This module implements points in hyperbolic space of arbitrary dimension. It also contains the implementations for specific models of hyperbolic geometry.

This module also implements ideal points in hyperbolic space of arbitrary dimension. It also contains the implementations for specific models of hyperbolic geometry.

Note that not all models of hyperbolic space are bounded, meaning that the ideal boundary is not the topological boundary of the set underlying the model. For example, the unit disk model is bounded with boundary given by the unit sphere. The hyperboloid model is not bounded.

AUTHORS:

- Greg Laun (2013): initial version

EXAMPLES:

We can construct points in the upper half plane model, abbreviated UHP for convenience:

```sage
sage: UHP = HyperbolicPlane().UHP()
sage: UHP.get_point(2 + I)
Point in UHP I + 2
sage: g = UHP.get_point(3 + I)
sage: g.dist(UHP.get_point(I))
arccosh(11/2)
```

We can also construct boundary points in the upper half plane model:

```sage
sage: UHP.get_point(3)
Boundary point in UHP 3
```

Some more examples:

```sage
sage: HyperbolicPlane().UHP().get_point(0)
Boundary point in UHP 0
sage: HyperbolicPlane().PD().get_point(I/2)
Point in PD 1/2*I
sage: HyperbolicPlane().KM().get_point((0,1))
Boundary point in KM (0, 1)
sage: HyperbolicPlane().HM().get_point((0,0,1))
Point in HM (0, 0, 1)
```
class sage.geometry.hyperbolic_space.hyperbolic_point.HyperbolicPoint(model, coordinates, is_boundary, check=True, **graphics_options)

Bases: sage.structure.element.Element

Abstract base class for hyperbolic points. This class should never be instantiated.

INPUT:

- model – the model of the hyperbolic space
- coordinates – the coordinates of a hyperbolic point in the appropriate model
- is_boundary – whether the point is a boundary point
- check – (default: True) if True, then check to make sure the coordinates give a valid point in the model

EXAMPLES:

Comparison between different models is performed via coercion:

```python
sage: UHP = HyperbolicPlane().UHP()
sage: p = UHP.get_point(.2 + .3*I); p
Point in UHP 0.200000000000000 + 0.300000000000000*I
sage: PD = HyperbolicPlane().PD()
sage: q = PD.get_point(0.2 + 0.3*I); q
Point in PD 0.200000000000000 + 0.300000000000000*I

sage: p == q
False
sage: PD(p)
Point in PD 0.231213872832370 - 0.502890173410405*I
sage: bool(p.coordinates() == q.coordinates())
True
```

Similarly for boundary points:

```python
sage: p = UHP.get_point(-1); p
Boundary point in UHP -1
sage: q = PD.get_point(-1); q
Boundary point in PD -1
sage: p == q
True
sage: PD(p)
Boundary point in PD -1
```

It is an error to specify a point that does not lie in the appropriate model:

```python
sage: HyperbolicPlane().UHP().get_point(0.2 - 0.3*I)
Traceback (most recent call last):
  ... ValueError: 0.200000000000000 - 0.300000000000000*I is not a valid point in the UHP model
```

(continues on next page)
sage: HyperbolicPlane().PD().get_point(1.2)
Traceback (most recent call last):
...  
ValueError: 1.200000000000000 is not a valid point in the PD model

sage: HyperbolicPlane().KM().get_point((1,1))
Traceback (most recent call last):
...  
ValueError: (1, 1) is not a valid point in the KM model

sage: HyperbolicPlane().HM().get_point((1, 1, 1))
Traceback (most recent call last):
...  
ValueError: (1, 1, 1) is not a valid point in the HM model

It is an error to specify an interior point of hyperbolic space as a boundary point:

sage: HyperbolicPlane().UHP().get_point(0.2 + 0.3*I, is_boundary=True)
Traceback (most recent call last):
...  
ValueError: 0.200000000000000 + 0.300000000000000*I is not a valid boundary point...
˓→in the UHP model

coordinates()

Return the coordinates of the point.

EXAMPLES:

sage: HyperbolicPlane().UHP().get_point(2 + I).coordinates()
I + 2

sage: HyperbolicPlane().PD().get_point(1/2 + 1/2*I).coordinates()
1/2*I + 1/2

sage: HyperbolicPlane().KM().get_point((1/3, 1/4)).coordinates()
(1/3, 1/4)

sage: HyperbolicPlane().HM().get_point((0,0,1)).coordinates()
(0, 0, 1)

graphics_options()

Return the graphics options of the current point.

EXAMPLES:

sage: p = HyperbolicPlane().UHP().get_point(2 + I, color="red")
sage: p.graphics_options()
{'color': 'red'}

is_boundary()

Return True if self is a boundary point.

EXAMPLES:
```python
sage: PD = HyperbolicPlane().PD()
sage: p = PD.get_point(0.5+.2*I)
sage: p.is_boundary()
False
sage: p = PD.get_point(I)
sage: p.is_boundary()
True
```

**model()**

Return the model to which the `HyperbolicPoint` belongs.

**EXAMPLES:**

```python
sage: HyperbolicPlane().UHP().get_point(I).model()
Hyperbolic plane in the Upper Half Plane Model
sage: HyperbolicPlane().PD().get_point(0).model()
Hyperbolic plane in the Poincare Disk Model
sage: HyperbolicPlane().KM().get_point((0,0)).model()
Hyperbolic plane in the Klein Disk Model
sage: HyperbolicPlane().HM().get_point((0,0,1)).model()
Hyperbolic plane in the Hyperboloid Model
```

**show**(boundary=True, **options)**

Plot self.

**EXAMPLES:**

```python
sage: HyperbolicPlane().PD().get_point(0).show()
Graphics object consisting of 2 graphics primitives
sage: HyperbolicPlane().KM().get_point((0,0)).show()
Graphics object consisting of 2 graphics primitives
sage: HyperbolicPlane().HM().get_point((0,0,1)).show()
Graphics3d Object
```

**symmetry_involution()**

Return the involutory isometry fixing the given point.

**EXAMPLES:**

```python
sage: z = HyperbolicPlane().UHP().get_point(3 + 2*I)
sage: z.symmetry_involution()
Isometry in UHP
[  3/2  -13/2]
[  1/2   -3/2]
sage: HyperbolicPlane().UHP().get_point(I).symmetry_involution()
Isometry in UHP
[   0    -1]
[   1     0]
sage: HyperbolicPlane().PD().get_point(0).symmetry_involution()
Isometry in PD
```

(continues on next page)
\begin{verbatim}
[-I 0]
[ 0 I]
sage: HyperbolicPlane().KM().get_point((0, 0)).symmetry_involution()
Isometry in KM
[ -1 0 0]
[ 0 -1 0]
[ 0 0 1]
sage: HyperbolicPlane().HM().get_point((0,0,1)).symmetry_involution()
Isometry in HM
[ -1 0 0]
[ 0 -1 0]
[ 0 0 1]
sage: p = HyperbolicPlane().UHP().random_element()
sage: A = p.symmetry_involution()
sage: p.dist(A*p)  # abs tol 1e-10
0
sage: A.preserves_orientation()
True
sage: A*A == HyperbolicPlane().UHP().get_isometry(identity_matrix(2))
True
\end{verbatim}

\texttt{to\_model}(model)

Convert self to the model.

INPUT:

* other – (a string representing) the image model

EXAMPLES:

\begin{verbatim}
sage: UHP = HyperbolicPlane().UHP()
sage: PD = HyperbolicPlane().PD()
sage: PD.get_point(1/2+I/2).to_model(UHP)
Point in UHP I + 2
\end{verbatim}

\texttt{update\_graphics}(update=False, **options)

Update the graphics options of a \texttt{HyperbolicPoint}. If update is True, update rather than overwrite.

EXAMPLES:

\begin{verbatim}
sage: p = HyperbolicPlane().UHP().get_point(I); p.graphics_options()
{}
sage: p.update_graphics(color = "red"); p.graphics_options()
{'color': 'red'}
sage: p.update_graphics(color = "blue"); p.graphics_options()
\end{verbatim}

(continues on next page)
class sage.geometry.hyperbolic_space.hyperbolic_point.HyperbolicPointUHP(model, coordinates, is_boundary, check=True, **graphics_options)

A point in the UHP model.

INPUT:

- the coordinates of a point in the unit disk in the complex plane $\mathbb{C}$

EXAMPLES:

```python
sage: HyperbolicPlane().UHP().get_point(2*I)
Point in UHP 2*I
sage: HyperbolicPlane().UHP().get_point(1)
Boundary point in UHP 1
```

`show(boundary=True, **options)`

Plot self.

EXAMPLES:

```python
sage: HyperbolicPlane().UHP().get_point(I).show()
Graphics object consisting of 2 graphics primitives
sage: HyperbolicPlane().UHP().get_point(0).show()
Graphics object consisting of 2 graphics primitives
sage: HyperbolicPlane().UHP().get_point(infinity).show()
Traceback (most recent call last):
... NotImplementedError: can't draw the point infinity
```

`symmetry_involution()`

Return the involutory isometry fixing the given point.

EXAMPLES:

```python
sage: HyperbolicPlane().UHP().get_point(3 + 2*I).symmetry_involution()
Isometry in UHP
[ 3/2  -13/2]
[  1/2   -3/2]
```
CHAPTER
TWO

HYPERBOLIC ISOMETRIES

This module implements the abstract base class for isometries of hyperbolic space of arbitrary dimension. It also contains the implementations for specific models of hyperbolic geometry.

The isometry groups of all implemented models are either matrix Lie groups or are doubly covered by matrix Lie groups. As such, the isometry constructor takes a matrix as input. However, since the isometries themselves may not be matrices, quantities like the trace and determinant are not directly accessible from this class.

AUTHORS:

- Greg Laun (2013): initial version

EXAMPLES:

We can construct isometries in the upper half plane model, abbreviated UHP for convenience:

```
sage: UHP = HyperbolicPlane().UHP()
sage: UHP.get_isometry(matrix(2,[1,2,3,4]))
Isometry in UHP
[1 2]
[3 4]
sage: A = UHP.get_isometry(matrix(2,[0,1,1,0]))
sage: A.inverse()
Isometry in UHP
[0 1]
[1 0]
```

class sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometry(model, A, check=True)

Bases: sage.categories.morphism.Morphism

Abstract base class for hyperbolic isometries. This class should never be instantiated.

INPUT:

- A – a matrix representing a hyperbolic isometry in the appropriate model

EXAMPLES:

```
sage: HyperbolicPlane().HM().get_isometry(identity_matrix(3))
Isometry in HM
[1 0 0]
[0 1 0]
[0 0 1]
```

attracting_fixed_point()

For a hyperbolic isometry, return the attracting fixed point; otherwise raise a ValueError.
• a hyperbolic point

EXAMPLES:
```
sage: UHP = HyperbolicPlane().UHP()
sage: A = UHP.get_isometry(Matrix(2,[4,0,0,1/4]))
sage: A.attracting_fixed_point()
Boundary point in UHP +Infinity
```

axis()
For a hyperbolic isometry, return the axis of the transformation; otherwise raise a `ValueError`.

EXAMPLES:
```
sage: UHP = HyperbolicPlane().UHP()
sage: H = UHP.get_isometry(matrix(2,[2,0,0,1/2]))
sage: H.axis()
Geodesic in UHP from 0 to +Infinity
```

It is an error to call this function on an isometry that is not hyperbolic:
```
sage: P = UHP.get_isometry(matrix(2,[1,4,0,1]))
sage: P.axis()
Traceback (most recent call last):
...
ValueError: the isometry is not hyperbolic: axis is undefined
```

classification()
Classify the hyperbolic isometry as elliptic, parabolic, hyperbolic or a reflection.
A hyperbolic isometry fixes two points on the boundary of hyperbolic space, a parabolic isometry fixes one point on the boundary of hyperbolic space, and an elliptic isometry fixes no points.

EXAMPLES:
```
sage: UHP = HyperbolicPlane().UHP()
sage: H = UHP.get_isometry(matrix(2,[2,0,0,1/2]))
sage: H.classification()
'hyperbolic'
sage: P = UHP.get_isometry(matrix(2,[1,1,0,1]))
sage: P.classification()
'parabolic'
sage: E = UHP.get_isometry(matrix(2,[-1,0,0,1]))
sage: E.classification()
'reflection'
```

fixed_geodesic()
If self is a reflection in a geodesic, return that geodesic.

EXAMPLES:
```
sage: A = HyperbolicPlane().PD().get_isometry(matrix([[0, 1], [1, 0]]))
sage: A.fixed_geodesic()
Geodesic in PD from -1 to 1
```
fixed_point_set()

Return a list containing the fixed point set of orientation-preserving isometries.

OUTPUT:
list of hyperbolic points or a hyperbolic geodesic

EXAMPLES:

```
sage: KM = HyperbolicPlane().KM()
sage: H = KM.get_isometry(matrix([[5/3,0,4/3], [0,1,0], [4/3,0,5/3]]))
sage: g = H.fixed_point_set(); g
Geodesic in KM from (1, 0) to (-1, 0)
sage: H(g.start()) == g.start()
True
sage: H(g.end()) == g.end()
True
sage: A = KM.get_isometry(matrix([[1,0,0], [0,-1,0], [0,0,1]]))
sage: A.preserves_orientation()
False
sage: A.fixed_point_set()
Geodesic in KM from (1, 0) to (-1, 0)
```

```
sage: B = KM.get_isometry(identity_matrix(3))
sage: B.fixed_point_set()
Traceback (most recent call last):
... ValueError: the identity transformation fixes the entire hyperbolic plane
```

inverse()

Return the inverse of the isometry self.

EXAMPLES:

```
sage: UHP = HyperbolicPlane().UHP()
sage: A = UHP.get_isometry(matrix(2,[4,1,3,2]))
sage: B = A.inverse()
sage: A*B == UHP.get_isometry(identity_matrix(2))
True
```

is_identity()

Return True if self is the identity isometry.

EXAMPLES:

```
sage: UHP = HyperbolicPlane().UHP()
sage: UHP.get_isometry(matrix(2,[4,1,3,2])).is_identity()
False
sage: UHP.get_isometry(identity_matrix(2)).is_identity()
True
```

matrix()

Return the matrix of the isometry.
Note: We do not allow the matrix constructor to work as these may be elements of a projective group (ex. \(PSL(n, \mathbb{R})\)), so these isometries aren’t true matrices.

EXAMPLES:

```
sage: UHP = HyperbolicPlane().UHP()
sage: UHP.get_isometry(-identity_matrix(2)).matrix()
[-1 0]
[ 0 -1]
```

**model()**

Return the model to which `self` belongs.

EXAMPLES:

```
sage: HyperbolicPlane().UHP().get_isometry(identity_matrix(2)).model()
Hyperbolic plane in the Upper Half Plane Model
sage: HyperbolicPlane().PD().get_isometry(identity_matrix(2)).model()
Hyperbolic plane in the Poincare Disk Model
sage: HyperbolicPlane().KM().get_isometry(identity_matrix(3)).model()
Hyperbolic plane in the Klein Disk Model
sage: HyperbolicPlane().HM().get_isometry(identity_matrix(3)).model()
Hyperbolic plane in the Hyperboloid Model
```

**preserves_orientation()**

Return `True` if `self` is orientation-preserving and `False` otherwise.

EXAMPLES:

```
sage: UHP = HyperbolicPlane().UHP()
sage: A = UHP.get_isometry(identity_matrix(2))
sage: A.preserves_orientation()
True
sage: B = UHP.get_isometry(matrix(2,[0,1,1,0]))
sage: B.preserves_orientation()
False
```

**repelling_fixed_point()**

For a hyperbolic isometry, return the attracting fixed point; otherwise raise a `ValueError`.

OUTPUT:

• a hyperbolic point

EXAMPLES:

```
sage: UHP = HyperbolicPlane().UHP()
sage: A = UHP.get_isometry(Matrix(2,[4,0,0,1/4]))
sage: A.repelling_fixed_point()
Boundary point in UHP 0
```

**to_model(other)**

Convert the current object to image in another model.
INPUT:

• other – (a string representing) the image model

EXAMPLES:

```sage
sage: H = HyperbolicPlane()
sage: UHP = H.UHP()
sage: PD = H.PD()
sage: KM = H.KM()
sage: HM = H.HM()

sage: A = UHP.get_isometry(identity_matrix(2))
sage: A.to_model(HM)
Isometry in HM
[1 0 0]
[0 1 0]
[0 0 1]
sage: A.to_model('HM')
Isometry in HM
[1 0 0]
[0 1 0]
[0 0 1]

sage: A = PD.get_isometry(matrix([[I, 0], [0, -I]]))
sage: A.to_model(UHP)
Isometry in UHP
[ 0 1]
[-1 0]
sage: A.to_model(HM)
Isometry in HM
[-1 0 0]
[ 0 -1 0]
[ 0 0 1]
sage: A.to_model(KM)
Isometry in KM
[-1 0 0]
[ 0 -1 0]
[ 0 0 1]

sage: A = HM.get_isometry(diagonal_matrix([-1, -1, 1]))
sage: A.to_model('UHP')
Isometry in UHP
[ 0 -1]
[ 1 0]
sage: A.to_model('PD')
Isometry in PD
[-I 0]
[ 0 I]
sage: A.to_model('KM')
Isometry in KM
[-1 0 0]
[ 0 -1 0]
[ 0 0 1]
```
translation_length()
For hyperbolic elements, return the translation length; otherwise, raise a ValueError.

EXAMPLES:

```
sage: UHP = HyperbolicPlane().UHP()
sage: H = UHP.get_isometry(matrix(2,[2,0,0,1/2]))
sage: H.translation_length()
2*arccosh(5/4)
```

```
sage: f_1 = UHP.get_point(-1)
sage: f_2 = UHP.get_point(1)
sage: H = UHP.isometry_from_fixed_points(f_1, f_2)
sage: p = UHP.get_point(exp(i*7*pi/8))
sage: bool((p.dist(H*p) - H.translation_length()) < 10**-9)
True
```
class sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometryKM(model, A, check=True)

Bases: sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometry

Create a hyperbolic isometry in the KM model.

INPUT:

• a matrix in $SO(2,1)$

EXAMPLES:

```
sage: HyperbolicPlane().KM().get_isometry(identity_matrix(3))
Isometry in KM
[1 0 0]
[0 1 0]
[0 0 1]
```
class sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometryPD(model, A, check=True)

Bases: sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometry

Create a hyperbolic isometry in the PD model.

INPUT:

• a matrix in $PU(1,1)$

EXAMPLES:

```
sage: HyperbolicPlane().PD().get_isometry(identity_matrix(2))
Isometry in PD
[1 0]
[0 1]
```
preserves_orientation()
Return True if self preserves orientation and False otherwise.

EXAMPLES:
class sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometryUHP(model, A, check=True)

Bases: sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometry

Create a hyperbolic isometry in the UHP model.

INPUT:
- a matrix in $GL(2, \mathbb{R})$

EXAMPLES:

```python
sage: HyperbolicPlane().UHP().get_isometry(identity_matrix(2))
Isometry in UHP
[1 0]
[0 1]
```

attracting_fixed_point()

Return the attracting fixed point; otherwise raise a ValueError.

OUTPUT:
- a hyperbolic point

EXAMPLES:

```python
sage: UHP = HyperbolicPlane().UHP()
sage: A = matrix(2, [4, 0, 0, 1/4])
sage: UHP.get_isometry(A).attracting_fixed_point()
Boundary point in UHP +Infinity
```

classification()

Classify the hyperbolic isometry as elliptic, parabolic, or hyperbolic.

A hyperbolic isometry fixes two points on the boundary of hyperbolic space, a parabolic isometry fixes one point on the boundary of hyperbolic space, and an elliptic isometry fixes no points.

EXAMPLES:

```python
sage: UHP = HyperbolicPlane().UHP()
sage: UHP.get_isometry(identity_matrix(2)).classification()
'identity'
sage: UHP.get_isometry(4*identity_matrix(2)).classification()
'identity'
sage: UHP.get_isometry(matrix(2, [2, 0, 0, 1/2])).classification()
'hyperbolic'
sage: UHP.get_isometry(matrix(2, [0, 3, -1/3, 6])).classification()
'hyperbolic'
```
fixed_point_set()

Return a list or geodesic containing the fixed point set of orientation-preserving isometries.

OUTPUT:
list of hyperbolic points or a hyperbolic geodesic

EXAMPLES:

```
sage: UHP = HyperbolicPlane().UHP()
sage: H = UHP.get_isometry(matrix(2, [-2/3,-1/3,-1/3,-2/3]))
sage: g = H.fixed_point_set(); g
Geodesic in UHP from -1 to 1
sage: H(g.start()) == g.start()
True
sage: H(g.end()) == g.end()
True
sage: A = UHP.get_isometry(matrix(2,[0,1,1,0]))
sage: A.preserves_orientation()
False
sage: A.fixed_point_set()
Geodesic in UHP from 1 to -1
```

repelling_fixed_point()

Return the repelling fixed point; otherwise raise a ValueError.

OUTPUT:
• a hyperbolic point

preserves_orientation()

Return True if self is orientation-preserving and False otherwise.

EXAMPLES:

```
sage: UHP = HyperbolicPlane().UHP()
sage: A = identity_matrix(2)
sage: UHP.get_isometry(A).preserves_orientation()
True
sage: B = matrix(2,[0,1,1,0])
sage: UHP.get_isometry(B).preserves_orientation()
False
```

UHP.get_isometry(matrix(2,[1,1,0,1])).classification()

'parabolic'

UHP.get_isometry(matrix(2,[1,1,0,1])).classification()

'reflection'
EXAMPLES:

```python
sage: UHP = HyperbolicPlane().UHP()
sage: A = matrix(2, [4, 0, 0, 1/4])
sage: UHP.get_isometry(A).repelling_fixed_point()
Boundary point in UHP 0
```

**translation_length()**

For hyperbolic elements, return the translation length; otherwise, raise a `ValueError`.

EXAMPLES:

```python
sage: UHP = HyperbolicPlane().UHP()
sage: UHP.get_isometry(matrix(2, [2, 0, 0, 1/2])).translation_length()
2*arccosh(5/4)
```

```
sage: H = UHP.isometry_from_fixed_points(-1, 1)
sage: p = UHP.get_point(exp(i*7*pi/8))
sage: Hp = H(p)
sage: bool((UHP.dist(p, Hp) - H.translation_length()) < 10**-9)
True
```

sage.geometry.hyperbolic_space.hyperbolic_isometry.moebius_transform(A, z)

Given a matrix `A` in $GL(2, \mathbb{C})$ and a point `z` in the complex plane return the Möbius transformation action of `A` on `z`.

**INPUT:**

- `A` – a $2 \times 2$ invertible matrix over the complex numbers
- `z` – a complex number or infinity

**OUTPUT:**

- a complex number or infinity

EXAMPLES:

```python
sage: from sage.geometry.hyperbolic_space.hyperbolic_isometry import moebius_transform
sage: moebius_transform(matrix(2, [1, 2, 3, 4]), 2 + I)
-2/109*I + 43/109
```

```
sage: y = var('y')
sage: moebius_transform(matrix(2, [1, 0, 0, 1]), x + I*y)
x + I*y
```

The matrix must be square and $2 \times 2$:

```python
sage: moebius_transform(matrix([[3, 1, 2], [1, 2, 5]]), I)
Traceback (most recent call last):
  ... TypeError: A must be an invertible 2x2 matrix over the complex numbers or a
  ~ symbolic ring
```

```python
sage: moebius_transform(identity_matrix(3), I)
Traceback (most recent call last):
  ... TypeError: A must be an invertible 2x2 matrix over the complex numbers or a
  ~ symbolic ring
```
The matrix can be symbolic or can be a matrix over the real or complex numbers, but must be provably invertible:

```python
sage: a, b, c, d = var('a, b, c, d')
sage: moebius_transform(matrix(2, [a, b, c, d]), I)
(I*a + b)/(I*c + d)
sage: moebius_transform(matrix(2, [1, b, c, b*c+1]), I)
(b + I)/(b*c + I*c + 1)
sage: moebius_transform(matrix(2, [0, 0, 0, 0]), I)
Traceback (most recent call last):
...
TypeError: A must be an invertible 2x2 matrix over the complex numbers or a symbolic ring
```
This module implements the abstract base class for geodesics in hyperbolic space of arbitrary dimension. It also contains the implementations for specific models of hyperbolic geometry.

AUTHORS:

• Greg Laun (2013): initial version

EXAMPLES:

We can construct geodesics in the upper half plane model, abbreviated UHP for convenience:

```sage
g = HyperbolicPlane().UHP().get_geodesic(2, 3)
g
Geodesic in UHP from 2 to 3
```

This geodesic can be plotted using `plot()`, in this example we will show the axis.

```sage
g.plot(axes=True)  # optional - sage.plot
Graphics object consisting of 2 graphics primitives
```

```sage
g = HyperbolicPlane().UHP().get_geodesic(I, 3 + I)
g.length()
arccosh(11/2)
g.plot(axes=True)  # optional - sage.plot
Graphics object consisting of 2 graphics primitives
```

Geodesics of both types in UHP are supported:

```sage
g = HyperbolicPlane().UHP().get_geodesic(I, 3*I)
g
Geodesic in UHP from I to 3*I
g.plot()  # optional - sage.plot
Graphics object consisting of 2 graphics primitives
```

Geodesics are oriented, which means that two geodesics with the same graph will only be equal if their starting and ending points are the same:

```sage
g1 = HyperbolicPlane().UHP().get_geodesic(1, 2)
g2 = HyperbolicPlane().UHP().get_geodesic(2, 1)
g1 == g2
False
```
class sage.geometry.hyperbolic_space.hyperbolic_geodesic.HyperbolicGeodesic(model, start, end, **graphics_options)

Bases: sage.structure.sage_object.SageObject

Abstract base class for oriented geodesics that are not necessarily complete.

INPUT:

- start – a HyperbolicPoint or coordinates of a point in hyperbolic space representing the start of the geodesic
- end – a HyperbolicPoint or coordinates of a point in hyperbolic space representing the end of the geodesic

EXAMPLES:

We can construct a hyperbolic geodesic in any model:

\begin{verbatim}
sage: HyperbolicPlane().UHP().get_geodesic(1, 0)
Geodesic in UHP from 1 to 0
sage: HyperbolicPlane().PD().get_geodesic(1, 0)
Geodesic in PD from 1 to 0
sage: HyperbolicPlane().KM().get_geodesic((0,1/2), (1/2, 0))
Geodesic in KM from (0, 1/2) to (1/2, 0)
sage: HyperbolicPlane().HM().get_geodesic((0,0,1), (0,1, sqrt(2)))
Geodesic in HM from (0, 0, 1) to (0, 1, sqrt(2))
\end{verbatim}

angle(other)

Return the angle between any two given geodesics if they intersect.

INPUT:

- other – a hyperbolic geodesic in the same model as self

OUTPUT:

- the angle in radians between the two given geodesics

EXAMPLES:

\begin{verbatim}
sage: PD = HyperbolicPlane().PD()
sage: g = PD.get_geodesic(3/5*I + 4/5, 15/17*I + 8/17)
sage: h = PD.get_geodesic(4/5*I + 3/5, I)
sage: g.angle(h)
1/2*pi
\end{verbatim}

common_perpendicular(other)

Return the unique hyperbolic geodesic perpendicular to two given geodesics, if such a geodesic exists. If none exists, raise a ValueError.

INPUT:

- other – a hyperbolic geodesic in the same model as self

OUTPUT:

- a hyperbolic geodesic
EXAMPLES:

```python
sage: g = HyperbolicPlane().UHP().get_geodesic(2,3)
sage: h = HyperbolicPlane().UHP().get_geodesic(4,5)
sage: g.common_perpendicular(h)
Geodesic in UHP from 1/2*sqrt(3) + 7/2 to -1/2*sqrt(3) + 7/2
```

It is an error to ask for the common perpendicular of two intersecting geodesics:

```python
sage: g = HyperbolicPlane().UHP().get_geodesic(2,4)
sage: h = HyperbolicPlane().UHP().get_geodesic(3, infinity)
sage: g.common_perpendicular(h)
Traceback (most recent call last):
...  
ValueError: geodesics intersect; no common perpendicular exists
```

```python
complete()
```

Return the geodesic with ideal endpoints in bounded models. Raise a `NotImplementedError` in models that are not bounded. In the following examples we represent complete geodesics by a dashed line.

EXAMPLES:

```python
sage: H = HyperbolicPlane()
sage: UHP = H.UHP()
```
sage: UHP.get_geodesic(1 + I, 1 + 3*I).complete()
Geodesic in UHP from 1 to +Infinity

sage: PD = H.PD()
sage: PD.get_geodesic(0, I/2).complete()
Geodesic in PD from -I to I
sage: PD.get_geodesic(0.25*(-1-I),0.25*(1-I)).complete()
Geodesic in PD from -0.895806416477617 - 0.444444444444444*I to 0.
→ 0.895806416477617 - 0.444444444444444*I

sage: KM = H.KM()
sage: KM.get_geodesic((0,0), (0, 1/2)).complete()
Geodesic in KM from (0, -1) to (0, 1)

sage: HM = H.HM()
sage: HM.get_geodesic((0,0,1), (1, 0, sqrt(2))).complete()
Geodesic in HM from (0, 0, 1) to (1, 0, sqrt(2))

sage: g = HM.get_geodesic((0,0,1), (1, 0, sqrt(2))).complete()
trace(g.is_complete())
True
dist(other)
Return the hyperbolic distance from a given hyperbolic geodesic to another geodesic or point.

INPUT:
• other – a hyperbolic geodesic or hyperbolic point in the same model

OUTPUT:
• the hyperbolic distance

EXAMPLES:

\[
\begin{align*}
\text{sage: } & g = \text{HyperbolicPlane().UHP().get_geodesic(2, 4.0)} \\
\text{sage: } & h = \text{HyperbolicPlane().UHP().get_geodesic(5, 7.0)} \\
\text{sage: } & \text{bool(abs(g.dist(h).n() - 1.92484730023841) < 10^{-9})} \\
& True
\end{align*}
\]

If the second object is a geodesic ultraparallel to the first, or if it is a point on the boundary that is not one of the first object’s endpoints, then return +infinity

\[
\begin{align*}
\text{sage: } & g = \text{HyperbolicPlane().UHP().get_geodesic(2, 2+I)} \\
\text{sage: } & p = \text{HyperbolicPlane().UHP().get_point(5)} \\
\text{sage: } & g.dist(p) \\
& +\infty
\end{align*}
\]

end()
Return the starting point of the geodesic.

EXAMPLES:

\[
\begin{align*}
\text{sage: } & g = \text{HyperbolicPlane().UHP().get_geodesic(I, 3*I)} \\
\text{sage: } & g.end() \\
& \text{Point in UHP 3*I}
\end{align*}
\]

endpoints()
Return a list containing the start and endpoints.

EXAMPLES:

\[
\begin{align*}
\text{sage: } & g = \text{HyperbolicPlane().UHP().get_geodesic(I, 3*I)} \\
\text{sage: } & g.endpoints() \\
& [\text{Point in UHP I, Point in UHP 3*I}]
\end{align*}
\]

graphics_options()
Return the graphics options of self.

EXAMPLES:

\[
\begin{align*}
\text{sage: } & g = \text{HyperbolicPlane().UHP().get_geodesic(I, 2*I, color="red")} \\
\text{sage: } & g.graphics_options() \\
& \{'color': 'red'}
\end{align*}
\]

ideal_endpoints()
Return the ideal endpoints in bounded models. Raise a \texttt{NotImplementedError} in models that are not bounded.

EXAMPLES:
intersection(other)
Return the point of intersection of two geodesics (if such a point exists).

INPUT:
• other – a hyperbolic geodesic in the same model as self

OUTPUT:
• a hyperbolic point or geodesic

EXAMPLES:

```sage
PD = HyperbolicPlane().PD()
```

is_asymptotically_parallel(other)
Return True if self and other are asymptotically parallel and False otherwise.

INPUT:
• other – a hyperbolic geodesic

EXAMPLES:

```sage
g = HyperbolicPlane().UHP().get_geodesic(-2,5)
h = HyperbolicPlane().UHP().get_geodesic(-2,4)
g.is_asymptotically_parallel(h)
```
True

Ultraparallel geodesics are not asymptotically parallel:

```sage
g = HyperbolicPlane().UHP().get_geodesic(-2,5)
h = HyperbolicPlane().UHP().get_geodesic(-1,4)
g.is_asymptotically_parallel(h)
```
False

No hyperbolic geodesic is asymptotically parallel to itself:
```
sage: g = HyperbolicPlane().UHP().get_geodesic(-2,5)
sage: g.is_asymptotically_parallel(g)
False
```

**is_complete()**

Return `True` if `self` is a complete geodesic (that is, both endpoints are on the ideal boundary) and `False` otherwise.

If we represent complete geodesics using green color and incomplete using red colors we have the following graphic:

Notice, that there is no visual indication that the *vertical* geodesic is complete

**EXAMPLES:**

```
sage: UHP = HyperbolicPlane().UHP()
sage: UHP.get_geodesic(1.5*I, 2.5*I).is_complete()
False
sage: UHP.get_geodesic(0, I).is_complete()
False
sage: UHP.get_geodesic(3, infinity).is_complete()
True
sage: UHP.get_geodesic(2,5).is_complete()
True
```
is_parallel(other)
Return True if the two given hyperbolic geodesics are either ultra parallel or asymptotically parallel and False otherwise.

INPUT:
• other – a hyperbolic geodesic in any model

OUTPUT:
True if the given geodesics are either ultra parallel or asymptotically parallel, False if not.

EXAMPLES:

```python
sage: g = HyperbolicPlane().UHP().get_geodesic(-2,5)
sage: h = HyperbolicPlane().UHP().get_geodesic(5,12)
sage: g.is_parallel(h)
True
```

```python
sage: g = HyperbolicPlane().UHP().get_geodesic(-2,5)
sage: h = HyperbolicPlane().UHP().get_geodesic(-2,4)
sage: g.is_parallel(h)
True
```
sage: g = HyperbolicPlane().UHP().get_geodesic(-2,2)
sage: h = HyperbolicPlane().UHP().get_geodesic(-1,4)
sage: g.is_parallel(h)
False

No hyperbolic geodesic is either ultra parallel or asymptotically parallel to itself:

sage: g = HyperbolicPlane().UHP().get_geodesic(-2,5)
sage: g.is_parallel(g)
False

.. function:: is ultra parallel(other)

   Return True if self and other are ultra parallel and False otherwise.

   INPUT:

   * other -- a hyperbolic geodesic

   EXAMPLES:

sage: from sage.geometry.hyperbolic_space.hyperbolic_geodesic \  
.....: import *
sage: g = HyperbolicPlane().UHP().get_geodesic(0,1)
sage: h = HyperbolicPlane().UHP().get_geodesic(-3,-1)
(continues on next page)
sage: g.is_ultra_parallel(h)
True

sage: h = HyperbolicPlane().UHP().get_geodesic(2,6)
sage: g.is_ultra_parallel(h)
False

sage: g = HyperbolicPlane().UHP().get_geodesic(-2,5)
sage: g.is_ultra_parallel(g)
False

length()

Return the Hyperbolic length of the hyperbolic line segment.

EXAMPLES:

sage: g = HyperbolicPlane().UHP().get_geodesic(2 + I, 3 + I/2)
sage: g.length()
arccosh(9/4)

midpoint()

Return the (hyperbolic) midpoint of a hyperbolic line segment.
Complete geodesics have no midpoint:

```python
sage: HyperbolicPlane().UHP().get_geodesic(0,2).midpoint()
Traceback (most recent call last):
  ... ValueError: the length must be finite
```

model()

Return the model to which the HyperbolicGeodesic belongs.

EXAMPLES:

```python
sage: UHP = HyperbolicPlane().UHP()
sage: UHP.get_geodesic(1, 2*I).model()
Hyperbolic plane in the Upper Half Plane Model
sage: PD = HyperbolicPlane().PD()
sage: PD.get_geodesic(0, I/2).model()
Hyperbolic plane in the Poincare Disk Model
sage: KM = HyperbolicPlane().KM()
sage: KM.get_geodesic((0, 0), (0, 1/2)).model()
Hyperbolic plane in the Klein Disk Model
sage: HM = HyperbolicPlane().HM()
sage: HM.get_geodesic((0, 0, 1), (0, 1, sqrt(2))).model()
Hyperbolic plane in the Hyperboloid Model
```

perpendicular_bisector()

Return the perpendicular bisector of self if self has finite length. Here distance is hyperbolic distance.

EXAMPLES:

```python
sage: PD = HyperbolicPlane().PD()
sage: g = PD.get_geodesic(-0.3+0.4*I,+0.7-0.1*I)
sage: h = g.perpendicular_bisector().complete()
sage: P = g.plot(color='blue')+h.plot(color='orange');P
Graphics object consisting of 4 graphics primitives
```

Complete geodesics cannot be bisected:

```python
sage: g = HyperbolicPlane().PD().get_geodesic(0, 1)
sage: g.perpendicular_bisector()
Traceback (most recent call last):
  ... ValueError: the length must be finite
```
**reflection_involution()**

Return the involution fixing self.

**EXAMPLES:**

```sage
data: H = HyperbolicPlane()
sage: gU = H.UHP().get_geodesic(2,4)
sage: RU = gU.reflection_involution(); RU
Isometry in UHP
[ 3 -8]
[ 1 -3]
sage: RU*gU == gU
True

sage: gP = H.PD().get_geodesic(0, I)
sage: RP = gP.reflection_involution(); RP
Isometry in PD
[ 1 0]
[ 0 -1]
sage: RP*gP == gP
True

sage: gK = H.KM().get_geodesic((0,0), (0,1))
sage: RK = gK.reflection_involution(); RK
Isometry in KM
[-1 0 0]
[ 0 1 0]
[ 0 0 1]
sage: RK*gK == gK
True

sage: HM = H.HM()
sage: g = HM.get_geodesic((0,0,1), (1,0, n(sqrt(2))))
sage: A = g.reflection_involution()
sage: B = diagonal_matrix([1, -1, 1])
sage: bool((B - A.matrix()).norm() < 10**-9)
True
```

The above tests go through the Upper Half Plane. It remains to test that the matrices in the models do what we intend.

```sage
data: from sage.geometry.hyperbolic_space.hyperbolic_isometry \"...
...: import moebius_transform
sage: R = H.PD().get_geodesic(-1,1).reflection_involution()

sage: bool(moebius_transform(R.matrix(), 0) == 0)
True
```

**start()**

Return the starting point of the geodesic.

**EXAMPLES:**
sage: g = HyperbolicPlane().UHP().get_geodesic(I, 3*I)
sage: g.start()
Point in UHP I

to_model(model)
Convert the current object to image in another model.

INPUT:
• model – the image model

EXAMPLES:

sage: UHP = HyperbolicPlane().UHP()
sage: PD = HyperbolicPlane().PD()
sage: UHP.get_geodesic(I, 2*I).to_model(PD)
Geodesic in PD from 0 to 1/3*I
sage: UHP.get_geodesic(I, 2*I).to_model('PD')
Geodesic in PD from 0 to 1/3*I

update_graphics(update=False, **options)
Update the graphics options of self.

INPUT:
• update – if True, the original option are updated rather than overwritten

EXAMPLES:

sage: g = HyperbolicPlane().UHP().get_geodesic(I, 2*I)
sage: g.graphics_options()
{}
sage: g.update_graphics(color = "red"); g.graphics_options()
{'color': 'red'}
sage: g.update_graphics(color = "blue"); g.graphics_options()
{'color': 'blue'}
sage: g.update_graphics(True, size = 20); g.graphics_options()
{'color': 'blue', 'size': 20}

class sage.geometry.hyperbolic_space.hyperbolic_geodesic.HyperbolicGeodesicHM(model, start, end, **graphics_options)
Bases: sage.geometry.hyperbolic_space.hyperbolic_geodesic.HyperbolicGeodesic
A geodesic in the hyperboloid model.
Valid points in the hyperboloid model satisfy $x^2 + y^2 - z^2 = -1$

INPUT:
• start – a HyperbolicPoint in hyperbolic space representing the start of the geodesic
• end – a HyperbolicPoint in hyperbolic space representing the end of the geodesic

EXAMPLES:
```sage
from sage.geometry.hyperbolic_space.hyperbolic_geodesic import *
sage: HM = HyperbolicPlane().HM()
sage: p1 = HM.get_point((4, -4, sqrt(33)))
sage: p2 = HM.get_point((-3, -3, sqrt(19)))
sage: g = HM.get_geodesic(p1, p2)
sage: g = HM.get_geodesic((4, -4, sqrt(33)), (-3, -3, sqrt(19)))
```

```
plot(\text{show\_hyperboloid=}True, **\text{graphics\_options})

Plot self.

EXAMPLES:
```sage
from sage.geometry.hyperbolic_space.hyperbolic_geodesic \n......: import *
sage: g = HyperbolicPlane().HM().random_geodesic()
sage: g.plot()  # optional - sage.plot
Graphics3d Object
```

```python
class sage.geometry.hyperbolic_space.hyperbolic_geodesic.HyperbolicGeodesicKM(model, start, end, **graphics_options):

Bases: sage.geometry.hyperbolic_space.hyperbolic_geodesic.HyperbolicGeodesic

A geodesic in the Klein disk model.
```
Geodesics are represented by the chords, straight line segments with ideal endpoints on the boundary circle.

**INPUT:**

- `start` – a `HyperbolicPoint` in hyperbolic space representing the start of the geodesic
- `end` – a `HyperbolicPoint` in hyperbolic space representing the end of the geodesic

**EXAMPLES:**

```python
sage: KM = HyperbolicPlane().KM()
sage: g = KM.get_geodesic(KM.get_point((0.1,0.9)), KM.get_point((-0.1,-0.9)))
sage: g = KM.get_geodesic((0.1,0.9),(-0.1,-0.9))
sage: h = KM.get_geodesic((-0.707106781,-0.707106781),(0.707106781,-0.707106781))
sage: P = g.plot(color='orange')+h.plot(); P  # optional - sage.plot
```

Graphics object consisting of 4 graphics primitives

```
plot(boundary=True, **options)
Plot self.

**EXAMPLES:**

```
sage: HyperbolicPlane().KM().get_geodesic((0,0), (1,0)).plot()  # optional ~ s sage.plot
```

Graphics object consisting of 2 graphics primitives
class sage.geometry.hyperbolic_space.hyperbolic_geodesic.HyperbolicGeodesicPD(model, start, end, **graphics_options)

Bases: sage.geometry.hyperbolic_space.hyperbolic_geodesic.HyperbolicGeodesic

A geodesic in the Poincaré disk model.

Geodesics in this model are represented by segments of circles contained within the unit disk that are orthogonal to the boundary of the disk, plus all diameters of the disk.

INPUT:
- start – a HyperbolicPoint in hyperbolic space representing the start of the geodesic
- end – a HyperbolicPoint in hyperbolic space representing the end of the geodesic

EXAMPLES:

```python
sage: PD = HyperbolicPlane().PD()
sage: g = PD.get_geodesic(PD.get_point(I), PD.get_point(-I/2))
sage: g = PD.get_geodesic(I,-I/2)
sage: h = PD.get_geodesic(-1/2+I/2,1/2+I/2)
```

```python
plot(boundary=True, **options)

Plot self.

EXAMPLES:
```
First some lines:

```python
sage: PD = HyperbolicPlane().PD()
sage: PD.get_geodesic(0, 1).plot()  # optional - sage.plot
Graphics object consisting of 2 graphics primitives
```

```python
sage: PD.get_geodesic(0, 0.3+0.8*I).plot()  # optional - sage.plot
Graphics object consisting of 2 graphics primitives
```

Then some generic geodesics:

```python
sage: PD.get_geodesic(-0.5, 0.3+0.4*I).plot()  # optional - sage.plot
```

```python
sage: g = PD.get_geodesic(-1, exp(3*I*pi/7))
sage: G = g.plot(linestyle="dashed",color="red"); G  # optional - sage.plot
```

```python
sage: h = PD.get_geodesic(exp(2*I*pi/11), exp(1*I*pi/11))
sage: H = h.plot(thickness=6, color="orange"); H  # optional - sage.plot
```

```python
sage: show(G+H)  # optional - sage.plot
```

(continues on next page)
class sage.geometry.hyperbolic_space.hyperbolic_geodesic.HyperbolicGeodesicUHP(model, start, end, **graphics_options)

Bases: sage.geometry.hyperbolic_space.hyperbolic_geodesic.HyperbolicGeodesic

Create a geodesic in the upper half plane model.

The geodesics in this model are represented by circular arcs perpendicular to the real axis (half-circles whose origin is on the real axis) and straight vertical lines ending on the real axis.

INPUT:

• start – a HyperbolicPoint in hyperbolic space representing the start of the geodesic
• end – a HyperbolicPoint in hyperbolic space representing the end of the geodesic

EXAMPLES:

```
sage: UHP = HyperbolicPlane().UHP()
sage: g = UHP.get_geodesic(UHP.get_point(I), UHP.get_point(2 + I))
sage: g = UHP.get_geodesic(I, 2 + I)
sage: h = UHP.get_geodesic(-1, -1+2*I)
```
angle(\textit{other})

Return the angle between the completions of any two given geodesics if they intersect.

INPUT:

- \textit{other} – a hyperbolic geodesic in the UHP model

OUTPUT:

- the angle in radians between the two given geodesics

EXAMPLES:

```
sage: UHP = HyperbolicPlane().UHP()
sage: g = UHP.get_geodesic(2, 4)
sage: h = UHP.get_geodesic(3, 3 + I)
sage: g.angle(h)
1/2*pi
sage: numerical_approx(g.angle(h))
1.57079632679490
```

If the geodesics are identical, return angle 0:

```
sage: g.angle(g)
0
```

It is an error to ask for the angle of two geodesics that do not intersect:
```python
sage: g = UHP.get_geodesic(2, 4)
sage: h = UHP.get_geodesic(5, 7)
sage: g.angle(h)
Traceback (most recent call last):
  ...  
ValueError: geodesics do not intersect
```

The `common_perpendicular` method returns the unique hyperbolic geodesic perpendicular to `self` and `other`, if such a geodesic exists; otherwise, it raises a `ValueError`.

**INPUT:**
- `other` – a hyperbolic geodesic in the current model

**OUTPUT:**
- a hyperbolic geodesic

**EXAMPLES:**
```python
sage: UHP = HyperbolicPlane().UHP()
sage: g = UHP.get_geodesic(2, 3)
sage: h = UHP.get_geodesic(4, 5)
sage: g.common_perpendicular(h)
```

Geodesic in `UHP` from \(1/2\sqrt{3} + 7/2\) to \(-1/2\sqrt{3} + 7/2\)
It is an error to ask for the common perpendicular of two intersecting geodesics:

```
sage: g = UHP.get_geodesic(2, 4)
sage: h = UHP.get_geodesic(3, infinity)
sage: g.common_perpendicular(h)
Traceback (most recent call last):
...
ValueError: geodesics intersect; no common perpendicular exists
```

### ideal_endpoints()

Determine the ideal (boundary) endpoints of the complete hyperbolic geodesic corresponding to `self`.

**OUTPUT:**

- a list of 2 boundary points

**EXAMPLES:**

```
sage: UHP = HyperbolicPlane().UHP()
sage: UHP.get_geodesic(I, 2*I).ideal_endpoints()
[Boundary point in UHP 0, Boundary point in UHP +Infinity]
sage: UHP.get_geodesic(1 + I, 2 + 4*I).ideal_endpoints()
[Boundary point in UHP -sqrt(65) + 9, Boundary point in UHP sqrt(65) + 9]
```

### intersection(other)

Return the point of intersection of `self` and `other` (if such a point exists).

**INPUT:**

- `other` – a hyperbolic geodesic in the current model

**OUTPUT:**

- a list of hyperbolic points or a hyperbolic geodesic

**EXAMPLES:**

```
sage: UHP = HyperbolicPlane().UHP()
sage: g = UHP.get_geodesic(3, 5)
sage: h = UHP.get_geodesic(4, 7)
sage: g.intersection(h)
[Point in UHP 2/3*sqrt(-2) + 13/3]
```

If the given geodesics do not intersect, the function returns an empty list:

```
sage: g = UHP.get_geodesic(4, 5)
sage: h = UHP.get_geodesic(6, 7)
sage: g.intersection(h)
[]
```

If the given geodesics are asymptotically parallel, the function returns the common boundary point:

```
sage: g = UHP.get_geodesic(4, 5)
sage: h = UHP.get_geodesic(5, 7)
sage: g.intersection(h)
[Boundary point in UHP 5.000000000000000]
```
If the given geodesics are identical, return that geodesic:

```python
sage: g = UHP.get_geodesic(4 + I, 18*I)
sage: h = UHP.get_geodesic(4 + I, 18*I)
sage: g.intersection(h)
Geodesic in UHP from I + 4 to 18*I
```

**midpoint()**

Return the (hyperbolic) midpoint of `self` if it exists.

**EXAMPLES:**

```python
sage: UHP = HyperbolicPlane().UHP()
sage: g = UHP.random_geodesic()
sage: m = g.midpoint()
sage: d1 = UHP.dist(m, g.start())
sage: d2 = UHP.dist(m, g.end())
sage: bool(abs(d1 - d2) < 10**-9)
```

Infinite geodesics have no midpoint:

```python
sage: UHP.get_geodesic(0, 2).midpoint()
Traceback (most recent call last):
... ValueError: the length must be finite
```

**perpendicular_bisector()**

Return the perpendicular bisector of the hyperbolic geodesic `self` if that geodesic has finite length.

**EXAMPLES:**

```python
sage: UHP = HyperbolicPlane().UHP()
sage: g = UHP.random_geodesic()
sage: h = g.perpendicular_bisector().complete()
sage: c = lambda x: x.coordinates()
sage: bool(c(g.intersection(h)[0]) - c(g.midpoint()) < 10**-9)
```

Infinite geodesics cannot be bisected:

```python
sage: UHP.get_geodesic(0, 1).perpendicular_bisector()
Traceback (most recent call last):
... ValueError: the length must be finite
```

**plot**(boundary=True, **options)**

Plot `self`.

**EXAMPLES:**
sage: UHP = HyperbolicPlane().UHP()
sage: UHP.get_geodesic(0, 1).plot()  # optional - sage.plot
Graphics object consisting of 2 graphics primitives

sage: UHP.get_geodesic(I, 3+4*I).plot(linestyle="dashed", color="brown")  # optional - sage.plot
Graphics object consisting of 2 graphics primitives

sage: UHP.get_geodesic(1, infinity).plot(color='orange')  # optional - sage.plot
Graphics object consisting of 2 graphics primitives

reflection_involution()
Return the isometry of the involution fixing the geodesic self.

EXAMPLES:

sage: UHP = HyperbolicPlane().UHP()
sage: g1 = UHP.get_geodesic(0, 1)
sage: g1.reflection_involution()
Isometry in UHP
[ 1  0]
[ 2 -1]
sage: UHP.get_geodesic(I, 2*I).reflection_involution()
Isometry in UHP

\[
\begin{bmatrix}
1 & 0 \\
0 & -1
\end{bmatrix}
\]
In this module, a hyperbolic model is a collection of data that allow the user to implement new models of hyperbolic space with minimal effort. The data include facts about the underlying set (such as whether the model is bounded), facts about the metric (such as whether the model is conformal), facts about the isometry group (such as whether it is a linear or projective group), and more. Generally speaking, any data or method that pertains to the model itself – rather than the points, geodesics, or isometries of the model – is implemented in this module.

Abstractly, a model of hyperbolic space is a connected, simply connected manifold equipped with a complete Riemannian metric of constant curvature $-1$. This module records information sufficient to enable computations in hyperbolic space without explicitly specifying the underlying set or its Riemannian metric. Although, see the SageManifolds project if you would like to take this approach.

This module implements the abstract base class for a model of hyperbolic space of arbitrary dimension. It also contains the implementations of specific models of hyperbolic geometry.

AUTHORS:

EXAMPLES:
We illustrate how the classes in this module encode data by comparing the upper half plane (UHP), Poincaré disk (PD) and hyperboloid (HM) models. First we create:

```python
sage: U = HyperbolicPlane().UHP()
sage: P = HyperbolicPlane().PD()
sage: H = HyperbolicPlane().HM()
```

We note that the UHP and PD models are bounded while the HM model is not:

```python
sage: U.is_bounded() and P.is_bounded()
True
sage: H.is_bounded()
False
```

The isometry groups of UHP and PD are projective, while that of HM is linear:

```python
sage: U.is_isometry_group_projective()
True
sage: H.is_isometry_group_projective()
False
```

The models are responsible for determining if the coordinates of points and the matrix of linear maps are appropriate for constructing points and isometries in hyperbolic space:
```plaintext
sage: U.point_in_model(2 + I)
True
sage: U.point_in_model(2 - I)
False
sage: U.point_in_model(2)
False
sage: U.boundary_point_in_model(2)
True
```

```plaintext
class sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModel(
space, name,
short_name, bounded,
conformal, dimension,
isometry_group, isometry_group_is_projective)
```

Bases: `sage.structure.parent.Parent`, `sage.structure.unique_representation.UniqueRepresentation`, `sage.misc.bindable_classBindableClass`

Abstract base class for hyperbolic models.

**Element**
alias of `sage.geometry.hyperbolic_space.hyperbolic_point.HyperbolicPoint`

**bdry_point_test**(p)
Test whether a point is in the model. If the point is in the model, do nothing; otherwise raise a `ValueError`.

```
sage: HyperbolicPlane().UHP().bdry_point_test(2)
sage: HyperbolicPlane().UHP().bdry_point_test(1 + I)
Traceback (most recent call last):
... ValueError: I + 1 is not a valid boundary point in the UHP model
```

**boundary_point_in_model**(p)
Return `True` if the point is on the ideal boundary of hyperbolic space and `False` otherwise.

**INPUT:**
• any object that can converted into a complex number

**OUTPUT:**
• boolean

**EXAMPLES:**
```
sage: HyperbolicPlane().UHP().boundary_point_in_model(1)
False
```

**dist**(a, b)
Calculate the hyperbolic distance between a and b.

**INPUT:**
• a, b – a point or geodesic

**OUTPUT:**
• the hyperbolic distance

**EXAMPLES:**
sage: UHP = HyperbolicPlane().UHP()
sage: p1 = UHP.get_point(5 + 7*I)
sage: p2 = UHP.get_point(1.0 + I)
sage: UHP.dist(p1, p2)
2.23230104635820

sage: PD = HyperbolicPlane().PD()
sage: p1 = PD.get_point(0)
sage: p2 = PD.get_point(I/2)
sage: PD.dist(p1, p2)
arccosh(5/3)

sage: KM = HyperbolicPlane().KM()
sage: p1 = KM.get_point((0, 0))
sage: p2 = KM.get_point((1/2, 1/2))
sage: numerical_approx(KM.dist(p1, p2))
0.881373587019543

sage: HM = HyperbolicPlane().HM()
sage: p1 = HM.get_point((0,0,1))
sage: p2 = HM.get_point((1,0,sqrt(2)))
sage: numerical_approx(HM.dist(p1, p2))
0.881373587019543

Distance between a point and itself is 0:

sage: p = UHP.get_point(47 + I)
sage: UHP.dist(p, p)
0

Points on the boundary are infinitely far from interior points:

sage: UHP.get_point(3).dist(UHP.get_point(I))
+Infinity

**get_geodesic**(start, end=None, **graphics_options)
Return a geodesic in the appropriate model.

EXAMPLES:

sage: HyperbolicPlane().UHP().get_geodesic(I, 2*I)
Geodesic in UHP from I to 2*I

sage: HyperbolicPlane().PD().get_geodesic(0, I/2)
Geodesic in PD from 0 to 1/2*I

sage: HyperbolicPlane().KM().get_geodesic((1/2, 1/2), (0,0))
Geodesic in KM from (1/2, 1/2) to (0, 0)

sage: HyperbolicPlane().HM().get_geodesic((0,0,1), (1,0, sqrt(2)))
Geodesic in HM from (0, 0, 1) to (1, 0, sqrt(2))
**get_iso\_metry**\(\(A\)\)
Return an isometry in \(\text{self}\) from the matrix \(A\) in the isometry group of \(\text{self}\).

**EXAMPLES:**

```
sage: HyperbolicPlane().UHP().get_isometry(identity_matrix(2))
Isometry in UHP
[1 0]
[0 1]
sage: HyperbolicPlane().PD().get_isometry(identity_matrix(2))
Isometry in PD
[1 0]
[0 1]
sage: HyperbolicPlane().KM().get_isometry(identity_matrix(3))
Isometry in KM
[1 0 0]
[0 1 0]
[0 0 1]
sage: HyperbolicPlane().HM().get_isometry(identity_matrix(3))
Isometry in HM
[1 0 0]
[0 1 0]
[0 0 1]
```

**get_point**\((coordinates, is\_boundary=\text{None}, **graphics\_options})
Return a point in \(\text{self}\).

Automatically determine the type of point to return given either:

1. the coordinates of a point in the interior or ideal boundary of hyperbolic space, or
2. a \texttt{HyperbolicPoint} object.

**INPUT:**

- a point in hyperbolic space or on the ideal boundary

**OUTPUT:**

- a \texttt{HyperbolicPoint}

**EXAMPLES:**

We can create an interior point via the coordinates:

```
sage: HyperbolicPlane().UHP().get_point(2*I)
Point in UHP 2*I
```

Or we can create a boundary point via the coordinates:

```
sage: HyperbolicPlane().UHP().get_point(23)
Boundary point in UHP 23
```

However we cannot create points outside of our model:
sage: HyperbolicPlane().UHP().get_point(12 - I)
Traceback (most recent call last):
  ...  
ValueError: -I + 12 is not a valid point in the UHP model

sage: HyperbolicPlane().UHP().get_point(2 + 3*I)
Point in UHP 3*I + 2

sage: HyperbolicPlane().PD().get_point(0)
Point in PD 0

sage: HyperbolicPlane().KM().get_point((0,0))
Point in KM (0, 0)

sage: HyperbolicPlane().HM().get_point((0,0,1))
Point in HM (0, 0, 1)

sage: p = HyperbolicPlane().UHP().get_point(I, color="red")
sage: p.graphics_options()
{'color': 'red'}

sage: HyperbolicPlane().UHP().get_point(12)
Boundary point in UHP 12

sage: HyperbolicPlane().UHP().get_point(infinity)
Boundary point in UHP +Infinity

sage: HyperbolicPlane().PD().get_point(I)
Boundary point in PD I

sage: HyperbolicPlane().KM().get_point((0,-1))
Boundary point in KM (0, -1)

is_bounded()
Return True if self is a bounded model.

EXAMPLES:

sage: HyperbolicPlane().UHP().is_bounded()
True
sage: HyperbolicPlane().PD().is_bounded()
True
sage: HyperbolicPlane().KM().is_bounded()
True
sage: HyperbolicPlane().HM().is_bounded()
False

is_conformal()
Return True if self is a conformal model.

EXAMPLES:

sage: UHP = HyperbolicPlane().UHP()
is_conformal()  
Return True if the isometry group of self is conformal.

EXAMPLES:

```
sage: UHP = HyperbolicPlane().UHP()
sage: UHP.is_conformal()
True
```

is_isometry_group_projective()  
Return True if the isometry group of self is projective.

EXAMPLES:

```
sage: UHP = HyperbolicPlane().UHP()
sage: UHP.is_isometry_group_projective()
True
```

isometry_from_fixed_points(repel, attract)  
Given two fixed points repel and attract as hyperbolic points return a hyperbolic isometry with repel as repelling fixed point and attract as attracting fixed point.

EXAMPLES:

```
sage: UHP = HyperbolicPlane().UHP()
sage: PD = HyperbolicPlane().PD()
sage: PD.isometry_from_fixed_points(-i, i)
Isometry in PD
[ 3/4  1/4*I]
[-1/4*I 3/4]
sage: p, q = PD.get_point(1/2 + I/2), PD.get_point(6/13 + 9/13*I)
sage: PD.isometry_from_fixed_points(p, q)
Traceback (most recent call last):
...  
ValueError: fixed points of hyperbolic elements must be ideal
sage: p, q = PD.get_point(4/5 + 3/5*I), PD.get_point(-I)
sage: PD.isometry_from_fixed_points(p, q)
Isometry in PD
[ 1/6*I - 2/3 -1/3*I - 1/6]
[ 1/3*I - 1/6 -1/6*I - 2/3]
```

isometry_in_model(A)  
Return True if the input matrix represents an isometry of the given model and False otherwise.

INPUT:
- a matrix that represents an isometry in the appropriate model

OUTPUT:
- boolean

EXAMPLES:

```
sage: HyperbolicPlane().UHP().isometry_in_model(identity_matrix(2))
True
sage: HyperbolicPlane().UHP().isometry_in_model(identity_matrix(3))
False
```
isometry_test(A)
Test whether an isometry A is in the model.
If the isometry is in the model, do nothing. Otherwise, raise a ValueError.

EXAMPLES:

```
sage: HyperbolicPlane().UHP().isometry_test(identity_matrix(2))
sage: HyperbolicPlane().UHP().isometry_test(matrix(2, [I,1,2,1]))
Traceback (most recent call last):
...  
ValueError:
[I 1]
[2 1] is not a valid isometry in the UHP model
```

name()
Return the name of this model.

EXAMPLES:

```
sage: UHP = HyperbolicPlane().UHP()
sage: UHP.name()
'Upper Half Plane Model'
```

point_in_model(p)
Return True if the point p is in the interior of the given model and False otherwise.

INPUT:
• any object that can converted into a complex number

OUTPUT:
• boolean

EXAMPLES:

```
sage: HyperbolicPlane().UHP().point_in_model(I)
True
sage: HyperbolicPlane().UHP().point_in_model(-I)
False
```

point_test(p)
Test whether a point is in the model. If the point is in the model, do nothing. Otherwise, raise a ValueError.

EXAMPLES:

```
sage: from sage.geometry.hyperbolic_space.hyperbolic_model import...
HyperbolicModelUHP
sage: HyperbolicPlane().UHP().point_test(2 + I)
sage: HyperbolicPlane().UHP().point_test(2 - I)
Traceback (most recent call last):
...  
ValueError: -I + 2 is not a valid point in the UHP model
```

random_element(**kwargs)
Return a random point in self.
The points are uniformly distributed over the rectangle $[-10, 10] \times [0, 10i]$ in the upper half plane model.
EXAMPLES:

```python
sage: p = HyperbolicPlane().UHP().random_element()
sage: bool((p.coordinates().imag()) > 0)
True

sage: p = HyperbolicPlane().PD().random_element()
sage: HyperbolicPlane().PD().point_in_model(p.coordinates())
True

sage: p = HyperbolicPlane().KM().random_element()
sage: HyperbolicPlane().KM().point_in_model(p.coordinates())
True

sage: p = HyperbolicPlane().HM().random_element().coordinates()
sage: bool((p[0]**2 + p[1]**2 - p[2]**2 - 1) < 10**-8)
True
```

random_geodesic(**kwargs)

Return a random hyperbolic geodesic.

Return the geodesic between two random points.

EXAMPLES:

```python
sage: h = HyperbolicPlane().PD().random_geodesic()
sage: bool((h.endpoints()[0].coordinates()).imag() >= 0)
True
```

random_isometry(preserve_orientation=True, **kwargs)

Return a random isometry in the model of self.

INPUT:

• `preserve_orientation` – if True return an orientation-preserving isometry

OUTPUT:

• a hyperbolic isometry

EXAMPLES:

```python
sage: A = HyperbolicPlane().PD().random_isometry()
sage: A.preserves_orientation()
True
sage: B = HyperbolicPlane().PD().random_isometry(preserve_orientation=False)
sage: B.preserves_orientation()
False
```

random_point(**kwargs)

Return a random point of self.

The points are uniformly distributed over the rectangle $[-10, 10] \times [0, 10i]$ in the upper half plane model.

EXAMPLES:

```python
sage: p = HyperbolicPlane().UHP().random_point()
sage: bool((p.coordinates().imag()) > 0)
```

(continues on next page)
True

```sage
sage: PD = HyperbolicPlane().PD()
sage: p = PD.random_point()
sage: PD.point_in_model(p.coordinates())
```

**short_name()**

Return the short name of this model.

EXAMPLES:

```sage
sage: UHP = HyperbolicPlane().UHP()
sage: UHP.short_name()
'UHP'
```

### class `sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModelHM(space)`

**Bases:** `sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModel`  
Hyperboloid Model.

**boundary_point_in_model(p)**

Return False since the Hyperboloid model has no boundary points.

EXAMPLES:

```sage
sage: HM = HyperbolicPlane().HM()
sage: HM.boundary_point_in_model((0,0,1))
False
sage: HM.boundary_point_in_model((1,0,sqrt(2)))
False
sage: HM.boundary_point_in_model((1,2,1))
False
```

**get_background_graphic(**bdry_options**)**

Return a graphic object that makes the model easier to visualize. For the hyperboloid model, the background object is the hyperboloid itself.

EXAMPLES:

```sage
sage: H = HyperbolicPlane().HM().get_background_graphic()
```

**isometry_in_model(A)**

Test that the matrix A is in the group $SO(2,1)^+$.  

EXAMPLES:

```sage
sage: A = diagonal_matrix([1,1,-1])
sage: HyperbolicPlane().HM().isometry_in_model(A)
```

**point_in_model(p)**

Check whether a complex number lies in the hyperboloid.

EXAMPLES:
class `sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModelKM(space)`

Bases: `sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModel`

Klein Model.

**boundary_point_in_model(p)**
Check whether a point lies in the unit circle, which corresponds to the ideal boundary of the hyperbolic plane in the Klein model.

**EXAMPLES:**

```python
sage: KM = HyperbolicPlane().KM()
sage: KM.boundary_point_in_model((1, 0))
    True
sage: KM.boundary_point_in_model((1/2, 1/2))
    False
sage: KM.boundary_point_in_model((1, .2))
    False
```

**get_background_graphic(**`**bdry_options`**)**

Return a graphic object that makes the model easier to visualize.

For the Klein model, the background object is the ideal boundary.

**EXAMPLES:**

```python
sage: circ = HyperbolicPlane().KM().get_background_graphic()
```

**isometry_in_model(A)**
Check if the given matrix $A$ is in the group $SO(2, 1)$.

**EXAMPLES:**

```python
sage: A = matrix(3, [[1, 0, 0], [0, 17/8, 15/8], [0, 15/8, 17/8]])
sage: HyperbolicPlane().KM().isometry_in_model(A)
    True
```

**point_in_model(p)**
Check whether a point lies in the open unit disk.

**EXAMPLES:**

```python
sage: KM = HyperbolicPlane().KM()
sage: KM.point_in_model((1, 0))
    False
sage: KM.point_in_model((1/2, 1/2))
    True
sage: KM.point_in_model((1, .2))
    False
```
class sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModelPD(space)
Bases: sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModel

Poincaré Disk Model.

boundary_point_in_model(p)
Check whether a complex number lies in the open unit disk.

EXAMPLES:

```python
sage: PD = HyperbolicPlane().PD()
sage: PD.boundary_point_in_model(1.00)
True
sage: PD.boundary_point_in_model(1/2 + I/2)
False
sage: PD.boundary_point_in_model(1 + .2*I)
False
```

get_background_graphic(**bdry_options)
Return a graphic object that makes the model easier to visualize.

For the Poincaré disk, the background object is the ideal boundary.

EXAMPLES:

```python
sage: circ = HyperbolicPlane().PD().get_background_graphic()
```

isometry_in_model(A)
Check if the given matrix A is in the group $U(1, 1)$.

EXAMPLES:

```python
sage: z = [CC.random_element() for k in range(2)]; z.sort(key=abs)
sage: A = matrix(2,[z[1], z[0],z[0].conjugate(),z[1].conjugate()])
sage: HyperbolicPlane().PD().isometry_in_model(A)
True
```

point_in_model(p)
Check whether a complex number lies in the open unit disk.

EXAMPLES:

```python
sage: PD = HyperbolicPlane().PD()
sage: PD.point_in_model(1.00)
False
sage: PD.point_in_model(1/2 + I/2)
True
sage: PD.point_in_model(1 + .2*I)
False
```

class sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModelUHP(space)
Bases: sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModel

Upper Half Plane model.

Element
alias of sage.geometry.hyperbolic_space.hyperbolic_point.HyperbolicPointUHP
boundary_point_in_model($p$)
Check whether a complex number is a real number or $\infty$. In the UHP.model_name_name, this is the ideal boundary of hyperbolic space.

EXAMPLES:

```python
sage: UHP = HyperbolicPlane().UHP()
sage: UHP.boundary_point_in_model(1 + I)
False
sage: UHP.boundary_point_in_model(infinity)
True
sage: UHP.boundary_point_in_model(CC(infinity))
True
sage: UHP.boundary_point_in_model(RR(infinity))
True
sage: UHP.boundary_point_in_model(1)
True
sage: UHP.boundary_point_in_model(12)
True
sage: UHP.boundary_point_in_model(1 - I)
False
sage: UHP.boundary_point_in_model(-2*I)
False
sage: UHP.boundary_point_in_model(0)
True
sage: UHP.boundary_point_in_model(I)
False
```

get_background_graphic(**bdry_options)
Return a graphic object that makes the model easier to visualize. For the upper half space, the background object is the ideal boundary.

EXAMPLES:

```python
sage: hp = HyperbolicPlane().UHP().get_background_graphic()
```

isometry_from_fixed_points(repel, attract)
Given two fixed points repel and attract as complex numbers return a hyperbolic isometry with repel as repelling fixed point and attract as attracting fixed point.

EXAMPLES:

```python
sage: UHP = HyperbolicPlane().UHP()
sage: UHP.isometry_from_fixed_points(2 + I, 3 + I)
Traceback (most recent call last):
... ValueError: fixed points of hyperbolic elements must be ideal
sage: UHP.isometry_from_fixed_points(2, 0)
Isometry in UHP
[[ -1  0]
 [ -1/3 -1/3]]
```

isometry_in_model($A$)
Check that $A$ acts as an isometry on the upper half plane. That is, $A$ must be an invertible $2 \times 2$ matrix with real entries.
EXAMPLES:

```
sage: UHP = HyperbolicPlane().UHP()
sage: A = matrix(2,[1,2,3,4])
sage: UHP.isometry_in_model(A)
True
sage: B = matrix(2,[1,2,4,1])
sage: UHP.isometry_in_model(B)
False
```

An example of a matrix $A$ such that $\det(A) \neq 1$, but the $A$ acts isometrically:

```
sage: C = matrix(2,[10,0,0,10])
sage: UHP.isometry_in_model(C)
True
```

`point_in_model(p)`
Check whether a complex number lies in the open upper half plane.

EXAMPLES:

```
sage: UHP = HyperbolicPlane().UHP()
sage: UHP.point_in_model(1 + I)
True
sage: UHP.point_in_model(infinity)
False
sage: UHP.point_in_model(CC(infinity))
False
sage: UHP.point_in_model(RR(infinity))
False
sage: UHP.point_in_model(1)
False
sage: UHP.point_in_model(12)
False
sage: UHP.point_in_model(1 - I)
False
sage: UHP.point_in_model(-2*I)
False
sage: UHP.point_in_model(I)
True
sage: UHP.point_in_model(0) # Not interior point
False
```

`random_isometry(preserve_orientation=True, **kwargs)`
Return a random isometry in the Upper Half Plane model.

INPUT:

- `preserve_orientation` – if `True` return an orientation-preserving isometry

OUTPUT:

- a hyperbolic isometry

EXAMPLES:
random_point(**kwargs)
Return a random point in the upper half plane. The points are uniformly distributed over the rectangle $[-10, 10] \times [0, 10i]$.

EXAMPLES:

```python
sage: p = HyperbolicPlane().UHP().random_point().coordinates()
sage: bool((p.imag()) > 0)
True
```
CHAPTER
FIVE

INTERFACE TO HYPERBOLIC MODELS

This module provides a convenient interface for interacting with models of hyperbolic space as well as their points, geodesics, and isometries.

The primary point of this module is to allow the code that implements hyperbolic space to be sufficiently decoupled while still providing a convenient user experience.

The interfaces are by default given abbreviated names. For example, UHP (upper half plane model), PD (Poincaré disk model), KM (Klein disk model), and HM (hyperboloid model).

Note: All of the current models of 2 dimensional hyperbolic space use the upper half plane model for their computations. This can lead to some problems, such as long coordinate strings for symbolic points. For example, the vector (1, 0, sqrt(2)) defines a point in the hyperboloid model. Performing mapping this point to the upper half plane and performing computations there may return with vector whose components are unsimplified strings have several sqrt(2)'s. Presently, this drawback is outweighed by the rapidity with which new models can be implemented.

AUTHORS:

EXAMPLES:

```sage
sage: HyperbolicPlane().UHP().get_point(2 + I)
Point in UHP I + 2
sage: HyperbolicPlane().PD().get_point(1/2 + I/2)
Point in PD 1/2*I + 1/2
```

class sage.geometry.hyperbolic_space.hyperbolic_interface.HyperbolicModels(base)

    Bases: sage.categories.realizations.Category_realization_of_parent

The category of hyperbolic models of hyperbolic space.

class ParentMethods

    Bases: object

    super_categories()

        The super categories of self.

    EXAMPLES:
from sage.geometry.hyperbolic_space.hyperbolic_interface import HyperbolicModels
H = HyperbolicPlane()
models = HyperbolicModels(H)
models.super_categories()

Class \code{sage.geometry.hyperbolic_space.hyperbolic_interface.HyperbolicPlane}

Bases: \code{sage.structure.parent.Parent}, \code{sage.structure.unique_representation.UniqueRepresentation}

The hyperbolic plane \( \mathbb{H}^2 \).

Here are the models currently implemented:

- \( \text{UHP} \) – upper half plane
- \( \text{PD} \) – Poincaré disk
- \( \text{KM} \) – Klein disk
- \( \text{HM} \) – hyperboloid model

- \( \text{HM} \) alias of \code{sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModelHM}
- \( \text{Hyperboloid} \) alias of \code{sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModelHM}
- \( \text{KM} \) alias of \code{sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModelKM}
- \( \text{KleinDisk} \) alias of \code{sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModelKM}
- \( \text{PD} \) alias of \code{sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModelPD}
- \( \text{PoincareDisk} \) alias of \code{sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModelPD}
- \( \text{UHP} \) alias of \code{sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModelUHP}
- \( \text{UpperHalfPlane} \) alias of \code{sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModelUHP}

a\_realization()

Return a realization of self.

EXAMPLES:

\begin{verbatim}
sage: H = HyperbolicPlane()
sage: H.a_realization()
\end{verbatim}

Hyperbolic plane in the Upper Half Plane Model

\code{sage.geometry.hyperbolic_space.hyperbolic_interface.HyperbolicSpace(n)}

Return \( n \) dimensional hyperbolic space.

EXAMPLES:
sage: from sage.geometry.hyperbolic_space.hyperbolic_interface import*
˓→HyperbolicSpace
sage: HyperbolicSpace(2)
Hyperbolic plane
CHAPTER
SIX

INDICES AND TABLES

- Index
- Module Index
- Search Page
sage.geometry.hyperbolic_space.hyperbolic_geodesic, 17
sage.geometry.hyperbolic_space.hyperbolic_interface, 77
sage.geometry.hyperbolic_space.hyperbolic_isometry, 7
sage.geometry.hyperbolic_space.hyperbolic_model, 63
sage.geometry.hyperbolic_space.hyperbolic_point, 1
INDEX

A
a_realization() (sage.geometry.hyperbolic_space.hyperbolic_interface.HyperbolicPlane method), 78
angle() (sage.geometry.hyperbolic_space.hyperbolic_geodesic.HyperbolicGeodesic method), 21
angle() (sage.geometry.hyperbolic_space.hyperbolic_geodesic.HyperbolicGeodesicUHP method), 49
attracting_fixed_point() (sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometry method), 7
attracting_fixed_point() (sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometryUHP method), 13
axis() (sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometry method), 8

B
bdry_point_test() (sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModel method), 64
boundary_point_in_model() (sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModel method), 64
boundary_point_in_model() (sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModelHM method), 71
boundary_point_in_model() (sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModelKM method), 72
boundary_point_in_model() (sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModelPD method), 73
boundary_point_in_model() (sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModelUHP method), 73

C
classification() (sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometry method), 8
classification() (sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometryUHP method), 13
common_perpendicular() (sage.geometry.hyperbolic_space.hyperbolic_geodesic.HyperbolicGeodesic method), 21
common_perpendicular() (sage.geometry.hyperbolic_space.hyperbolic_geodesic.HyperbolicGeodesicUHP method), 52
complete() (sage.geometry.hyperbolic_space.hyperbolic_geodesic.HyperbolicGeodesic method), 23
coordinates() (sage.geometry.hyperbolic_space.hyperbolic_point.HyperbolicPoint method), 3
dist() (sage.geometry.hyperbolic_space.hyperbolic_geodesic.HyperbolicGeodesic method), 24
dist() (sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModel method), 64

D
element (sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModel attribute), 64
element (sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModelUHP attribute), 73
dist() (sage.geometry.hyperbolic_space.hyperbolic_geodesic.HyperbolicGeodesicUHP method), 52
endpoints() (sage.geometry.hyperbolic_space.hyperbolic_geodesic.HyperbolicGeodesicUHP method), 28

E
element (sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModel attribute), 64
element (sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModelUHP attribute), 73

F
element (sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModel attribute), 64
element (sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModelUHP attribute), 73
fixed_point_set() (sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometry method), 8
fixed_point_set() (sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometryUHP method), 14

G
get_background_graphic() (sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModel method), 72
get_background_graphic() (sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModelUHP method), 72
get_background_graphic() (sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModelPD method), 73

get_background_graphic() (sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModelUHP method), 74

get_geodesic() (sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModel method), 65

get_isometry() (sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModel method), 65

get_point() (sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModel method), 66

graphics_options() (sage.geometry.hyperbolic_space.hyperbolic_geodesic.HyperbolicGeodesic method), 28

graphics_options() (sage.geometry.hyperbolic_space.hyperbolic_point.HyperbolicPoint method), 3

H (sage.geometry.hyperbolic_space.hyperbolic_interface.HyperbolicPlane attribute), 78

HyperbolicGeodesic (class in sage.geometry.hyperbolic_space.hyperbolic_geodesic), 21

HyperbolicGeodesicHM (class in sage.geometry.hyperbolic_space.hyperbolic_geodesic), 41

HyperbolicGeodesicKM (class in sage.geometry.hyperbolic_space.hyperbolic_geodesic), 42

HyperbolicGeodesicPD (class in sage.geometry.hyperbolic_space.hyperbolic_geodesic), 44

HyperbolicGeodesicUHP (class in sage.geometry.hyperbolic_space.hyperbolic_geodesic), 49

HyperbolicIsometry (class in sage.geometry.hyperbolic_space.hyperbolic_isometry), 7

HyperbolicIsometryKM (class in sage.geometry.hyperbolic_space.hyperbolic_isometry), 12

HyperbolicIsometryPD (class in sage.geometry.hyperbolic_space.hyperbolic_isometry), 12

HyperbolicIsometryUHP (class in sage.geometry.hyperbolic_space.hyperbolic_isometry), 13

HyperbolicModel (class in sage.geometry.hyperbolic_space.hyperbolic_model), 64

HyperbolicModelHM (class in sage.geometry.hyperbolic_space.hyperbolic_model), 71

HyperbolicModelKM (class in sage.geometry.hyperbolic_space.hyperbolic_model), 72

HyperbolicModelPD (class in sage.geometry.hyperbolic_space.hyperbolic_model), 72

HyperbolicModelUHP (class in sage.geometry.hyperbolic_space.hyperbolic_model), 73

Hyperboloid (class in sage.geometry.hyperbolic_space.hyperbolic_interface.HyperbolicPlane attribute), 78

ideal_endpoints() (sage.geometry.hyperbolic_space.hyperbolic_geodesic.HyperbolicGeodesic method), 28

ideal_endpoints() (sage.geometry.hyperbolic_space.hyperbolic_geodesic.HyperbolicGeodesicUHP method), 53

intersection() (sage.geometry.hyperbolic_space.hyperbolic_geodesic.HyperbolicGeodesic method), 29

intersection() (sage.geometry.hyperbolic_space.hyperbolic_geodesic.HyperbolicGeodesicUHP method), 53

inverse() (sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometry method), 9

is_asymptotically_parallel() (sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometry method), 9

is_boundary() (sage.geometry.hyperbolic_space.hyperbolic_point.HyperbolicPoint method), 29

is_bounded() (sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModel method), 67

is_complete() (sage.geometry.hyperbolic_space.hyperbolic_geodesic.HyperbolicGeodesic method), 32

is_conformal() (sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModel method), 67

is_identity() (sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometry method), 9

is_isometry_group_projective()
preserves_orientation() (sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometryPD method), 12
preserves_orientation() (sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometryUHP method), 14

random_element() (sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometryPD method), 69
random_geodesic() (sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometryPD method), 70
random_isometry() (sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometryPD method), 70
random_isometry() (sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometryUHP method), 75
random_point() (sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometryPD method), 70
random_point() (sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometryUHP method), 76
reflection_involution() (sage.geometry.hyperbolic_space.hyperbolic_geodesic.HyperbolicGeodesic method), 38
reflection_involution() (sage.geometry.hyperbolic_space.hyperbolic_point.HyperbolicPoint method), 4
reflection_involution() (sage.geometry.hyperbolic_space.hyperbolic_point.HyperbolicPointUHP method), 6
repelling_fixed_point() (sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometry method), 10
repelling_fixed_point() (sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometryUHP method), 14
translation_length() (sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometry method), 11
translation_length() (sage.geometry.hyperbolic_space.hyperbolic_isometry.HyperbolicIsometryUHP method), 15

short_name() (sage.geometry.hyperbolic_space.hyperbolic_model.HyperbolicModel method), 71
show() (sage.geometry.hyperbolic_space.hyperbolic_point.HyperbolicPoint method), 4
show() (sage.geometry.hyperbolic_space.hyperbolic_point.HyperbolicPointUHP method), 6
start() (sage.geometry.hyperbolic_space.hyperbolic_geodesic.HyperbolicGeodesic method), 40
super_categories() (sage.geometry.hyperbolic_space.hyperbolic_interf race.HyperbolicModels method), 77