Notes

This is an abstract class and is not meant to be instantiated.

See also:

sympy.geometry.entity.GeometryEntity (page 2194)

Attributes

p1	
p2	
coefficients	
slope	
points	

property bounds

Return a tuple (xmin, ymin, xmax, ymax) representing the bounding rectangle for the geometric figure.

perpendicular_line(*p*)

Create a new Line perpendicular to this linear entity which passes through the point *p*.

Parameters
p : Point

Returns
line : Line

Examples

```
>>> from sympy import Point, Line
>>> p1, p2, p3 = Point(0, 0), Point(2, 3), Point(-2, 2)
>>> L = Line(p1, p2)
>>> P = L.perpendicular_line(p3); P
Line2D(Point2D(-2, 2), Point2D(-5, 4))
>>> L.is_perpendicular(P)
True
```

In 2D, the first point of the perpendicular line is the point through which was required to pass; the second point is arbitrarily chosen. To get a line that explicitly uses a point in the line, create a line from the perpendicular segment from the line to the point:

```
>>> Line(L.perpendicular_segment(p3))
Line2D(Point2D(-2, 2), Point2D(4/13, 6/13))
```

See also:

sympy.geometry.line.LinearEntity.is_perpendicular (page 2221), *perpendicular_segment* (page 2224)

property slope

The slope of this linear entity, or infinity if vertical.

Returns

slope : number or SymPy expression

Examples

```
>>> from sympy import Point, Line
>>> p1, p2 = Point(0, 0), Point(3, 5)
>>> l1 = Line(p1, p2)
>>> l1.slope
5/3
```

```
>>> p3 = Point(0, 4)
>>> l2 = Line(p1, p3)
>>> l2.slope
oo
```

See also:

coefficients (page 2239)

class sympy.geometry.line.**Line2D**(*p1*, *pt=None*, *slope=None*, ***kwargs*)

An infinite line in space 2D.

A line is declared with two distinct points or a point and slope as defined using keyword *slope*.

Parameters

p1 : Point

pt : Point

slope : SymPy expression

Examples

```
>>> from sympy import Line, Segment, Point
>>> L = Line(Point(2,3), Point(3,5))
>>> L
Line2D(Point2D(2, 3), Point2D(3, 5))
>>> L.points
(Point2D(2, 3), Point2D(3, 5))
>>> L.equation()
-2*x + y + 1
>>> L.coefficients
(-2, 1, 1)
```

Instantiate with keyword `slope`:

```
>>> Line(Point(0, 0), slope=0)
Line2D(Point2D(0, 0), Point2D(1, 0))
```

Instantiate with another linear object

```
>>> s = Segment((0, 0), (0, 1))
>>> Line(s).equation()
x
```

See also:

sympy.geometry.point.Point (page 2202)

property coefficients

The coefficients (a, b, c) for $ax + by + c = 0$.

Examples

```
>>> from sympy import Point, Line
>>> from sympy.abc import x, y
>>> p1, p2 = Point(0, 0), Point(5, 3)
>>> l = Line(p1, p2)
>>> l.coefficients
(-3, 5, 0)
```

```
>>> p3 = Point(x, y)
>>> l2 = Line(p1, p3)
>>> l2.coefficients
(-y, x, 0)
```

See also:

sympy.geometry.line.Line2D.equation (page 2239)

equation(*x*='x', *y*='y')

The equation of the line: ax + by + c.

Parameters

x : str, optional

The name to use for the x-axis, default value is ‘x’.

y : str, optional

The name to use for the y-axis, default value is ‘y’.

Returns

equation : SymPy expression

Examples

```
>>> from sympy import Point, Line
>>> p1, p2 = Point(1, 0), Point(5, 3)
>>> l1 = Line(p1, p2)
>>> l1.equation()
-3*x + 4*y + 3
```

See also:

sympy.geometry.line.Line2D.coefficients (page 2239)

class sympy.geometry.line.**Ray2D**(*p1*, *pt=None*, *angle=None*, ***kwargs*)

A Ray is a semi-line in the space with a source point and a direction.

Parameters

p1 : Point

The source of the Ray

p2 : Point or radian value

This point determines the direction in which the Ray propagates. If given as an angle it is interpreted in radians with the positive direction being ccw.

Examples

```
>>> from sympy import Point, pi, Ray
>>> r = Ray(Point(2, 3), Point(3, 5))
>>> r
Ray2D(Point2D(2, 3), Point2D(3, 5))
>>> r.points
(Point2D(2, 3), Point2D(3, 5))
>>> r.source
Point2D(2, 3)
>>> r.xdirection
oo
>>> r.ydirection
oo
>>> r.slope
2
>>> Ray(Point(0, 0), angle=pi/4).slope
1
```

See also:

sympy.geometry.point.Point (page 2202), *Line* (page 2227)

Attributes

source	
xdirection	
ydirection	

closing_angle(*r2*)

Return the angle by which r2 must be rotated so it faces the same direction as r1.

Parameters

r1 : Ray2D

r2 : Ray2D

Returns

angle : angle in radians (ccw angle is positive)

Examples

```
>>> from sympy import Ray, pi
>>> r1 = Ray((0, 0), (1, 0))
>>> r2 = r1.rotate(-pi/2)
>>> angle = r1.closing_angle(r2); angle
pi/2
>>> r2.rotate(angle).direction.unit == r1.direction.unit
True
>>> r2.closing_angle(r1)
-pi/2
```

See also:

LinearEntity.angle_between (page 2216)

property xdirection

The x direction of the ray.

Positive infinity if the ray points in the positive x direction, negative infinity if the ray points in the negative x direction, or 0 if the ray is vertical.

Examples

```
>>> from sympy import Point, Ray
>>> p1, p2, p3 = Point(0, 0), Point(1, 1), Point(0, -1)
>>> r1, r2 = Ray(p1, p2), Ray(p1, p3)
>>> r1.xdirection
oo
>>> r2.xdirection
0
```

See also:

ydirection (page 2241)

property ydirection

The y direction of the ray.

Positive infinity if the ray points in the positive y direction, negative infinity if the ray points in the negative y direction, or 0 if the ray is horizontal.

Examples

```
>>> from sympy import Point, Ray
>>> p1, p2, p3 = Point(0, 0), Point(-1, -1), Point(-1, 0)
>>> r1, r2 = Ray(p1, p2), Ray(p1, p3)
>>> r1.ydirection
-oo
>>> r2.ydirection
0
```

See also:

xdirection (page 2241)

class sympy.geometry.line.**Segment2D**(*p1*, *p2*, ***kwargs*)

A line segment in 2D space.

Parameters

p1 : Point

p2 : Point

Examples

```
>>> from sympy import Point, Segment
>>> Segment((1, 0), (1, 1)) # tuples are interpreted as pts
Segment2D(Point2D(1, 0), Point2D(1, 1))
>>> s = Segment(Point(4, 3), Point(1, 1)); s
Segment2D(Point2D(4, 3), Point2D(1, 1))
>>> s.points
(Point2D(4, 3), Point2D(1, 1))
>>> s.slope
2/3
>>> s.length
sqrt(13)
>>> s.midpoint
Point2D(5/2, 2)
```

See also:

sympy.geometry.point.Point (page 2202), *Line* (page 2227)

Attributes

length	(number or SymPy expression)
midpoint	(Point)

class sympy.geometry.line.**LinearEntity3D**(*p1*, *p2*, ***kwargs*)

An base class for all linear entities (line, ray and segment) in a 3-dimensional Euclidean space.

Notes

This is a base class and is not meant to be instantiated.

Attributes

p1	
p2	
direction_ratio	
direction_cosine	
points	

property direction_cosine

The normalized direction ratio of a given line in 3D.

Examples

```
>>> from sympy import Point3D, Line3D
>>> p1, p2 = Point3D(0, 0, 0), Point3D(5, 3, 1)
>>> l = Line3D(p1, p2)
>>> l.direction_cosine
[sqrt(35)/7, 3*sqrt(35)/35, sqrt(35)/35]
>>> sum(i**2 for i in _)
1
```

See also:

sympy.geometry.line.Line3D.equation (page 2244)

property direction_ratio

The direction ratio of a given line in 3D.

Examples

```
>>> from sympy import Point3D, Line3D
>>> p1, p2 = Point3D(0, 0, 0), Point3D(5, 3, 1)
>>> l = Line3D(p1, p2)
>>> l.direction_ratio
[5, 3, 1]
```

See also:

sympy.geometry.line.Line3D.equation (page 2244)

class sympy.geometry.line.**Line3D**(*p1*, *pt=None*, *direction_ratio=()*, ***kwargs*)

An infinite 3D line in space.

A line is declared with two distinct points or a point and direction_ratio as defined using keyword $direction_ratio$.

Parameters

p1 : Point3D

pt : Point3D

direction_ratio : list

Examples

```
>>> from sympy import Line3D, Point3D
>>> L = Line3D(Point3D(2, 3, 4), Point3D(3, 5, 1))
>>> L
Line3D(Point3D(2, 3, 4), Point3D(3, 5, 1))
>>> L.points
(Point3D(2, 3, 4), Point3D(3, 5, 1))
```

See also:

sympy.geometry.point.Point3D (page 2211), *sympy.geometry.line.Line* (page 2227), *sympy.geometry.line.Line2D* (page 2238)

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

Return the equations that define the line in 3D.

Parameters

x : str, optional

The name to use for the x-axis, default value is 'x'.

y : str, optional

The name to use for the y-axis, default value is 'y'.

z : str, optional

The name to use for the z-axis, default value is 'z'.

k : str, optional

Deprecated since version 1.2: The k flag is deprecated. It does nothing.

Returns

equation : Tuple of simultaneous equations

Examples

```
>>> from sympy import Point3D, Line3D, solve
>>> from sympy.abc import x, y, z
>>> p1, p2 = Point3D(1, 0, 0), Point3D(5, 3, 0)
>>> l1 = Line3D(p1, p2)
>>> eq = l1.equation(x, y, z); eq
(-3*x + 4*y + 3, z)
>>> solve(eq.subs(z, 0), (x, y, z))
{x: 4*y/3 + 1}
```

class sympy.geometry.line.**Ray3D**(*p1*, *pt=None*, *direction_ratio=()*, ***kwargs*)

A Ray is a semi-line in the space with a source point and a direction.

Parameters

p1 : Point3D

The source of the Ray

p2 : Point or a direction vector

direction_ratio: Determines the direction in which the Ray propagates.

Examples

```
>>> from sympy import Point3D, Ray3D
>>> r = Ray3D(Point3D(2, 3, 4), Point3D(3, 5, 0))
>>> r
Ray3D(Point3D(2, 3, 4), Point3D(3, 5, 0))
>>> r.points
(Point3D(2, 3, 4), Point3D(3, 5, 0))
>>> r.source
Point3D(2, 3, 4)
>>> r.xdirection
oo
>>> r.ydirection
oo
>>> r.direction_ratio
[1, 2, -4]
```

See also:

sympy.geometry.point.Point3D (page 2211), *Line3D* (page 2243)

Attributes

source	
xdirection	
ydirection	
zdirection	

property xdirection

The x direction of the ray.

Positive infinity if the ray points in the positive x direction, negative infinity if the ray points in the negative x direction, or 0 if the ray is vertical.

Examples

```
>>> from sympy import Point3D, Ray3D
>>> p1, p2, p3 = Point3D(0, 0, 0), Point3D(1, 1, 1), Point3D(0, -1, 0)
>>> r1, r2 = Ray3D(p1, p2), Ray3D(p1, p3)
>>> r1.xdirection
oo
>>> r2.xdirection
0
```

See also:

ydirection (page 2246)

property ydirection

The y direction of the ray.

Positive infinity if the ray points in the positive y direction, negative infinity if the ray points in the negative y direction, or 0 if the ray is horizontal.

Examples

```
>>> from sympy import Point3D, Ray3D
>>> p1, p2, p3 = Point3D(0, 0, 0), Point3D(-1, -1, -1), Point3D(-1, 0,
 ↪ 0)
>>> r1, r2 = Ray3D(p1, p2), Ray3D(p1, p3)
>>> r1.ydirection
-oo
>>> r2.ydirection
0
```

See also:

xdirection (page 2245)

property zdirection

The z direction of the ray.

Positive infinity if the ray points in the positive z direction, negative infinity if the ray points in the negative z direction, or 0 if the ray is horizontal.

Examples

```
>>> from sympy import Point3D, Ray3D
>>> p1, p2, p3 = Point3D(0, 0, 0), Point3D(-1, -1, -1), Point3D(-1, 0,
 ↪ 0)
>>> r1, r2 = Ray3D(p1, p2), Ray3D(p1, p3)
>>> r1.ydirection
-oo
>>> r2.ydirection
0
>>> r2.zdirection
0
```

See also:

xdirection (page 2245)

class sympy.geometry.line.**Segment3D**(*p1*, *p2*, ***kwargs*)

A line segment in a 3D space.

Parameters

p1 : Point3D

p2 : Point3D

Examples

```
>>> from sympy import Point3D, Segment3D
>>> Segment3D((1, 0, 0), (1, 1, 1)) # tuples are interpreted as pts
Segment3D(Point3D(1, 0, 0), Point3D(1, 1, 1))
>>> s = Segment3D(Point3D(4, 3, 9), Point3D(1, 1, 7)); s
Segment3D(Point3D(4, 3, 9), Point3D(1, 1, 7))
>>> s.points
(Point3D(4, 3, 9), Point3D(1, 1, 7))
>>> s.length
sqrt(17)
>>> s.midpoint
Point3D(5/2, 2, 8)
```

See also:

sympy.geometry.point.Point3D (page 2211), *Line3D* (page 2243)

Attributes

length	(number or SymPy expression)
midpoint	(Point3D)

Curves

class sympy.geometry.curve.**Curve**(*function*, *limits*)

A curve in space.

A curve is defined by parametric functions for the coordinates, a parameter and the lower and upper bounds for the parameter value.

Parameters

function : list of functions

limits : 3-tuple

Function parameter and lower and upper bounds.

Raises

ValueError

When $functions$ are specified incorrectly. When $limits$ are specified incorrectly.

Examples

```
>>> from sympy import Curve, sin, cos, interpolate
>>> from sympy.abc import t, a
>>> C = Curve((sin(t), cos(t)), (t, 0, 2))
>>> C.functions
(sin(t), cos(t))
>>> C.limits
(t, 0, 2)
>>> C.parameter
t
>>> C = Curve((t, interpolate([1, 4, 9, 16], t)), (t, 0, 1)); C
Curve((t, t**2), (t, 0, 1))
>>> C.subs(t, 4)
Point2D(4, 16)
>>> C.arbitrary_point(a)
Point2D(a, a**2)
```

See also:

sympy.core.function.Function (page 1050), *sympy.polys.polyfuncs.interpolate* (page 2426)

Attributes

functions	
parameter	
limits	

property ambient_dimension

The dimension of the curve.

Returns

int :

the dimension of curve.

Examples

```
>>> from sympy.abc import t
>>> from sympy import Curve
>>> C = Curve((t, t**2), (t, 0, 2))
>>> C.ambient_dimension
2
```

arbitrary_point(*parameter='t'*)

A parameterized point on the curve.

Parameters

parameter : str or Symbol, optional

Default value is 't'. The Curve's parameter is selected with None or self.parameter otherwise the provided symbol is used.

Returns

Point :

Returns a point in parametric form.

Raises

ValueError

When *parameter* already appears in the functions.

Examples

```
>>> from sympy import Curve, Symbol
>>> from sympy.abc import s
>>> C = Curve([2*s, s**2], (s, 0, 2))
>>> C.arbitrary_point()
Point2D(2*t, t**2)
>>> C.arbitrary_point(C.parameter)
Point2D(2*s, s**2)
>>> C.arbitrary_point(None)
Point2D(2*s, s**2)
>>> C.arbitrary_point(Symbol('a'))
Point2D(2*a, a**2)
```

See also:

sympy.geometry.point.Point (page 2202)

property free_symbols

Return a set of symbols other than the bound symbols used to parametrically define the Curve.

Returns

set :

Set of all non-parameterized symbols.

Examples

```
>>> from sympy.abc import t, a
>>> from sympy import Curve
>>> Curve((t, t**2), (t, 0, 2)).free_symbols
set()
>>> Curve((t, t**2), (t, a, 2)).free_symbols
{a}
```

property functions

The functions specifying the curve.

Returns

functions :

list of parameterized coordinate functions.

Examples

```
>>> from sympy.abc import t
>>> from sympy import Curve
>>> C = Curve((t, t**2), (t, 0, 2))
>>> C.functions
(t, t**2)
```

See also:

parameter (page 2250)

property length

The curve length.

Examples

```
>>> from sympy import Curve
>>> from sympy.abc import t
>>> Curve((t, t), (t, 0, 1)).length
sqrt(2)
```

property limits

The limits for the curve.

Returns

limits : tuple

Contains parameter and lower and upper limits.

Examples

```
>>> from sympy.abc import t
>>> from sympy import Curve
>>> C = Curve([t, t**3], (t, -2, 2))
>>> C.limits
(t, -2, 2)
```

See also:

plot_interval (page 2251)

property parameter

The curve function variable.

Returns

Symbol :

returns a bound symbol.

Examples

```
>>> from sympy.abc import t
>>> from sympy import Curve
>>> C = Curve([t, t**2], (t, 0, 2))
>>> C.parameter
t
```

See also:

functions (page 2249)

plot_interval(*parameter='t'*)

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

Parameters

parameter : str or Symbol, optional

Default value is 't'; otherwise the provided symbol is used.

Returns

List :

the plot interval as below:

[parameter, lower_bound, upper_bound]

Examples

```
>>> from sympy import Curve, sin
>>> from sympy.abc import x, s
>>> Curve((x, sin(x)), (x, 1, 2)).plot_interval()
[t, 1, 2]
>>> Curve((x, sin(x)), (x, 1, 2)).plot_interval(s)
[s, 1, 2]
```

See also:

***limits* (page 2250)**

Returns limits of the parameter interval

rotate(*angle=0, pt=None*)

This function is used to rotate a curve along given point `pt` at given angle(in radian).

Parameters

angle :

the angle at which the curve will be rotated(in radian) in counter-clockwise direction. default value of angle is 0.

pt : Point

the point along which the curve will be rotated. If no point given, the curve will be rotated around origin.

Returns

Curve :

returns a curve rotated at given angle along given point.

Examples

```
>>> from sympy import Curve, pi
>>> from sympy.abc import x
>>> Curve((x, x), (x, 0, 1)).rotate(pi/2)
Curve((-x, x), (x, 0, 1))
```

scale(*x=1*, *y=1*, *pt=None*)

Override GeometryEntity.scale since Curve is not made up of Points.

Returns

Curve :

returns scaled curve.

Examples

```
>>> from sympy import Curve
>>> from sympy.abc import x
>>> Curve((x, x), (x, 0, 1)).scale(2)
Curve((2*x, x), (x, 0, 1))
```

translate(*x=0*, *y=0*)

Translate the Curve by (x, y).

Returns

Curve :

returns a translated curve.

Examples

```
>>> from sympy import Curve
>>> from sympy.abc import x
>>> Curve((x, x), (x, 0, 1)).translate(1, 2)
Curve((x + 1, x + 2), (x, 0, 1))
```

Ellipses

class sympy.geometry.ellipse.**Ellipse**(*center=None*, *hradius=None*, *vradius=None*, *eccentricity=None*, ***kwargs*)

An elliptical GeometryEntity.

Parameters

center : Point, optional

Default value is Point(0, 0)

hradius : number or SymPy expression, optional

vradius : number or SymPy expression, optional

eccentricity : number or SymPy expression, optional

Two of *hradius*, *vradius* and *eccentricity* must be supplied to create an Ellipse. The third is derived from the two supplied.

Raises

GeometryError

When *hradius*, *vradius* and *eccentricity* are incorrectly supplied as parameters.

TypeError

When *center* is not a Point.

Notes

Constructed from a center and two radii, the first being the horizontal radius (along the x-axis) and the second being the vertical radius (along the y-axis).

When symbolic value for hradius and vradius are used, any calculation that refers to the foci or the major or minor axis will assume that the ellipse has its major radius on the x-axis. If this is not true then a manual rotation is necessary.

Examples

```
>>> from sympy import Ellipse, Point, Rational
>>> e1 = Ellipse(Point(0, 0), 5, 1)
>>> e1.hradius, e1.vradius
(5, 1)
>>> e2 = Ellipse(Point(3, 1), hradius=3, eccentricity=Rational(4, 5))
>>> e2
Ellipse(Point2D(3, 1), 3, 9/5)
```

See also:

Circle (page 2269)

Attributes

center	
hradius	
vradius	
area	
circumference	
eccentricity	
periapsis	
apoapsis	
focus_distance	
foci	

property apoapsis

The apoapsis of the ellipse.

The greatest distance between the focus and the contour.

Returns

apoapsis : number

Examples

```
>>> from sympy import Point, Ellipse
>>> p1 = Point(0, 0)
>>> e1 = Ellipse(p1, 3, 1)
>>> e1.apoapsis
2*sqrt(2) + 3
```

See also:

periapsis **(page 2263)**

Returns shortest distance between foci and contour

arbitrary_point(*parameter*='*t*')

A parameterized point on the ellipse.

Parameters

parameter : str, optional

Default value is 't'.

Returns

arbitrary_point : Point

Raises

ValueError

When *parameter* already appears in the functions.

Examples

```
>>> from sympy import Point, Ellipse
>>> e1 = Ellipse(Point(0, 0), 3, 2)
>>> e1.arbitrary_point()
Point2D(3*cos(t), 2*sin(t))
```

See also:

sympy.geometry.point.Point (page 2202)

property area

The area of the ellipse.

Returns

area : number

Examples

```
>>> from sympy import Point, Ellipse
>>> p1 = Point(0, 0)
>>> e1 = Ellipse(p1, 3, 1)
>>> e1.area
3*pi
```

auxiliary_circle()

Returns a Circle whose diameter is the major axis of the ellipse.

Examples

```
>>> from sympy import Ellipse, Point, symbols
>>> c = Point(1, 2)
>>> Ellipse(c, 8, 7).auxiliary_circle()
Circle(Point2D(1, 2), 8)
>>> a, b = symbols('a b')
>>> Ellipse(c, a, b).auxiliary_circle()
Circle(Point2D(1, 2), Max(a, b))
```

property bounds

Return a tuple (xmin, ymin, xmax, ymax) representing the bounding rectangle for the geometric figure.

property center

The center of the ellipse.

Returns

center : number

Examples

```
>>> from sympy import Point, Ellipse
>>> p1 = Point(0, 0)
>>> e1 = Ellipse(p1, 3, 1)
>>> e1.center
Point2D(0, 0)
```

See also:

sympy.geometry.point.Point (page 2202)

property circumference

The circumference of the ellipse.

Examples

```
>>> from sympy import Point, Ellipse
>>> p1 = Point(0, 0)
>>> e1 = Ellipse(p1, 3, 1)
>>> e1.circumference
12*elliptic_e(8/9)
```

director_circle()

Returns a Circle consisting of all points where two perpendicular tangent lines to the ellipse cross each other.

Returns

Circle

A director circle returned as a geometric object.

Examples

```
>>> from sympy import Ellipse, Point, symbols
>>> c = Point(3,8)
>>> Ellipse(c, 7, 9).director_circle()
Circle(Point2D(3, 8), sqrt(130))
>>> a, b = symbols('a b')
>>> Ellipse(c, a, b).director_circle()
Circle(Point2D(3, 8), sqrt(a**2 + b**2))
```

References

[R503]

property eccentricity

The eccentricity of the ellipse.

Returns
eccentricity : number

Examples

```
>>> from sympy import Point, Ellipse, sqrt
>>> p1 = Point(0, 0)
>>> e1 = Ellipse(p1, 3, sqrt(2))
>>> e1.eccentricity
sqrt(7)/3
```

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 Ellipse, S
>>> from sympy.abc import t
>>> e = Ellipse((0, 0), 3, 2)
>>> e.encloses_point((0, 0))
True
>>> e.encloses_point(e.arbitrary_point(t).subs(t, S.Half))
False
>>> e.encloses_point((4, 0))
False
```

See also:

sympy.geometry.point.Point (page 2202)

equation(*x='x'*, *y='y'*, *_slope=None*)

Returns the equation of an ellipse aligned with the x and y axes; when slope is given, the equation returned corresponds to an ellipse with a major axis having that slope.

Parameters
x : str, optional

Label for the x-axis. Default value is 'x'.

y : str, optional

Label for the y-axis. Default value is 'y'.

_slope : Expr, optional

The slope of the major axis. Ignored when 'None'.

Returns

equation : SymPy expression

Examples

```
>>> from sympy import Point, Ellipse, pi
>>> from sympy.abc import x, y
>>> e1 = Ellipse(Point(1, 0), 3, 2)
>>> eq1 = e1.equation(x, y); eq1
y**2/4 + (x/3 - 1/3)**2 - 1
>>> eq2 = e1.equation(x, y, _slope=1); eq2
(-x + y + 1)**2/8 + (x + y - 1)**2/18 - 1
```

A point on e1 satisfies eq1. Let's use one on the x-axis:

```
>>> p1 = e1.center + Point(e1.major, 0)
>>> assert eq1.subs(x, p1.x).subs(y, p1.y) == 0
```

When rotated the same as the rotated ellipse, about the center point of the ellipse, it will satisfy the rotated ellipse's equation, too:

```
>>> r1 = p1.rotate(pi/4, e1.center)
>>> assert eq2.subs(x, r1.x).subs(y, r1.y) == 0
```

See also:

arbitrary_point **(page 2254)**

Returns parameterized point on ellipse

References

[R504], [R505]

evolute(*x*='*x*', *y*='*y*')

The equation of evolute of the ellipse.

Parameters

x : str, optional

Label for the x-axis. Default value is 'x'.

y : str, optional

Label for the y-axis. Default value is 'y'.

Returns

equation : SymPy expression

Examples

```
>>> from sympy import Point, Ellipse
>>> e1 = Ellipse(Point(1, 0), 3, 2)
>>> e1.evolute()
2**(2/3)*y**(2/3) + (3*x - 3)**(2/3) - 5**(2/3)
```

property foci

The foci of the ellipse.

Raises
ValueError

When the major and minor axis cannot be determined.

Notes

The foci can only be calculated if the major/minor axes are known.

Examples

```
>>> from sympy import Point, Ellipse
>>> p1 = Point(0, 0)
>>> e1 = Ellipse(p1, 3, 1)
>>> e1.foci
(Point2D(-2*sqrt(2), 0), Point2D(2*sqrt(2), 0))
```

See also:

sympy.geometry.point.Point (page 2202)

***focus_distance* (page 2259)**
Returns the distance between focus and center

property focus_distance

The focal distance of the ellipse.

The distance between the center and one focus.

Returns
focus_distance : number

Examples

```
>>> from sympy import Point, Ellipse
>>> p1 = Point(0, 0)
>>> e1 = Ellipse(p1, 3, 1)
>>> e1.focus_distance
2*sqrt(2)
```

See also:

foci (page 2259)

property hradius

The horizontal radius of the ellipse.

Returns

hradius : number

Examples

```
>>> from sympy import Point, Ellipse
>>> p1 = Point(0, 0)
>>> e1 = Ellipse(p1, 3, 1)
>>> e1.hradius
3
```

See also:

vradius (page 2269), *major* (page 2261), *minor* (page 2262)

intersection(*o*)

The intersection of this ellipse and another geometrical entity *o*.

Parameters

o : GeometryEntity

Returns

intersection : list of GeometryEntity objects

Notes

Currently supports intersections with Point, Line, Segment, Ray, Circle and Ellipse types.

Examples

```
>>> from sympy import Ellipse, Point, Line
>>> e = Ellipse(Point(0, 0), 5, 7)
>>> e.intersection(Point(0, 0))
[]
>>> e.intersection(Point(5, 0))
[Point2D(5, 0)]
>>> e.intersection(Line(Point(0,0), Point(0, 1)))
[Point2D(0, -7), Point2D(0, 7)]
>>> e.intersection(Line(Point(5,0), Point(5, 1)))
[Point2D(5, 0)]
>>> e.intersection(Line(Point(6,0), Point(6, 1)))
[]
>>> e = Ellipse(Point(-1, 0), 4, 3)
>>> e.intersection(Ellipse(Point(1, 0), 4, 3))
[Point2D(0, -3*sqrt(15)/4), Point2D(0, 3*sqrt(15)/4)]
>>> e.intersection(Ellipse(Point(5, 0), 4, 3))
[Point2D(2, -3*sqrt(7)/4), Point2D(2, 3*sqrt(7)/4)]
```

(continues on next page)

(continued from previous page)

```
>>> e.intersection(Ellipse(Point(100500, 0), 4, 3))
[]
>>> e.intersection(Ellipse(Point(0, 0), 3, 4))
[Point2D(3, 0), Point2D(-363/175, -48*sqrt(111)/175), Point2D(-363/
↪175, 48*sqrt(111)/175)]
>>> e.intersection(Ellipse(Point(-1, 0), 3, 4))
[Point2D(-17/5, -12/5), Point2D(-17/5, 12/5), Point2D(7/5, -12/5),␣
↪Point2D(7/5, 12/5)]
```

See also:

sympy.geometry.entity.GeometryEntity (page 2194)

is_tangent(*o*)

Is *o* tangent to the ellipse?

Parameters

o : GeometryEntity

An Ellipse, LinearEntity or Polygon

Returns

is_tangent: boolean

True if o is tangent to the ellipse, False otherwise.

Raises

NotImplementedError

When the wrong type of argument is supplied.

Examples

```
>>> from sympy import Point, Ellipse, Line
>>> p0, p1, p2 = Point(0, 0), Point(3, 0), Point(3, 3)
>>> e1 = Ellipse(p0, 3, 2)
>>> l1 = Line(p1, p2)
>>> e1.is_tangent(l1)
True
```

See also:

tangent_lines (page 2268)

property major

Longer axis of the ellipse (if it can be determined) else hradius.

Returns

major : number or expression

Examples

```
>>> from sympy import Point, Ellipse, Symbol
>>> p1 = Point(0, 0)
>>> e1 = Ellipse(p1, 3, 1)
>>> e1.major
3
```

```
>>> a = Symbol('a')
>>> b = Symbol('b')
>>> Ellipse(p1, a, b).major
a
>>> Ellipse(p1, b, a).major
b
```

```
>>> m = Symbol('m')
>>> M = m + 1
>>> Ellipse(p1, m, M).major
m + 1
```

See also:

hradius (page 2259), *vradius* (page 2269), *minor* (page 2262)

property minor

Shorter axis of the ellipse (if it can be determined) else vradius.

Returns

minor : number or expression

Examples

```
>>> from sympy import Point, Ellipse, Symbol
>>> p1 = Point(0, 0)
>>> e1 = Ellipse(p1, 3, 1)
>>> e1.minor
1
```

```
>>> a = Symbol('a')
>>> b = Symbol('b')
>>> Ellipse(p1, a, b).minor
b
>>> Ellipse(p1, b, a).minor
a
```

```
>>> m = Symbol('m')
>>> M = m + 1
>>> Ellipse(p1, m, M).minor
m
```

See also:

hradius (page 2259), *vradius* (page 2269), *major* (page 2261)

normal_lines(*p*, *prec=None*)

Normal lines between p and the ellipse.

Parameters
p : Point

Returns
normal_lines : list with 1, 2 or 4 Lines

Examples

```
>>> from sympy import Point, Ellipse
>>> e = Ellipse((0, 0), 2, 3)
>>> c = e.center
>>> e.normal_lines(c + Point(1, 0))
[Line2D(Point2D(0, 0), Point2D(1, 0))]
>>> e.normal_lines(c)
[Line2D(Point2D(0, 0), Point2D(0, 1)), Line2D(Point2D(0, 0),␣
↪Point2D(1, 0))]
```

Off-axis points require the solution of a quartic equation. This often leads to very large expressions that may be of little practical use. An approximate solution of $prec$ digits can be obtained by passing in the desired value:

```
>>> e.normal_lines((3, 3), prec=2)
[Line2D(Point2D(-0.81, -2.7), Point2D(0.19, -1.2)),
Line2D(Point2D(1.5, -2.0), Point2D(2.5, -2.7))]
```

Whereas the above solution has an operation count of 12, the exact solution has an operation count of 2020.

property periapsis

The periapsis of the ellipse.

The shortest distance between the focus and the contour.

Returns
periapsis : number

Examples

```
>>> from sympy import Point, Ellipse
>>> p1 = Point(0, 0)
>>> e1 = Ellipse(p1, 3, 1)
>>> e1.periapsis
3 - 2*sqrt(2)
```

See also:

apoapsis **(page 2254)**
Returns greatest distance between focus and contour

plot_interval(*parameter='t'*)

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

Parameters

parameter : str, optional

Default value is 't'.

Returns

plot_interval : list

[parameter, lower_bound, upper_bound]

Examples

```
>>> from sympy import Point, Ellipse
>>> e1 = Ellipse(Point(0, 0), 3, 2)
>>> e1.plot_interval()
[t, -pi, pi]
```

polar_second_moment_of_area()

Returns the polar second moment of area of an Ellipse

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 symbols, Circle, Ellipse
>>> c = Circle((5, 5), 4)
>>> c.polar_second_moment_of_area()
128*pi
>>> a, b = symbols('a, b')
>>> e = Ellipse((0, 0), a, b)
>>> e.polar_second_moment_of_area()
pi*a**3*b/4 + pi*a*b**3/4
```

References

[R506]

random_point(*seed=None*)

A random point on the ellipse.

Returns

point : Point

Examples

```
>>> from sympy import Point, Ellipse
>>> e1 = Ellipse(Point(0, 0), 3, 2)
>>> e1.random_point() # gives some random point
Point2D(...)
>>> p1 = e1.random_point(seed=0); p1.n(2)
Point2D(2.1, 1.4)
```

Notes

When creating a random point, one may simply replace the parameter with a random number. When doing so, however, the random number should be made a Rational or else the point may not test as being in the ellipse:

```
>>> from sympy.abc import t
>>> from sympy import Rational
>>> arb = e1.arbitrary_point(t); arb
Point2D(3*cos(t), 2*sin(t))
>>> arb.subs(t, .1) in e1
False
>>> arb.subs(t, Rational(.1)) in e1
True
>>> arb.subs(t, Rational('.1')) in e1
True
```

See also:

sympy.geometry.point.Point (page 2202)

***arbitrary_point* (page 2254)**
Returns parameterized point on ellipse

reflect(*line*)

Override GeometryEntity.reflect since the radius is not a GeometryEntity.

Examples

```
>>> from sympy import Circle, Line
>>> Circle((0, 1), 1).reflect(Line((0, 0), (1, 1)))
Circle(Point2D(1, 0), -1)
>>> from sympy import Ellipse, Line, Point
>>> Ellipse(Point(3, 4), 1, 3).reflect(Line(Point(0, -4), Point(5,␣
↪0)))
Traceback (most recent call last):
...
NotImplementedError:
General Ellipse is not supported but the equation of the reflected
Ellipse is given by the zeros of: f(x, y) = (9*x/41 + 40*y/41 +
37/41)**2 + (40*x/123 - 3*y/41 - 364/123)**2 - 1
```

Notes

Until the general ellipse (with no axis parallel to the x-axis) is supported a NotImplemented error is raised and the equation whose zeros define the rotated ellipse is given.

rotate(*angle=0*, *pt=None*)

Rotate `angle` radians counterclockwise about Point `pt`.

Note: since the general ellipse is not supported, only rotations that are integer multiples of pi/2 are allowed.

Examples

```
>>> from sympy import Ellipse, pi
>>> Ellipse((1, 0), 2, 1).rotate(pi/2)
Ellipse(Point2D(0, 1), 1, 2)
>>> Ellipse((1, 0), 2, 1).rotate(pi)
Ellipse(Point2D(-1, 0), 2, 1)
```

scale(*x=1*, *y=1*, *pt=None*)

Override GeometryEntity.scale since it is the major and minor axes which must be scaled and they are not GeometryEntities.

Examples

```
>>> from sympy import Ellipse
>>> Ellipse((0, 0), 2, 1).scale(2, 4)
Circle(Point2D(0, 0), 4)
>>> Ellipse((0, 0), 2, 1).scale(2)
Ellipse(Point2D(0, 0), 4, 1)
```

second_moment_of_area(*point=None*)

Returns the second moment and product moment area of an ellipse.

Parameters

point : Point, two-tuple of sympifiable 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 ellipse.

Returns

I_xx, I_yy, I_xy : number or SymPy expression

I_xx, I_yy are second moment of area of an ellise. I_xy is product moment of area of an ellipse.

Examples

```
>>> from sympy import Point, Ellipse
>>> p1 = Point(0, 0)
>>> e1 = Ellipse(p1, 3, 1)
>>> e1.second_moment_of_area()
(3*pi/4, 27*pi/4, 0)
```

References

[R507]

section_modulus(*point=None*)

Returns a tuple with the section modulus of an ellipse

Section modulus is a geometric property of an ellipse defined as the ratio of second moment of area to the distance of the extreme end of the ellipse 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" section modulus will be calculated for the point farthest from the centroidal axis of the ellipse.

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 Symbol, Ellipse, Circle, Point2D
>>> d = Symbol('d', positive=True)
>>> c = Circle((0, 0), d/2)
>>> c.section_modulus()
(pi*d**3/32, pi*d**3/32)
>>> e = Ellipse(Point2D(0, 0), 2, 4)
>>> e.section_modulus()
(8*pi, 4*pi)
>>> e.section_modulus((2, 2))
(16*pi, 4*pi)
```

References

[R508]

property semilatus_rectum

Calculates the semi-latus rectum of the Ellipse.

Semi-latus rectum is defined as one half of the chord through a focus parallel to the conic section directrix of a conic section.

Returns
semilatus_rectum : number

Examples

```
>>> from sympy import Point, Ellipse
>>> p1 = Point(0, 0)
>>> e1 = Ellipse(p1, 3, 1)
>>> e1.semilatus_rectum
1/3
```

See also:

apoapsis **(page 2254)**
Returns greatest distance between focus and contour

periapsis **(page 2263)**
The shortest distance between the focus and the contour

References

[R509], [R510]

tangent_lines(*p*)

Tangent lines between p and the ellipse.

If p is on the ellipse, returns the tangent line through point p. Otherwise, returns the tangent line(s) from p to the ellipse, or None if no tangent line is possible (e.g., p inside ellipse).

Parameters
p : Point

Returns
tangent_lines : list with 1 or 2 Lines

Raises
NotImplementedError

Can only find tangent lines for a point, p, on the ellipse.

Examples

```
>>> from sympy import Point, Ellipse
>>> e1 = Ellipse(Point(0, 0), 3, 2)
>>> e1.tangent_lines(Point(3, 0))
[Line2D(Point2D(3, 0), Point2D(3, -12))]
```

See also:

sympy.geometry.point.Point (page 2202), *sympy.geometry.line.Line* (page 2227)

property vradius

The vertical radius of the ellipse.

Returns

vradius : number

Examples

```
>>> from sympy import Point, Ellipse
>>> p1 = Point(0, 0)
>>> e1 = Ellipse(p1, 3, 1)
>>> e1.vradius
1
```

See also:

hradius (page 2259), *major* (page 2261), *minor* (page 2262)

class sympy.geometry.ellipse.**Circle**(**args*, ***kwargs*)

A circle in space.

Constructed simply from a center and a radius, from three non-collinear points, or the equation of a circle.

Parameters

center : Point

radius : number or SymPy expression

points : sequence of three Points

equation : equation of a circle

Raises

GeometryError

When the given equation is not that of a circle. When trying to construct circle from incorrect parameters.

Examples

```
>>> from sympy import Point, Circle, Eq
>>> from sympy.abc import x, y, a, b
```

A circle constructed from a center and radius:

```
>>> c1 = Circle(Point(0, 0), 5)
>>> c1.hradius, c1.vradius, c1.radius
(5, 5, 5)
```

A circle constructed from three points:

```
>>> c2 = Circle(Point(0, 0), Point(1, 1), Point(1, 0))
>>> c2.hradius, c2.vradius, c2.radius, c2.center
(sqrt(2)/2, sqrt(2)/2, sqrt(2)/2, Point2D(1/2, 1/2))
```

A circle can be constructed from an equation in the form $a**x**2 + by**2 + gx + hy + c = 0$, too:

```
>>> Circle(x**2 + y**2 - 25)
Circle(Point2D(0, 0), 5)
```

If the variables corresponding to x and y are named something else, their name or symbol can be supplied:

```
>>> Circle(Eq(a**2 + b**2, 25), x='a', y=b)
Circle(Point2D(0, 0), 5)
```

See also:

Ellipse (page 2253), *sympy.geometry.point.Point* (page 2202)

Attributes

radius (synonymous with hradius, vradius, major and minor)	
circumference	
equation	

property circumference

The circumference of the circle.

Returns

circumference : number or SymPy expression

Examples

```
>>> from sympy import Point, Circle
>>> c1 = Circle(Point(3, 4), 6)
>>> c1.circumference
12*pi
```

equation(*x*='x', *y*='y')

The equation of the circle.

Parameters

x : str or Symbol, optional

Default value is 'x'.

y : str or Symbol, optional

Default value is 'y'.

Returns

equation : SymPy expression

Examples

```
>>> from sympy import Point, Circle
>>> c1 = Circle(Point(0, 0), 5)
>>> c1.equation()
x**2 + y**2 - 25
```

intersection(*o*)

The intersection of this circle with another geometrical entity.

Parameters

o : GeometryEntity

Returns

intersection : list of GeometryEntities

Examples

```
>>> from sympy import Point, Circle, Line, Ray
>>> p1, p2, p3 = Point(0, 0), Point(5, 5), Point(6, 0)
>>> p4 = Point(5, 0)
>>> c1 = Circle(p1, 5)
>>> c1.intersection(p2)
[]
>>> c1.intersection(p4)
[Point2D(5, 0)]
>>> c1.intersection(Ray(p1, p2))
[Point2D(5*sqrt(2)/2, 5*sqrt(2)/2)]
>>> c1.intersection(Line(p2, p3))
[]
```

property radius

The radius of the circle.

Returns

radius : number or SymPy expression

Examples

```
>>> from sympy import Point, Circle
>>> c1 = Circle(Point(3, 4), 6)
>>> c1.radius
6
```

See also:

Ellipse.major (page 2261), *Ellipse.minor* (page 2262), *Ellipse.hradius* (page 2259), *Ellipse.vradius* (page 2269)

reflect(*line*)

Override GeometryEntity.reflect since the radius is not a GeometryEntity.

Examples

```
>>> from sympy import Circle, Line
>>> Circle((0, 1), 1).reflect(Line((0, 0), (1, 1)))
Circle(Point2D(1, 0), -1)
```

scale(*x=1*, *y=1*, *pt=None*)

Override GeometryEntity.scale since the radius is not a GeometryEntity.

Examples

```
>>> from sympy import Circle
>>> Circle((0, 0), 1).scale(2, 2)
Circle(Point2D(0, 0), 2)
>>> Circle((0, 0), 1).scale(2, 4)
Ellipse(Point2D(0, 0), 2, 4)
```

property vradius

This Ellipse property is an alias for the Circle's radius.

Whereas hradius, major and minor can use Ellipse's conventions, the vradius does not exist for a circle. It is always a positive value in order that the Circle, like Polygons, will have an area that can be positive or negative as determined by the sign of the hradius.

Examples

```
>>> from sympy import Point, Circle
>>> c1 = Circle(Point(3, 4), 6)
>>> c1.vradius
6
```

Polygons

class sympy.geometry.polygon.**Polygon**(**args*, *n=0*, ***kwargs*)

A two-dimensional polygon.

A simple polygon in space. Can be constructed from a sequence of points or from a center, radius, number of sides and rotation angle.

Parameters

vertices : sequence of Points

Optional parameters

==========

n : If > 0, an n-sided RegularPolygon is created. See below.

Default value is 0.

Raises

GeometryError

If all parameters are not Points.

Notes

Polygons are treated as closed paths rather than 2D areas so some calculations can be be negative or positive (e.g., area) based on the orientation of the points.

Any consecutive identical points are reduced to a single point and any points collinear and between two points will be removed unless they are needed to define an explicit intersection (see examples).

A Triangle, Segment or Point will be returned when there are 3 or fewer points provided.

Examples

```
>>> from sympy import Polygon, pi
>>> p1, p2, p3, p4, p5 = [(0, 0), (1, 0), (5, 1), (0, 1), (3, 0)]
>>> Polygon(p1, p2, p3, p4)
Polygon(Point2D(0, 0), Point2D(1, 0), Point2D(5, 1), Point2D(0, 1))
>>> Polygon(p1, p2)
Segment2D(Point2D(0, 0), Point2D(1, 0))
>>> Polygon(p1, p2, p5)
Segment2D(Point2D(0, 0), Point2D(3, 0))
```

The area of a polygon is calculated as positive when vertices are traversed in a ccw direction. When the sides of a polygon cross the area will have positive and negative contributions. The following defines a Z shape where the bottom right connects back to the top left.

```
>>> Polygon((0, 2), (2, 2), (0, 0), (2, 0)).area
0
```

When the keyword n is used to define the number of sides of the Polygon then a RegularPolygon is created and the other arguments are interpreted as center, radius and rotation. The unrotated RegularPolygon will always have a vertex at Point(r, 0) where r is the radius of the circle that circumscribes the RegularPolygon. Its method $spin$ can be used to increment that angle.

```
>>> p = Polygon((0,0), 1, n=3)
>>> p
RegularPolygon(Point2D(0, 0), 1, 3, 0)
>>> p.vertices[0]
Point2D(1, 0)
>>> p.args[0]
Point2D(0, 0)
>>> p.spin(pi/2)
>>> p.vertices[0]
Point2D(0, 1)
```

See also:

sympy.geometry.point.Point (page 2202), *sympy.geometry.line.Segment* (page 2233), *Triangle* (page 2293)

Attributes

area	
angles	
perimeter	
vertices	
centroid	
sides	

property angles

The internal angle at each vertex.

Returns

angles : dict

A dictionary where each key is a vertex and each value is the internal angle at that vertex. The vertices are represented as Points.

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.angles[p1]
pi/2
>>> poly.angles[p2]
acos(-4*sqrt(17)/17)
```

See also:

sympy.geometry.point.Point (page 2202), *sympy.geometry.line.LinearEntity.angle_between* (page 2216)

arbitrary_point(*parameter='t'*)

A parameterized point on the polygon.

The parameter, varying from 0 to 1, assigns points to the position on the perimeter that is that fraction of the total perimeter. So the point evaluated at t=1/2 would return the point from the first vertex that is 1/2 way around the polygon.

Parameters

parameter : str, optional

Default value is 't'.

Returns

arbitrary_point : Point

Raises

ValueError

When *parameter* already appears in the Polygon's definition.

Examples

```
>>> from sympy import Polygon, Symbol
>>> t = Symbol('t', real=True)
>>> tri = Polygon((0, 0), (1, 0), (1, 1))
>>> p = tri.arbitrary_point('t')
>>> perimeter = tri.perimeter
>>> s1, s2 = [s.length for s in tri.sides[:2]]
>>> p.subs(t, (s1 + s2/2)/perimeter)
Point2D(1, 1/2)
```

See also:

sympy.geometry.point.Point (page 2202)

property area

The area of the polygon.

Notes

The area calculation can be positive or negative based on the orientation of the points. If any side of the polygon crosses any other side, there will be areas having opposite signs.

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.area
3
```

In the Z shaped polygon (with the lower right connecting back to the upper left) the areas cancel out:

```
>>> Z = Polygon((0, 1), (1, 1), (0, 0), (1, 0))
>>> Z.area
0
```

In the M shaped polygon, areas do not cancel because no side crosses any other (though there is a point of contact).

```
>>> M = Polygon((0, 0), (0, 1), (2, 0), (3, 1), (3, 0))
>>> M.area
-3/2
```

See also:

sympy.geometry.ellipse.Ellipse.area (page 2255)

bisectors(*prec=None*)

Returns angle bisectors of a polygon. If prec is given then approximate the point defining the ray to that precision.

The distance between the points defining the bisector ray is 1.

Examples

```
>>> from sympy import Polygon, Point
>>> p = Polygon(Point(0, 0), Point(2, 0), Point(1, 1), Point(0, 3))
>>> p.bisectors(2)
{Point2D(0, 0): Ray2D(Point2D(0, 0), Point2D(0.71, 0.71)),
 Point2D(0, 3): Ray2D(Point2D(0, 3), Point2D(0.23, 2.0)),
 Point2D(1, 1): Ray2D(Point2D(1, 1), Point2D(0.19, 0.42)),
 Point2D(2, 0): Ray2D(Point2D(2, 0), Point2D(1.1, 0.38))}
```

property bounds

Return a tuple (xmin, ymin, xmax, ymax) representing the bounding rectangle for the geometric figure.