign indicates that the section modulus is determined for a point below the centroidal axis

Examples

```
>>> from sympy import symbols, Polygon, Point
>>> a, b = symbols('a, b', positive=True)
>>> rectangle = Polygon((0, 0), (a, 0), (a, b), (0, b))
>>> rectangle.section_modulus()
(a*b**2/6, a**2*b/6)
>>> rectangle.section_modulus(Point(a/4, b/4))
(-a*b**2/3, -a**2*b/3)
```

References

[R517]

property sides

The directed line segments that form the sides of the polygon.

Returns

sides : list of sides

Each side is a directed Segment.

Examples

```
>>> from sympy import Point, Polygon
>>> p1, p2, p3, p4 = map(Point, [(0, 0), (1, 0), (5, 1), (0, 1)])
>>> poly = Polygon(p1, p2, p3, p4)
>>> poly.sides
[Segment2D(Point2D(0, 0), Point2D(1, 0)),
Segment2D(Point2D(1, 0), Point2D(5, 1)),
Segment2D(Point2D(5, 1), Point2D(0, 1)), Segment2D(Point2D(0, 1),␣
↪Point2D(0, 0))]
```

See also:

sympy.geometry.point.Point (page 2202), *sympy.geometry.line.Segment* (page 2233)

property vertices

The vertices of the polygon.

Returns

vertices : list of Points

Notes

When iterating over the vertices, it is more efficient to index self rather than to request the vertices and index them. Only use the vertices when you want to process all of them at once. This is even more important with RegularPolygons that calculate each vertex.

Examples

```
>>> from sympy import Point, Polygon
>>> p1, p2, p3, p4 = map(Point, [(0, 0), (1, 0), (5, 1), (0, 1)])
>>> poly = Polygon(p1, p2, p3, p4)
>>> poly.vertices
[Point2D(0, 0), Point2D(1, 0), Point2D(5, 1), Point2D(0, 1)]
>>> poly.vertices[0]
Point2D(0, 0)
```

See also:

sympy.geometry.point.Point (page 2202)

class sympy.geometry.polygon.**RegularPolygon**(*c, r, n, rot=0, **kwargs*)

A regular polygon.

Such a polygon has all internal angles equal and all sides the same length.

Parameters

center : Point

radius : number or Basic instance

The distance from the center to a vertex

n : int

The number of sides

Raises

GeometryError

If the *center* is not a Point, or the *radius* is not a number or Basic instance, or the number of sides, n, is less than three.

Notes

A RegularPolygon can be instantiated with Polygon with the kwarg n.

Regular polygons are instantiated with a center, radius, number of sides and a rotation angle. Whereas the arguments of a Polygon are vertices, the vertices of the Regular-Polygon must be obtained with the vertices method.

Examples

```
>>> from sympy import RegularPolygon, Point
>>> r = RegularPolygon(Point(0, 0), 5, 3)
>>> r
RegularPolygon(Point2D(0, 0), 5, 3, 0)
>>> r.vertices[0]
Point2D(5, 0)
```

See also:

sympy.geometry.point.Point (page 2202), *Polygon* (page 2273)

Attributes

vertices	
center	
radius	
rotation	
apothem	
interior_angle	
exterior_angle	
circumcircle	
incircle	
angles	

property angles

Returns a dictionary with keys, the vertices of the Polygon, and values, the interior angle at each vertex.

Examples

```
>>> from sympy import RegularPolygon, Point
>>> r = RegularPolygon(Point(0, 0), 5, 3)
>>> r.angles
{Point2D(-5/2, -5*sqrt(3)/2): pi/3,
 Point2D(-5/2, 5*sqrt(3)/2): pi/3,
 Point2D(5, 0): pi/3}
```

property apothem

The inradius of the RegularPolygon.

The apothem/inradius is the radius of the inscribed circle.

Returns

apothem : number or instance of Basic

Examples

```
>>> from sympy import Symbol
>>> from sympy import RegularPolygon, Point
>>> radius = Symbol('r')
>>> rp = RegularPolygon(Point(0, 0), radius, 4)
>>> rp.apothem
sqrt(2)*r/2
```

See also:

sympy.geometry.line.Segment.length (page 2235), *sympy.geometry.ellipse.Circle.radius* (page 2271)

property area

Returns the area.

Examples

```
>>> from sympy import RegularPolygon
>>> square = RegularPolygon((0, 0), 1, 4)
>>> square.area
2
>>> _ == square.length**2
True
```

property args

Returns the center point, the radius, the number of sides, and the orientation angle.

Examples

```
>>> from sympy import RegularPolygon, Point
>>> r = RegularPolygon(Point(0, 0), 5, 3)
>>> r.args
(Point2D(0, 0), 5, 3, 0)
```

property center

The center of the RegularPolygon

This is also the center of the circumscribing circle.

Returns

center : Point

Examples

```
>>> from sympy import RegularPolygon, Point
>>> rp = RegularPolygon(Point(0, 0), 5, 4)
>>> rp.center
Point2D(0, 0)
```

See also:

sympy.geometry.point.Point (page 2202), *sympy.geometry.ellipse.Ellipse.center* (page 2255)

property centroid

The center of the RegularPolygon

This is also the center of the circumscribing circle.

Returns

center : Point

Examples

```
>>> from sympy import RegularPolygon, Point
>>> rp = RegularPolygon(Point(0, 0), 5, 4)
>>> rp.center
Point2D(0, 0)
```

See also:

sympy.geometry.point.Point (page 2202), *sympy.geometry.ellipse.Ellipse.center* (page 2255)

property circumcenter

Alias for center.

Examples

```
>>> from sympy import RegularPolygon, Point
>>> rp = RegularPolygon(Point(0, 0), 5, 4)
>>> rp.circumcenter
Point2D(0, 0)
```

property circumcircle

The circumcircle of the RegularPolygon.

Returns

circumcircle : Circle

Examples

```
>>> from sympy import RegularPolygon, Point
>>> rp = RegularPolygon(Point(0, 0), 4, 8)
>>> rp.circumcircle
Circle(Point2D(0, 0), 4)
```

See also:

circumcenter (page 2287), *sympy.geometry.ellipse.Circle* (page 2269)

property circumradius

Alias for radius.

Examples

```
>>> from sympy import Symbol
>>> from sympy import RegularPolygon, Point
>>> radius = Symbol('r')
>>> rp = RegularPolygon(Point(0, 0), radius, 4)
>>> rp.circumradius
r
```

encloses_point(*p*)

Return True if p is enclosed by (is inside of) self.

Parameters

p : Point

Returns

encloses_point : True, False or None

Notes

Being on the border of self is considered False.

The general Polygon.encloses_point method is called only if a point is not within or beyond the incircle or circumcircle, respectively.

Examples

```
>>> from sympy import RegularPolygon, S, Point, Symbol
>>> p = RegularPolygon((0, 0), 3, 4)
>>> p.encloses_point(Point(0, 0))
True
>>> r, R = p.inradius, p.circumradius
>>> p.encloses_point(Point((r + R)/2, 0))
True
>>> p.encloses_point(Point(R/2, R/2 + (R - r)/10))
False
>>> t = Symbol('t', real=True)
>>> p.encloses_point(p.arbitrary_point().subs(t, S.Half))
False
>>> p.encloses_point(Point(5, 5))
False
```

See also:

sympy.geometry.ellipse.Ellipse.encloses_point (page 2257)

property exterior_angle

Measure of the exterior angles.

Returns

exterior_angle : number

Examples

```
>>> from sympy import RegularPolygon, Point
>>> rp = RegularPolygon(Point(0, 0), 4, 8)
>>> rp.exterior_angle
pi/4
```

See also:

sympy.geometry.line.LinearEntity.angle_between (page 2216)

property incircle

The incircle of the RegularPolygon.

Returns

incircle : Circle

Examples

```
>>> from sympy import RegularPolygon, Point
>>> rp = RegularPolygon(Point(0, 0), 4, 7)
>>> rp.incircle
Circle(Point2D(0, 0), 4*cos(pi/7))
```

See also:

inradius (page 2290), *sympy.geometry.ellipse.Circle* (page 2269)

property inradius

Alias for apothem.

Examples

```
>>> from sympy import Symbol
>>> from sympy import RegularPolygon, Point
>>> radius = Symbol('r')
>>> rp = RegularPolygon(Point(0, 0), radius, 4)
>>> rp.inradius
sqrt(2)*r/2
```

property interior_angle

Measure of the interior angles.

Returns

interior_angle : number

Examples

```
>>> from sympy import RegularPolygon, Point
>>> rp = RegularPolygon(Point(0, 0), 4, 8)
>>> rp.interior_angle
3*pi/4
```

See also:

sympy.geometry.line.LinearEntity.angle_between (page 2216)

property length

Returns the length of the sides.

The half-length of the side and the apothem form two legs of a right triangle whose hypotenuse is the radius of the regular polygon.

Examples

```
>>> from sympy import RegularPolygon
>>> from sympy import sqrt
>>> s = square_in_unit_circle = RegularPolygon((0, 0), 1, 4)
>>> s.length
sqrt(2)
>>> sqrt((_/2)**2 + s.apothem**2) == s.radius
True
```

property radius

Radius of the RegularPolygon

This is also the radius of the circumscribing circle.

Returns

radius : number or instance of Basic

Examples

```
>>> from sympy import Symbol
>>> from sympy import RegularPolygon, Point
>>> radius = Symbol('r')
>>> rp = RegularPolygon(Point(0, 0), radius, 4)
>>> rp.radius
r
```

See also:

sympy.geometry.line.Segment.length (page 2235), *sympy.geometry.ellipse.Circle.radius* (page 2271)

reflect(*line*)

Override GeometryEntity.reflect since this is not made of only points.

Examples

```
>>> from sympy import RegularPolygon, Line
```

```
>>> RegularPolygon((0, 0), 1, 4).reflect(Line((0, 1), slope=-2))
RegularPolygon(Point2D(4/5, 2/5), -1, 4, atan(4/3))
```

rotate(*angle*, *pt=None*)

Override GeometryEntity.rotate to first rotate the RegularPolygon about its center.

```
>>> from sympy import Point, RegularPolygon, pi
>>> t = RegularPolygon(Point(1, 0), 1, 3)
>>> t.vertices[0] # vertex on x-axis
Point2D(2, 0)
>>> t.rotate(pi/2).vertices[0] # vertex on y axis now
Point2D(0, 2)
```

See also:

rotation (page 2292)

spin (page 2292)
Rotates a RegularPolygon in place

property rotation
CCW angle by which the RegularPolygon is rotated

Returns
rotation : number or instance of Basic

Examples

```
>>> from sympy import pi
>>> from sympy.abc import a
>>> from sympy import RegularPolygon, Point
>>> RegularPolygon(Point(0, 0), 3, 4, pi/4).rotation
pi/4
```

Numerical rotation angles are made canonical:

```
>>> RegularPolygon(Point(0, 0), 3, 4, a).rotation
a
>>> RegularPolygon(Point(0, 0), 3, 4, pi).rotation
0
```

scale(*x=1, y=1, pt=None*)
Override GeometryEntity.scale since it is the radius that must be scaled (if x == y) or else a new Polygon must be returned.

```
>>> from sympy import RegularPolygon
```

Symmetric scaling returns a RegularPolygon:

```
>>> RegularPolygon((0, 0), 1, 4).scale(2, 2)
RegularPolygon(Point2D(0, 0), 2, 4, 0)
```

Asymmetric scaling returns a kite as a Polygon:

```
>>> RegularPolygon((0, 0), 1, 4).scale(2, 1)
Polygon(Point2D(2, 0), Point2D(0, 1), Point2D(-2, 0), Point2D(0, -1))
```

spin(*angle*)
Increment *in place* the virtual Polygon's rotation by ccw angle.

See also: rotate method which moves the center.

```
>>> from sympy import Polygon, Point, pi
>>> r = Polygon(Point(0,0), 1, n=3)
>>> r.vertices[0]
Point2D(1, 0)
>>> r.spin(pi/6)
```

(continues on next page)

(continued from previous page)

```
>>> r.vertices[0]
Point2D(sqrt(3)/2, 1/2)
```

See also:

rotation (page 2292)

***rotate* (page 2291)**
Creates a copy of the RegularPolygon rotated about a Point

property vertices
The vertices of the RegularPolygon.

Returns
vertices : list

Each vertex is a Point.

Examples

```
>>> from sympy import RegularPolygon, Point
>>> rp = RegularPolygon(Point(0, 0), 5, 4)
>>> rp.vertices
[Point2D(5, 0), Point2D(0, 5), Point2D(-5, 0), Point2D(0, -5)]
```

See also:

sympy.geometry.point.Point (page 2202)

class sympy.geometry.polygon.**Triangle**(**args*, ***kwargs*)
A polygon with three vertices and three sides.

Parameters
points : sequence of Points

keyword: asa, sas, or sss to specify sides/angles of the triangle

Raises
GeometryError

If the number of vertices is not equal to three, or one of the vertices is not a Point, or a valid keyword is not given.

Examples

```
>>> from sympy import Triangle, Point
>>> Triangle(Point(0, 0), Point(4, 0), Point(4, 3))
Triangle(Point2D(0, 0), Point2D(4, 0), Point2D(4, 3))
```

Keywords sss, sas, or asa can be used to give the desired side lengths (in order) and interior angles (in degrees) that define the triangle:

```
>>> Triangle(sss=(3, 4, 5))
Triangle(Point2D(0, 0), Point2D(3, 0), Point2D(3, 4))
>>> Triangle(asa=(30, 1, 30))
Triangle(Point2D(0, 0), Point2D(1, 0), Point2D(1/2, sqrt(3)/6))
>>> Triangle(sas=(1, 45, 2))
Triangle(Point2D(0, 0), Point2D(2, 0), Point2D(sqrt(2)/2, sqrt(2)/2))
```

See also:

sympy.geometry.point.Point (page 2202), *Polygon* (page 2273)

Attributes

vertices	
altitudes	
orthocenter	
circumcenter	
circumradius	
circumcircle	
inradius	
incircle	
exradii	
medians	
medial	
nine_point_circle	

property altitudes

The altitudes of the triangle.

An altitude of a triangle is a segment through a vertex, perpendicular to the opposite side, with length being the height of the vertex measured from the line containing the side.

Returns

altitudes : dict

The dictionary consists of keys which are vertices and values which are Segments.

Examples

```
>>> from sympy import Point, Triangle
>>> p1, p2, p3 = Point(0, 0), Point(1, 0), Point(0, 1)
>>> t = Triangle(p1, p2, p3)
>>> t.altitudes[p1]
Segment2D(Point2D(0, 0), Point2D(1/2, 1/2))
```

See also:

sympy.geometry.point.Point (page 2202), *sympy.geometry.line.Segment.length* (page 2235)

bisectors()

The angle bisectors of the triangle.

An angle bisector of a triangle is a straight line through a vertex which cuts the corresponding angle in half.

Returns

bisectors : dict

Each key is a vertex (Point) and each value is the corresponding bisector (Segment).

Examples

```
>>> from sympy import Point, Triangle, Segment
>>> p1, p2, p3 = Point(0, 0), Point(1, 0), Point(0, 1)
>>> t = Triangle(p1, p2, p3)
>>> from sympy import sqrt
>>> t.bisectors()[p2] == Segment(Point(1, 0), Point(0, sqrt(2) - 1))
True
```

See also:

sympy.geometry.point.Point (page 2202), *sympy.geometry.line.Segment* (page 2233)

property circumcenter

The circumcenter of the triangle

The circumcenter is the center of the circumcircle.

Returns

circumcenter : Point

Examples

```
>>> from sympy import Point, Triangle
>>> p1, p2, p3 = Point(0, 0), Point(1, 0), Point(0, 1)
>>> t = Triangle(p1, p2, p3)
>>> t.circumcenter
Point2D(1/2, 1/2)
```

See also:

sympy.geometry.point.Point (page 2202)

property circumcircle

The circle which passes through the three vertices of the triangle.

Returns

circumcircle : Circle

Examples

```
>>> from sympy import Point, Triangle
>>> p1, p2, p3 = Point(0, 0), Point(1, 0), Point(0, 1)
>>> t = Triangle(p1, p2, p3)
>>> t.circumcircle
Circle(Point2D(1/2, 1/2), sqrt(2)/2)
```

See also:

sympy.geometry.ellipse.Circle (page 2269)

property circumradius

The radius of the circumcircle of the triangle.

Returns

circumradius : number of Basic instance

Examples

```
>>> from sympy import Symbol
>>> from sympy import Point, Triangle
>>> a = Symbol('a')
>>> p1, p2, p3 = Point(0, 0), Point(1, 0), Point(0, a)
>>> t = Triangle(p1, p2, p3)
>>> t.circumradius
sqrt(a**2/4 + 1/4)
```

See also:

sympy.geometry.ellipse.Circle.radius (page 2271)

property eulerline

The Euler line of the triangle.

The line which passes through circumcenter, centroid and orthocenter.

Returns

eulerline : Line (or Point for equilateral triangles in which case all centers coincide)

Examples

```
>>> from sympy import Point, Triangle
>>> p1, p2, p3 = Point(0, 0), Point(1, 0), Point(0, 1)
>>> t = Triangle(p1, p2, p3)
>>> t.eulerline
Line2D(Point2D(0, 0), Point2D(1/2, 1/2))
```

property excenters

Excenters of the triangle.

An excenter is the center of a circle that is tangent to a side of the triangle and the extensions of the other two sides.

Returns

excenters : dict

Examples

The excenters are keyed to the side of the triangle to which their corresponding excircle is tangent: The center is keyed, e.g. the excenter of a circle touching side 0 is:

```
>>> from sympy import Point, Triangle
>>> p1, p2, p3 = Point(0, 0), Point(6, 0), Point(0, 2)
>>> t = Triangle(p1, p2, p3)
>>> t.excenters[t.sides[0]]
Point2D(12*sqrt(10), 2/3 + sqrt(10)/3)
```

See also:

sympy.geometry.polygon.Triangle.exradii (page 2297)

References

[R518]

property exradii

The radius of excircles of a triangle.

An excircle of the triangle is a circle lying outside the triangle, tangent to one of its sides and tangent to the extensions of the other two.

Returns

exradii : dict

Examples

The exradius touches the side of the triangle to which it is keyed, e.g. the exradius touching side 2 is:

```
>>> from sympy import Point, Triangle
>>> p1, p2, p3 = Point(0, 0), Point(6, 0), Point(0, 2)
>>> t = Triangle(p1, p2, p3)
>>> t.exradii[t.sides[2]]
-2 + sqrt(10)
```

See also:

sympy.geometry.polygon.Triangle.inradius (page 2298)

References

[R519], [R520]

property incenter

The center of the incircle.

The incircle is the circle which lies inside the triangle and touches all three sides.

Returns

incenter : Point

Examples

```
>>> from sympy import Point, Triangle
>>> p1, p2, p3 = Point(0, 0), Point(1, 0), Point(0, 1)
>>> t = Triangle(p1, p2, p3)
>>> t.incenter
Point2D(1 - sqrt(2)/2, 1 - sqrt(2)/2)
```

See also:

incircle (page 2298), *sympy.geometry.point.Point* (page 2202)

property incircle

The incircle of the triangle.

The incircle is the circle which lies inside the triangle and touches all three sides.

Returns

incircle : Circle

Examples

```
>>> from sympy import Point, Triangle
>>> p1, p2, p3 = Point(0, 0), Point(2, 0), Point(0, 2)
>>> t = Triangle(p1, p2, p3)
>>> t.incircle
Circle(Point2D(2 - sqrt(2), 2 - sqrt(2)), 2 - sqrt(2))
```

See also:

sympy.geometry.ellipse.Circle (page 2269)

property inradius

The radius of the incircle.

Returns

inradius : number of Basic instance

Examples

```
>>> from sympy import Point, Triangle
>>> p1, p2, p3 = Point(0, 0), Point(4, 0), Point(0, 3)
>>> t = Triangle(p1, p2, p3)
>>> t.inradius
1
```

See also:

incircle (page 2298), *sympy.geometry.ellipse.Circle.radius* (page 2271)

is_equilateral()

Are all the sides the same length?

Returns

is_equilateral : boolean

Examples

```
>>> from sympy import Triangle, Point
>>> t1 = Triangle(Point(0, 0), Point(4, 0), Point(4, 3))
>>> t1.is_equilateral()
False
```

```
>>> from sympy import sqrt
>>> t2 = Triangle(Point(0, 0), Point(10, 0), Point(5, 5*sqrt(3)))
>>> t2.is_equilateral()
True
```

See also:

sympy.geometry.entity.GeometryEntity.is_similar (page 2195), *RegularPolygon* (page 2284), *is_isosceles* (page 2299), *is_right* (page 2299), *is_scalene* (page 2300)

is_isosceles()

Are two or more of the sides the same length?

Returns

is_isosceles : boolean

Examples

```
>>> from sympy import Triangle, Point
>>> t1 = Triangle(Point(0, 0), Point(4, 0), Point(2, 4))
>>> t1.is_isosceles()
True
```

See also:

is_equilateral (page 2299), *is_right* (page 2299), *is_scalene* (page 2300)

is_right()

Is the triangle right-angled.

Returns

is_right : boolean

Examples

```
>>> from sympy import Triangle, Point
>>> t1 = Triangle(Point(0, 0), Point(4, 0), Point(4, 3))
>>> t1.is_right()
True
```

See also:

sympy.geometry.line.LinearEntity.is_perpendicular (page 2221), *is_equilateral* (page 2299), *is_isosceles* (page 2299), *is_scalene* (page 2300)

is_scalene()

Are all the sides of the triangle of different lengths?

Returns

is_scalene : boolean

Examples

```
>>> from sympy import Triangle, Point
>>> t1 = Triangle(Point(0, 0), Point(4, 0), Point(1, 4))
>>> t1.is_scalene()
True
```

See also:

is_equilateral (page 2299), *is_isosceles* (page 2299), *is_right* (page 2299)

is_similar(*t2*)

Is another triangle similar to this one.

Two triangles are similar if one can be uniformly scaled to the other.

Parameters

other: Triangle

Returns

is_similar : boolean

Examples

```
>>> from sympy import Triangle, Point
>>> t1 = Triangle(Point(0, 0), Point(4, 0), Point(4, 3))
>>> t2 = Triangle(Point(0, 0), Point(-4, 0), Point(-4, -3))
>>> t1.is_similar(t2)
True
```

```
>>> t2 = Triangle(Point(0, 0), Point(-4, 0), Point(-4, -4))
>>> t1.is_similar(t2)
False
```

See also:

sympy.geometry.entity.GeometryEntity.is_similar (page 2195)

property medial

The medial triangle of the triangle.

The triangle which is formed from the midpoints of the three sides.

Returns

medial : Triangle

Examples

```
>>> from sympy import Point, Triangle
>>> p1, p2, p3 = Point(0, 0), Point(1, 0), Point(0, 1)
>>> t = Triangle(p1, p2, p3)
>>> t.medial
Triangle(Point2D(1/2, 0), Point2D(1/2, 1/2), Point2D(0, 1/2))
```

See also:

sympy.geometry.line.Segment.midpoint (page 2235)

property medians

The medians of the triangle.

A median of a triangle is a straight line through a vertex and the midpoint of the opposite side, and divides the triangle into two equal areas.

Returns

medians : dict

Each key is a vertex (Point) and each value is the median (Segment) at that point.

Examples

```
>>> from sympy import Point, Triangle
>>> p1, p2, p3 = Point(0, 0), Point(1, 0), Point(0, 1)
>>> t = Triangle(p1, p2, p3)
>>> t.medians[p1]
Segment2D(Point2D(0, 0), Point2D(1/2, 1/2))
```

See also:

sympy.geometry.point.Point.midpoint (page 2207), *sympy.geometry.line.Segment.midpoint* (page 2235)

property nine_point_circle

The nine-point circle of the triangle.

Nine-point circle is the circumcircle of the medial triangle, which passes through the feet of altitudes and the middle points of segments connecting the vertices and the orthocenter.

Returns
nine_point_circle : Circle

Examples

```
>>> from sympy import Point, Triangle
>>> p1, p2, p3 = Point(0, 0), Point(1, 0), Point(0, 1)
>>> t = Triangle(p1, p2, p3)
>>> t.nine_point_circle
Circle(Point2D(1/4, 1/4), sqrt(2)/4)
```

See also:

sympy.geometry.line.Segment.midpoint (page 2235), *sympy.geometry.polygon.Triangle.medial* (page 2301), *sympy.geometry.polygon.Triangle.orthocenter* (page 2302)

property orthocenter

The orthocenter of the triangle.

The orthocenter is the intersection of the altitudes of a triangle. It may lie inside, outside or on the triangle.

Returns
orthocenter : Point

Examples

```
>>> from sympy import Point, Triangle
>>> p1, p2, p3 = Point(0, 0), Point(1, 0), Point(0, 1)
>>> t = Triangle(p1, p2, p3)
>>> t.orthocenter
Point2D(0, 0)
```

See also:

sympy.geometry.point.Point (page 2202)

property vertices

The triangle's vertices

Returns

vertices : tuple

Each element in the tuple is a Point

Examples

```
>>> from sympy import Triangle, Point
>>> t = Triangle(Point(0, 0), Point(4, 0), Point(4, 3))
>>> t.vertices
(Point2D(0, 0), Point2D(4, 0), Point2D(4, 3))
```

See also:

sympy.geometry.point.Point (page 2202)

Plane

class sympy.geometry.plane.**Plane**(*p1, a=None, b=None, **kwargs*)

A plane is a flat, two-dimensional surface. A plane is the two-dimensional analogue of a point (zero-dimensions), a line (one-dimension) and a solid (three-dimensions). A plane can generally be constructed by two types of inputs. They are three non-collinear points and a point and the plane's normal vector.

Examples

```
>>> from sympy import Plane, Point3D
>>> Plane(Point3D(1, 1, 1), Point3D(2, 3, 4), Point3D(2, 2, 2))
Plane(Point3D(1, 1, 1), (-1, 2, -1))
>>> Plane((1, 1, 1), (2, 3, 4), (2, 2, 2))
Plane(Point3D(1, 1, 1), (-1, 2, -1))
>>> Plane(Point3D(1, 1, 1), normal_vector=(1,4,7))
Plane(Point3D(1, 1, 1), (1, 4, 7))
```

Attributes

p1	
normal_vector	

angle_between(*o*)

Angle between the plane and other geometric entity.

Parameters
LinearEntity3D, Plane.

Returns
angle : angle in radians

Notes

This method accepts only 3D entities as it's parameter, but if you want to calculate the angle between a 2D entity and a plane you should first convert to a 3D entity by projecting onto a desired plane and then proceed to calculate the angle.

Examples

```
>>> from sympy import Point3D, Line3D, Plane
>>> a = Plane(Point3D(1, 2, 2), normal_vector=(1, 2, 3))
>>> b = Line3D(Point3D(1, 3, 4), Point3D(2, 2, 2))
>>> a.angle_between(b)
-asin(sqrt(21)/6)
```

arbitrary_point(*u=None, v=None*)

Returns an arbitrary point on the Plane. If given two parameters, the point ranges over the entire plane. If given 1 or no parameters, returns a point with one parameter which, when varying from 0 to 2*pi, moves the point in a circle of radius 1 about p1 of the Plane.

Returns
Point3D

Examples

```
>>> from sympy import Plane, Ray
>>> from sympy.abc import u, v, t, r
>>> p = Plane((1, 1, 1), normal_vector=(1, 0, 0))
>>> p.arbitrary_point(u, v)
Point3D(1, u + 1, v + 1)
>>> p.arbitrary_point(t)
Point3D(1, cos(t) + 1, sin(t) + 1)
```

While arbitrary values of u and v can move the point anywhere in the plane, the single-parameter point can be used to construct a ray whose arbitrary point can be located at angle t and radius r from p.p1:

```
>>> Ray(p.p1, _).arbitrary_point(r)
Point3D(1, r*cos(t) + 1, r*sin(t) + 1)
```

static are_concurrent(**planes*)

Is a sequence of Planes concurrent?

Two or more Planes are concurrent if their intersections are a common line.

Parameters

planes: list

Returns

Boolean

Examples

```
>>> from sympy import Plane, Point3D
>>> a = Plane(Point3D(5, 0, 0), normal_vector=(1, -1, 1))
>>> b = Plane(Point3D(0, -2, 0), normal_vector=(3, 1, 1))
>>> c = Plane(Point3D(0, -1, 0), normal_vector=(5, -1, 9))
>>> Plane.are_concurrent(a, b)
True
>>> Plane.are_concurrent(a, b, c)
False
```

distance(*o*)

Distance between the plane and another geometric entity.

Parameters

Point3D, LinearEntity3D, Plane.

Returns

distance

Notes

This method accepts only 3D entities as it's parameter, but if you want to calculate the distance between a 2D entity and a plane you should first convert to a 3D entity by projecting onto a desired plane and then proceed to calculate the distance.

Examples

```
>>> from sympy import Point3D, Line3D, Plane
>>> a = Plane(Point3D(1, 1, 1), normal_vector=(1, 1, 1))
>>> b = Point3D(1, 2, 3)
>>> a.distance(b)
sqrt(3)
>>> c = Line3D(Point3D(2, 3, 1), Point3D(1, 2, 2))
>>> a.distance(c)
0
```

equals(*o*)

Returns True if self and o are the same mathematical entities.

Examples

```
>>> from sympy import Plane, Point3D
>>> a = Plane(Point3D(1, 2, 3), normal_vector=(1, 1, 1))
>>> b = Plane(Point3D(1, 2, 3), normal_vector=(2, 2, 2))
>>> c = Plane(Point3D(1, 2, 3), normal_vector=(-1, 4, 6))
>>> a.equals(a)
True
>>> a.equals(b)
True
>>> a.equals(c)
False
```

equation(*x=None, y=None, z=None*)

The equation of the Plane.

Examples

```
>>> from sympy import Point3D, Plane
>>> a = Plane(Point3D(1, 1, 2), Point3D(2, 4, 7), Point3D(3, 5, 1))
>>> a.equation()
-23*x + 11*y - 2*z + 16
>>> a = Plane(Point3D(1, 4, 2), normal_vector=(6, 6, 6))
>>> a.equation()
6*x + 6*y + 6*z - 42
```

intersection(*o*)

The intersection with other geometrical entity.

Parameters
  Point, Point3D, LinearEntity, LinearEntity3D, Plane

Returns
  List

Examples

```
>>> from sympy import Point3D, Line3D, Plane
>>> a = Plane(Point3D(1, 2, 3), normal_vector=(1, 1, 1))
>>> b = Point3D(1, 2, 3)
>>> a.intersection(b)
[Point3D(1, 2, 3)]
>>> c = Line3D(Point3D(1, 4, 7), Point3D(2, 2, 2))
>>> a.intersection(c)
[Point3D(2, 2, 2)]
>>> d = Plane(Point3D(6, 0, 0), normal_vector=(2, -5, 3))
```

(continues on next page)

(continued from previous page)

```
>>> e = Plane(Point3D(2, 0, 0), normal_vector=(3, 4, -3))
>>> d.intersection(e)
[Line3D(Point3D(78/23, -24/23, 0), Point3D(147/23, 321/23, 23))]
```

is_coplanar(*o*)

Returns True if *o* is coplanar with self, else False.

Examples

```
>>> from sympy import Plane
>>> o = (0, 0, 0)
>>> p = Plane(o, (1, 1, 1))
>>> p2 = Plane(o, (2, 2, 2))
>>> p == p2
False
>>> p.is_coplanar(p2)
True
```

is_parallel(*l*)

Is the given geometric entity parallel to the plane?

Parameters

LinearEntity3D or Plane

Returns

Boolean

Examples

```
>>> from sympy import Plane, Point3D
>>> a = Plane(Point3D(1,4,6), normal_vector=(2, 4, 6))
>>> b = Plane(Point3D(3,1,3), normal_vector=(4, 8, 12))
>>> a.is_parallel(b)
True
```

is_perpendicular(*l*)

is the given geometric entity perpendicualar to the given plane?

Parameters

LinearEntity3D or Plane

Returns

Boolean

Examples

```
>>> from sympy import Plane, Point3D
>>> a = Plane(Point3D(1,4,6), normal_vector=(2, 4, 6))
>>> b = Plane(Point3D(2, 2, 2), normal_vector=(-1, 2, -1))
>>> a.is_perpendicular(b)
True
```

property normal_vector

Normal vector of the given plane.

Examples

```
>>> from sympy import Point3D, Plane
>>> a = Plane(Point3D(1, 1, 1), Point3D(2, 3, 4), Point3D(2, 2, 2))
>>> a.normal_vector
(-1, 2, -1)
>>> a = Plane(Point3D(1, 1, 1), normal_vector=(1, 4, 7))
>>> a.normal_vector
(1, 4, 7)
```

property p1

The only defining point of the plane. Others can be obtained from the arbitrary_point method.

Examples

```
>>> from sympy import Point3D, Plane
>>> a = Plane(Point3D(1, 1, 1), Point3D(2, 3, 4), Point3D(2, 2, 2))
>>> a.p1
Point3D(1, 1, 1)
```

See also:

sympy.geometry.point.Point3D (page 2211)

parallel_plane(*pt*)

Plane parallel to the given plane and passing through the point pt.

Parameters
pt: Point3D

Returns
Plane

Examples

```
>>> from sympy import Plane, Point3D
>>> a = Plane(Point3D(1, 4, 6), normal_vector=(2, 4, 6))
>>> a.parallel_plane(Point3D(2, 3, 5))
Plane(Point3D(2, 3, 5), (2, 4, 6))
```

parameter_value(*other*, *u*, *v=None*)

Return the parameter(s) corresponding to the given point.

Examples

```
>>> from sympy import pi, Plane
>>> from sympy.abc import t, u, v
>>> p = Plane((2, 0, 0), (0, 0, 1), (0, 1, 0))
```

By default, the parameter value returned defines a point that is a distance of 1 from the Plane's p1 value and in line with the given point:

```
>>> on_circle = p.arbitrary_point(t).subs(t, pi/4)
>>> on_circle.distance(p.p1)
1
>>> p.parameter_value(on_circle, t)
{t: pi/4}
```

Moving the point twice as far from p1 does not change the parameter value:

```
>>> off_circle = p.p1 + (on_circle - p.p1)*2
>>> off_circle.distance(p.p1)
2
>>> p.parameter_value(off_circle, t)
{t: pi/4}
```

If the 2-value parameter is desired, supply the two parameter symbols and a replacement dictionary will be returned:

```
>>> p.parameter_value(on_circle, u, v)
{u: sqrt(10)/10, v: sqrt(10)/30}
>>> p.parameter_value(off_circle, u, v)
{u: sqrt(10)/5, v: sqrt(10)/15}
```

perpendicular_line(*pt*)

A line perpendicular to the given plane.

Parameters
pt: Point3D

Returns
Line3D

Examples

```
>>> from sympy import Plane, Point3D
>>> a = Plane(Point3D(1,4,6), normal_vector=(2, 4, 6))
>>> a.perpendicular_line(Point3D(9, 8, 7))
Line3D(Point3D(9, 8, 7), Point3D(11, 12, 13))
```

perpendicular_plane(**pts*)

Return a perpendicular passing through the given points. If the direction ratio between the points is the same as the Plane's normal vector then, to select from the infinite number of possible planes, a third point will be chosen on the z-axis (or the y-axis if the normal vector is already parallel to the z-axis). If less than two points are given they will be supplied as follows: if no point is given then pt1 will be self.p1; if a second point is not given it will be a point through pt1 on a line parallel to the z-axis (if the normal is not already the z-axis, otherwise on the line parallel to the y-axis).

Parameters
pts: 0, 1 or 2 Point3D

Returns
Plane

Examples

```
>>> from sympy import Plane, Point3D
>>> a, b = Point3D(0, 0, 0), Point3D(0, 1, 0)
>>> Z = (0, 0, 1)
>>> p = Plane(a, normal_vector=Z)
>>> p.perpendicular_plane(a, b)
Plane(Point3D(0, 0, 0), (1, 0, 0))
```

projection(*pt*)

Project the given point onto the plane along the plane normal.

Parameters
Point or Point3D

Returns
Point3D

Examples

```
>>> from sympy import Plane, Point3D
>>> A = Plane(Point3D(1, 1, 2), normal_vector=(1, 1, 1))
```

The projection is along the normal vector direction, not the z axis, so (1, 1) does not project to (1, 1, 2) on the plane A:

```
>>> b = Point3D(1, 1)
>>> A.projection(b)
Point3D(5/3, 5/3, 2/3)
```

(continues on next page)

(continued from previous page)

```
>>> _ in A
True
```

But the point (1, 1, 2) projects to (1, 1) on the XY-plane:

```
>>> XY = Plane((0, 0, 0), (0, 0, 1))
>>> XY.projection((1, 1, 2))
Point3D(1, 1, 0)
```

projection_line(*line*)

Project the given line onto the plane through the normal plane containing the line.

Parameters

LinearEntity or LinearEntity3D

Returns

Point3D, Line3D, Ray3D or Segment3D

Notes

For the interaction between 2D and 3D lines(segments, rays), you should convert the line to 3D by using this method. For example for finding the intersection between a 2D and a 3D line, convert the 2D line to a 3D line by projecting it on a required plane and then proceed to find the intersection between those lines.

Examples

```
>>> from sympy import Plane, Line, Line3D, Point3D
>>> a = Plane(Point3D(1, 1, 1), normal_vector=(1, 1, 1))
>>> b = Line(Point3D(1, 1), Point3D(2, 2))
>>> a.projection_line(b)
Line3D(Point3D(4/3, 4/3, 1/3), Point3D(5/3, 5/3, -1/3))
>>> c = Line3D(Point3D(1, 1, 1), Point3D(2, 2, 2))
>>> a.projection_line(c)
Point3D(1, 1, 1)
```

random_point(*seed=None*)

Returns a random point on the Plane.

Returns

Point3D

Examples

```
>>> from sympy import Plane
>>> p = Plane((1, 0, 0), normal_vector=(0, 1, 0))
>>> r = p.random_point(seed=42)  # seed value is optional
>>> r.n(3)
Point3D(2.29, 0, -1.35)
```

The random point can be moved to lie on the circle of radius 1 centered on p1:

```
>>> c = p.p1 + (r - p.p1).unit
>>> c.distance(p.p1).equals(1)
True
```

Holonomic

The *holonomic* (page 2312) module is intended to deal with holonomic functions along with various operations on them like addition, multiplication, composition, integration and differentiation. The module also implements various kinds of conversions such as converting holonomic functions to a different form and the other way around.

Contents

About Holonomic Functions

This text aims to explain holonomic functions. We assume you have a basic idea of Differential equations and Abstract algebra.

Definition

Holonomic function is a very general type of special function that includes lots of simple known functions as its special cases. In fact the more known hypergeometric function and Meijer G-function are also a special case of it.

A function is called holonomic if it's a solution to an ordinary differential equation having polynomial coefficients only. Since the general solution of a differential equation consists of a family of functions rather than a single function, holonomic functions are usually defined by a set of initial conditions along with the differential equation.

Let K be a field of characteristic `0`. For example, K can be `QQ` or `RR`. A function $f(x)$ will be holonomic if there exists polynomials $p_0, p_1, p_2, ...p_r \in K[x]$ such that

$$p_0 \cdot f(x) + p_1 \cdot f^{(1)}(x) + p_2 \cdot f^{(2)}(x) + ... + p_r \cdot f^{(r)}(x) = 0$$

This differential equation can also be written as $L \cdot f(x) = 0$ where

$$L = p_0 + p_1 \cdot D + p_2 \cdot D^2 + ...p_r \cdot D^r$$

Here D is the Differential Operator and L is called the annihilator of the function.

A unique holonomic function can be defined from the annihilator and a set of initial conditions. For instance:

$$f(x) = \exp(x) : L = D - 1,\; f(0) = 1$$
$$f(x) = \sin(x) : L = D^2 + 1,\; f(0) = 0, f'(0) = 1$$

Other fundamental functions such as $\cos(x)$, $\log(x)$, bessel functions etc. are also holonomic.

The family of holonomic functions is closed under addition, multiplication, integration, composition. This means if two functions are given are holonomic, then the function resulting on applying these operation on them will also be holonomic.

References

https://en.wikipedia.org/wiki/Holonomic_function

Representation of holonomic functions in SymPy

Class *DifferentialOperator* (page 2315) is used to represent the annihilator but we create differential operators easily using the function *DifferentialOperators()* (page 2316). Class *HolonomicFunction* (page 2314) represents a holonomic function.

Let's explain this with an example:

Take $\sin(x)$ for instance, the differential equation satisfied by it is $y^{(2)}(x)+y(x) = 0$. By definition we conclude it is a holonomic function. The general solution of this ODE is $C_1 \cdot \sin(x)+C_2 \cdot \cos(x)$ but to get $\sin(x)$ we need to provide initial conditions i.e. $y(0) = 0, y^{(1)}(0) = 1$.

To represent the same in this module one needs to provide the differential equation in the form of annihilator. Basically a differential operator is an operator on functions that differentiates them. So $D^n \cdot y(x) = y^{(n)}(x)$ where $y^{(n)}(x)$ denotes n times differentiation of $y(x)$ with respect to x.

So the differential equation can also be written as $D^2 \cdot y(x) + y(x) = 0$ or $(D^2 + 1) \cdot y(x) = 0$. The part left of $y(x)$ is the annihilator i.e. $D^2 + 1$.

So this is how one will represent $\sin(x)$ as a Holonomic Function:

```
>>> from sympy.holonomic import DifferentialOperators, HolonomicFunction
>>> from sympy.abc import x
>>> from sympy import ZZ
>>> R, D = DifferentialOperators(ZZ.old_poly_ring(x), 'D')
>>> HolonomicFunction(D**2 + 1, x, 0, [0, 1])
HolonomicFunction((1) + (1)*D**2, x, 0, [0, 1])
```

The polynomial coefficients will be members of the ring `ZZ[x]` in the example. The `D` operator returned by the function *DifferentialOperators()* (page 2316) can be used to create annihilators just like SymPy expressions. We currently use the older implementations of rings in SymPy for priority mechanism.

HolonomicFunction

class sympy.holonomic.holonomic.**HolonomicFunction**(*annihilator, x, x0=0, y0=None*)

A Holonomic Function is a solution to a linear homogeneous ordinary differential equation with polynomial coefficients. This differential equation can also be represented by an annihilator i.e. a Differential Operator L such that $L.f = 0$. For uniqueness of these functions, initial conditions can also be provided along with the annihilator.

Explanation

Holonomic functions have closure properties and thus forms a ring. Given two Holonomic Functions f and g, their sum, product, integral and derivative is also a Holonomic Function.

For ordinary points initial condition should be a vector of values of the derivatives i.e. $[y(x_0), y'(x_0), y''(x_0)...]$.

For regular singular points initial conditions can also be provided in this format: $s0 : [C_0, C_1, ...], s1 : [C_0^1, C_1^1, ...], ...$ where s0, s1, … are the roots of indicial equation and vectors $[C_0, C_1, ...], [C_0^0, C_1^0, ...], ...$ are the corresponding initial terms of the associated power series. See Examples below.

Examples

```
>>> from sympy.holonomic.holonomic import HolonomicFunction,␣
↪DifferentialOperators
>>> from sympy import QQ
>>> from sympy import symbols, S
>>> x = symbols('x')
>>> R, Dx = DifferentialOperators(QQ.old_poly_ring(x),'Dx')
```

```
>>> p = HolonomicFunction(Dx - 1, x, 0, [1])  # e^x
>>> q = HolonomicFunction(Dx**2 + 1, x, 0, [0, 1])  # sin(x)
```

```
>>> p + q  # annihilator of e^x + sin(x)
HolonomicFunction((-1) + (1)*Dx + (-1)*Dx**2 + (1)*Dx**3, x, 0, [1, 2,␣
↪1])
```

```
>>> p * q  # annihilator of e^x * sin(x)
HolonomicFunction((2) + (-2)*Dx + (1)*Dx**2, x, 0, [0, 1])
```

An example of initial conditions for regular singular points, the indicial equation has only one root $1/2$.

```
>>> HolonomicFunction(-S(1)/2 + x*Dx, x, 0, {S(1)/2: [1]})
HolonomicFunction((-1/2) + (x)*Dx, x, 0, {1/2: [1]})
```

```
>>> HolonomicFunction(-S(1)/2 + x*Dx, x, 0, {S(1)/2: [1]}).to_expr()
sqrt(x)
```

To plot a Holonomic Function, one can use $.evalf()$ for numerical computation. Here's an example on $sin(x)**2/x$ using numpy and matplotlib.

```
>>> import sympy.holonomic
>>> from sympy import var, sin
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> var("x")
>>> r = np.linspace(1, 5, 100)
>>> y = sympy.holonomic.expr_to_holonomic(sin(x)**2/x, x0=1).evalf(r)
>>> plt.plot(r, y, label="holonomic function")
>>> plt.show()
```

DifferentialOperator

class sympy.holonomic.holonomic.**DifferentialOperator**(*list_of_poly, parent*)

Differential Operators are elements of Weyl Algebra. The Operators are defined by a list of polynomials in the base ring and the parent ring of the Operator i.e. the algebra it belongs to.

Explanation

Takes a list of polynomials for each power of Dx and the parent ring which must be an instance of DifferentialOperatorAlgebra.

A Differential Operator can be created easily using the operator Dx. See examples below.

Examples

```
>>> from sympy.holonomic.holonomic import DifferentialOperator,␣
↪DifferentialOperators
>>> from sympy import ZZ
>>> from sympy import symbols
>>> x = symbols('x')
>>> R, Dx = DifferentialOperators(ZZ.old_poly_ring(x),'Dx')
```

```
>>> DifferentialOperator([0, 1, x**2], R)
(1)*Dx + (x**2)*Dx**2
```

```
>>> (x*Dx*x + 1 - Dx**2)**2
(2*x**2 + 2*x + 1) + (4*x**3 + 2*x**2 - 4)*Dx + (x**4 - 6*x - 2)*Dx**2 +␣
↪(-2*x**2)*Dx**3 + (1)*Dx**4
```

See also:

DifferentialOperatorAlgebra (page 2316)

is_singular(*x0*)

Checks if the differential equation is singular at x0.

DifferentialOperators

sympy.holonomic.holonomic.**DifferentialOperators**(*base, generator*)

This function is used to create annihilators using Dx.

Parameters

base:

Base polynomial ring for the algebra. The base polynomial ring is the ring of polynomials in x that will appear as coefficients in the operators.

generator:

Generator of the algebra which can be either a noncommutative Symbol or a string. e.g. “Dx” or “D”.

Explanation

Returns an Algebra of Differential Operators also called Weyl Algebra and the operator for differentiation i.e. the Dx operator.

Examples

```
>>> from sympy import ZZ
>>> from sympy.abc import x
>>> from sympy.holonomic.holonomic import DifferentialOperators
>>> R, Dx = DifferentialOperators(ZZ.old_poly_ring(x), 'Dx')
>>> R
Univariate Differential Operator Algebra in intermediate Dx over the␣
↪base ring ZZ[x]
>>> Dx*x
(1) + (x)*Dx
```

DifferentialOperatorAlgebra

class sympy.holonomic.holonomic.**DifferentialOperatorAlgebra**(*base, generator*)

An Ore Algebra is a set of noncommutative polynomials in the intermediate Dx and coefficients in a base polynomial ring A. It follows the commutation rule:

$$Dxa = \sigma(a)Dx + \delta(a)$$

for $a \subset A$.

Where $\sigma : A \Rightarrow A$ is an endomorphism and $\delta : A \rightarrow A$ is a skew-derivation i.e. $\delta(ab) = \delta(a)b + \sigma(a)\delta(b)$.

If one takes the sigma as identity map and delta as the standard derivation then it becomes the algebra of Differential Operators also called a Weyl Algebra i.e. an algebra whose elements are Differential Operators.

This class represents a Weyl Algebra and serves as the parent ring for Differential Operators.