Sébastien Labbé Research Code Reference Manual

Release 0.2

Sébastien Labbé

November 25, 2015

This is the reference manual for the Sébastien Labbé Research Code extension to the Sage mathematical software system. Sage is free open source math software that supports research and teaching in algebra, geometry, number theory, cryptography, and related areas. Sébastien Labbé Research Code implements digital geometry, combinatorics on words and symbolic dynamical systems simulation code in Sage, via a set of new Python classes. Many of the modules corresponds to research code written for published articles (double square tiles, Christoffel graphs, factor complexity). It is meant to be reused and reusable (full documentation including doctests). Comments are welcome.

To use this module, you need to import it:

```
from slabbe import *
```

This reference manual contains many examples that illustrate the usage of slabbe spkg. The examples are all tested with each release of slabbe spkg, and should produce exactly the same output as in this manual, except for line breaks.

CHAPTER

ONE

DIGITAL GEOMETRY

1.1 Discrete Subset

Subsets of ZZ^d with the edge relation +e_i and -e_i.

EXAMPLES:

```
sage: from slabbe import DiscreteSubset
sage: DiscreteSubset(2)
Subset of ZZ^2
sage: DiscreteSubset(4)
Subset of ZZ^4
```

A discrete 2d disk:

```
sage: D = DiscreteSubset(2, lambda (x,y) : x^2 + y^2 < 4)
sage: D.list()
[(0, 0), (0, 1), (0, -1), (1, 0), (-1, 0), (-1, 1), (1, -1), (1, 1), (-1, -1)]
sage: D
Subset of ZZ^2
```

A discrete 3d ball:

```
sage: predicate = lambda (x,y,z) : x^2 + y^2 + z^2 <= 4
sage: D = DiscreteSubset(3, predicate)
sage: D
Subset of ZZ^3
sage: (0,0,0) in D
True
sage: (10,10,10) in D
False
sage: len(D.list())
33
sage: D.plot()    # optional long
```

A discrete 4d hyperplane:

```
sage: predicate = lambda (x,y,z,w) : 0 <= 2*x + 3*y + 4*z + 5*w < 14
sage: D = DiscreteSubset(4, predicate)
sage: D
Subset of ZZ^4
sage: D.an_element()
(0, 0, 0, 0)
```

A 2d discrete box:

```
sage: from slabbe import DiscreteBox
sage: b = DiscreteBox([-5,5], [-5,5])
```

```
sage: b
Box: [-5, 5] x [-5, 5]
sage: b.plot()       # optional long
```

A 3d discrete box:

```
sage: b = DiscreteBox([-2,2], [-5,5], [-5,5])
sage: b
Box: [-2, 2] x [-5, 5] x [-5, 5]
sage: b.plot()       # optional long
```

The intersection of two discrete objects of the same dimension:

```
sage: circ = DiscreteSubset(2, lambda p: p[0]^2+p[1]^2<=100)
sage: b = DiscreteBox([0,10], [0,10])
sage: I = circ & b
sage: I
Intersection of the following objects:
Subset of ZZ^2
[0, 10] x [0, 10]
sage: I.an_element()
(0, 0)
sage: I.plot()      # optional long
```

A discrete tube (preimage of a discrete box by a matrix):

```
sage: M3to2 = matrix(2,[-sqrt(3),sqrt(3),0,-1,-1,2],ring=RR)/2
sage: M3to2
[-0.866025403784439  0.866025403784439  0.000000000000000]
[-0.500000000000000 -0.500000000000000   1.00000000000000]
sage: from slabbe import DiscreteTube
sage: tube = DiscreteTube([-5,5],[-5,5], projmat=M3to2)
sage: tube
DiscreteTube: Preimage of [-5, 5] x [-5, 5] by a 2 by 3 matrix
sage: it = iter(tube)
sage: [next(it) for _ in range(4)]
[(0, 0, 0), (1, 0, 0), (0, 0, 1), (0, 0, -1)]
```

TODO:

- Code Complement
- The method projection_matrix should be outside of the class?
- DiscreteTube should have a method projection_matrix
- The user should be able to provide an element to the object or a list of element
- Their should be an input saying whether the object is connected or not and what kind of neighbor connectedness
- When zero is not in self, then the an_element method fails (see below)

```
sage: D = DiscreteSubset(2, lambda (x,y) : 4 < x^2 + y^2 < 25)
sage: D.an_element()
Traceback (most recent call last):
...
AssertionError: an_element method returns an element which is not in self
```

class slabbe.discrete_subset.**DiscreteBox**(**args*)

Bases: slabbe.discrete_subset.DiscreteSubset

Cartesian product of intervals.

INPUT:

- *args - intervals, lists of size two : [min, max]

EXAMPLES:

```
sage: from slabbe import DiscreteBox
sage: DiscreteBox([-5,5],[-5,5])
Box: [-5, 5] x [-5, 5]

sage: D = DiscreteBox([-3,3],[-3,3],[-3,3],[-3,3])
sage: next(iter(D))
(0, 0, 0, 0)
```

TESTS:

```
sage: d = DiscreteBox([-5,5], [-5,5], [-4,4])
sage: d.edges_iterator().next()
((0, 0, 0), (1, 0, 0))
```

clip(*space=1*)

Return a good clip rectangle for this box.

INPUT:

- space – number (default: 1), inner space within the box

EXAMPLES:

```
sage: from slabbe import DiscreteBox
sage: box = DiscreteBox([-6,6],[-6,6])
sage: box
Box: [-6, 6] x [-6, 6]
sage: box.clip()
[(-5, -5), (5, -5), (5, 5), (-5, 5), (-5, -5)]

sage: box = DiscreteBox([-6,6],[-4,3])
sage: box.clip()
[(-5, -3), (5, -3), (5, 2), (-5, 2), (-5, -3)]
```

class slabbe.discrete_subset.**DiscreteSubset**(*dimension=3*, *predicate=None*, *edge_predicate=None*)

Bases: sage.structure.sage_object.SageObject

A subset of ZZ^d.

INPUT:

- dimension - integer, dimension of the space
- predicate - function ZZ^d -> {False, True}
- edge_predicate - function ZZ^d,{-1,0,1}^d -> {False, True}

EXAMPLES:

```
sage: from slabbe import DiscreteSubset
sage: DiscreteSubset(3)
Subset of ZZ^3

sage: p = DiscreteSubset(3, lambda x:True)
sage: p
Subset of ZZ^3

sage: fn = lambda p : p[0]+p[1]<p[2]
sage: p = DiscreteSubset(3, fn)
sage: p
Subset of ZZ^3
```

```
sage: F = lambda p: Integers(7)(2*p[0]+5*p[1])
sage: edge_predicate = lambda p,s: F(s) < F(s)
sage: D = DiscreteSubset(3, edge_predicate=edge_predicate)
sage: D
Subset of ZZ^3
```

TESTS:

No edges go outside of the box:

```
sage: from slabbe import DiscreteBox
sage: B = DiscreteBox([-1,1],[-1,1])
sage: len(list(B.edges_iterator()))
12
sage: sorted(B.edges_iterator())
[((-1, -1), (-1, 0)), ((-1, -1), (0, -1)), ((-1, 0), (-1, 1)),
((-1, 0), (0, 0)), ((-1, 1), (0, 1)), ((0, -1), (0, 0)), ((0, -1),
(1, -1)), ((0, 0), (0, 1)), ((0, 0), (1, 0)), ((0, 1), (1, 1)),
((1, -1), (1, 0)), ((1, 0), (1, 1))]
```

an_element()

Returns an immutable element in self.

EXAMPLES:

```
sage: from slabbe import DiscreteSubset
sage: p = DiscreteSubset(3)
sage: p.an_element()
(0, 0, 0)
sage: p.an_element().is_immutable()
True
```

base_edges()

Return a list of positive canonical vectors.

EXAMPLES:

```
sage: from slabbe import DiscreteSubset
sage: d = DiscreteSubset(2)
sage: d.base_edges()
[(1, 0), (0, 1)]
```

```
sage: from slabbe import DiscretePlane
sage: P = DiscretePlane([3,4,5], 12)
sage: P.base_edges()
[(1, 0, 0), (0, 1, 0), (0, 0, 1)]
```

children(*p*)

EXAMPLES:

```
sage: from slabbe import DiscretePlane
sage: p = DiscretePlane([1,pi,7], 1+pi+7, mu=0)
sage: list(p.children(vector((0,0,0))))
[(1, 0, 0), (0, 1, 0), (0, 0, 1)]
```

connected_component_iterator(*roots=None*)

Return an iterator over the connected component of the root.

INPUT:

- `roots` - list of some elements immutable in self

EXAMPLES:

```
sage: from slabbe import DiscreteSubset
sage: p = DiscreteSubset(3)
sage: root = vector((0,0,0))
sage: root.set_immutable()
sage: it = p.connected_component_iterator(roots=[root])
sage: [next(it) for _ in range(5)]
[(0, 0, 0), (1, 0, 0), (0, 0, 1), (0, 0, -1), (0, -1, 0)]

sage: from slabbe import DiscretePlane
sage: p = DiscretePlane([1,pi,7], 1+pi+7, mu=0)
sage: root = vector((0,0,0))
sage: root.set_immutable()
sage: it = p.connected_component_iterator(roots=[root])
sage: [next(it) for _ in range(5)]
[(0, 0, 0), (1, 0, 0), (0, 1, 0), (0, 0, 1), (-1, 1, 0)]
```

d_neighbors(*p*, *d=2*)

Retourne le voisinage du point p, i.e. les points parmi les 3^d possible qui appartiennent a l'objet discret.

INPUT:

- p - un point discret
- d - integer (optional, default:2),

OUTPUT:

liste de points

EXAMPLES:

```
sage: from slabbe import DiscretePlane
sage: p = DiscretePlane([1,3,7], 10)
sage: p.d_neighbors((0,0,0))
[(-1, -1, 1), (-1, 0, 1), (-1, 1, 0), (-1, 1, 1), (0, -1, 1),
(0, 0, 0), (0, 0, 1), (0, 1, 0), (1, -1, 1), (1, 0, 0), (1, 0,
1), (1, 1, 0)]
```

dimension()

Returns the dimension of the ambiant space.

OUTPUT:

integer

EXAMPLES:

```
sage: from slabbe import DiscreteSubset
sage: d = DiscreteSubset(3)
sage: d.dimension()
3

sage: from slabbe import DiscreteBox
sage: p = DiscreteBox([0,3], [0,3], [0,3], [0,3])
sage: p.dimension()
4
```

edges_iterator(*roots=None*)

Returns an iterator over the pair of points in self that are adjacents, i.e. their difference is a canonical vector.

It considers only points that are connected to the given roots.

INPUT:

- roots - list of some elements in self

EXAMPLES:

```
sage: from slabbe import DiscretePlane
sage: p = DiscretePlane([1,pi,7], 1+pi+7, mu=0)
sage: root = vector((0,0,0))
sage: root.set_immutable()
sage: it = p.edges_iterator(roots=[root])
sage: next(it)
((0, 0, 0), (1, 0, 0))
sage: next(it)
((0, 0, 0), (0, 1, 0))
sage: next(it)
((0, 0, 0), (0, 0, 1))
sage: next(it)
((-1, 1, 0), (0, 1, 0))
sage: next(it)
((-2, 1, 0), (-1, 1, 0))
```

has_edge(*p*, *s*)

Returns whether it has the edge (p, s) where s-p is a canonical vector.

INPUT:

- `p` - point in the space
- `s` - point in the space

EXAMPLES:

```
sage: from slabbe import DiscreteSubset
sage: F = lambda p: Integers(7)(2*p[0]+5*p[1])
sage: edge_predicate = lambda p,s: F(p) < F(s)
sage: D = DiscreteSubset(dimension=3, edge_predicate=edge_predicate)
sage: D.has_edge(vector((0,0)),vector((1,0)))
True
sage: D.has_edge(vector((0,0)),vector((-1,0)))
True
sage: D.has_edge(vector((-1,1)),vector((1,0)))
False
```

level_iterator(*roots=None*)

INPUT:

- `roots` - iterator of some elements in self

EXAMPLES:

```
sage: from slabbe import DiscreteSubset
sage: p = DiscreteSubset(3)
sage: root = vector((0,0,0))
sage: root.set_immutable()
sage: it = p.level_iterator(roots=[root])
sage: sorted(next(it))
[(0, 0, 0)]
sage: sorted(next(it))
[(-1, 0, 0), (0, -1, 0), (0, 0, -1), (0, 0, 1), (0, 1, 0), (1, 0, 0)]
sage: sorted(next(it))
[(-2, 0, 0),
 (-1, -1, 0),
 (-1, 0, -1),
 (-1, 0, 1),
 (-1, 1, 0),
 (0, -2, 0),
 (0, -1, -1),
 (0, -1, 1),
 (0, 0, -2),
 (0, 0, 2),
 (0, 1, -1),
```

```
 (0, 1, 1),
 (0, 2, 0),
 (1, -1, 0),
 (1, 0, -1),
 (1, 0, 1),
 (1, 1, 0),
 (2, 0, 0)]
```

```
sage: from slabbe import DiscretePlane
sage: p = DiscretePlane([1,pi,7], 1+pi+7, mu=0)
sage: root = vector((0,0,0))
sage: root.set_immutable()
sage: it = p.level_iterator(roots=[root])
sage: sorted(next(it))
[(0, 0, 0)]
sage: sorted(next(it))
[(0, 0, 1), (0, 1, 0), (1, 0, 0)]
sage: sorted(next(it))
[(-1, 0, 1),
 (-1, 1, 0),
 (0, -1, 1),
 (0, 1, 1),
 (0, 2, 0),
 (1, 0, 1),
 (1, 1, 0),
 (2, 0, 0)]
```

list()

Return the list of elements in self.

EXAMPLES:

```
sage: from slabbe import DiscretePlane, DiscreteTube
sage: P = DiscretePlane([3,4,5], 12, mu=20)
sage: tube = DiscreteTube([0,2],[0,2])
sage: I = P & tube
sage: sorted(I.list())
[(-3, -1, -1), (-3, -1, 0), (-2, -2, -1), (-2, -2, 0), (-2, -1,
-1), (-2, -1, 0), (-2, 0, -1), (-1, -1, -1)]
```

plot(*frame=False*, *edgecolor='blue'*, *pointcolor='blue'*)

Return a plot (2d or 3d) of the points and edges of self.

INPUT:

- `frame` - (default: False) if True, draw a bounding frame with labels
- `edgecolor` – string (default: `'blue'`), the color of the edges
- `pointcolor` – string (default: `'blue'`), the color of the points

EXAMPLES:

2d example:

```
sage: from slabbe import DiscreteBox
sage: box = DiscreteBox([-5,5],[-5,5])
sage: box.plot() # optional long
```

3d example:

```
sage: from slabbe import DiscretePlane, DiscreteTube
sage: P = DiscretePlane([1,3,7], 11)
sage: tube = DiscreteTube([-5,5],[-5,5])
sage: I = P & tube
sage: I.plot() # optional long
```

plot_cubes(***kwds*)

Returns the discrete object as cubes in 3d.

EXAMPLES:

```
sage: from slabbe import DiscretePlane, DiscreteTube
sage: P = DiscretePlane([3,4,5], 12, mu=20)
sage: tube = DiscreteTube([-5,5],[-5,5])
sage: I = P & tube
sage: I.plot_cubes(color='red', frame_thickness=1 # optional long)
```

TESTS:

```
sage: from slabbe import DiscreteBox
sage: box = DiscreteBox([-5,5],[-5,5])
sage: box.plot_cubes() # optional long
Traceback (most recent call last):
...
ValueError: this method is currently implemented only for objects living in 3 dimensions
```

plot_edges(*roots=None*, *color='blue'*, *m=None*)

Returns the mesh of the plane. The mesh is the union of segments joining two adjacents points.

INPUT:

- `roots` - list of some elements in self
- `color` – string (default: `'blue'`), the color of the edges
- `m` – projection matrix (default: `None`), it can be one of the following:
  - `None` - no projection is done
  - `'isometric'` - the isometric projection
  - matrix - a 2 x 3 matrix
  - `'belle'` - shortcut for `matrix(2, [1/3.0, 1, 0, 2/3.0, 0, 1])`
  - vector - defines the projection on the plane orthogonal to the vector.

EXAMPLES:

A 2d plot of a 2d object:

```
sage: from slabbe import DiscreteSubset, DiscreteBox
sage: D = DiscreteSubset(2)
sage: box = DiscreteBox([-5,5],[-5,5])
sage: I = D & box
sage: I.plot_edges(color='green') # optional long
```

A 3d plot of a 3d object:

```
sage: D = DiscreteSubset(3)
sage: box = DiscreteBox([-3,3],[-3,3],[-3,3])
sage: I = D & box
sage: I.plot_edges(color='green') # optional long
```

A 2d plot of a 3d object:

```
sage: D = DiscreteSubset(3)
sage: box = DiscreteBox([-3,3],[-3,3],[-3,3])
sage: I = D & box
sage: I.plot_edges(color='green', m='isometric') # optional long
```

plot_points(*color='blue'*, *m=None*)

Returns a 2d or 3d graphics object of the points of self.

INPUT:

- `color` – string (default: `'blue'`), the color of the points
- `m` – projection matrix (default: `None`), it can be one of the following:
  - `None` - no projection is done
  - `'isometric'` - the isometric projection
  - matrix - a 2 x n projection matrix
  - `'belle'` - shortcut for `matrix(2, [1/3.0, 1, 0, 2/3.0, 0, 1])`
  - vector - defines the projection on the plane orthogonal to the vector.

EXAMPLES:

A 2d plot of a 2d object:

```
sage: from slabbe import DiscreteSubset, DiscreteBox
sage: D = DiscreteSubset(2)
sage: box = DiscreteBox([-5,5],[-5,5])
sage: I = D & box
sage: I.plot_points(color='green')      # optional long
```

A 3d plot of a 3d object:

```
sage: D = DiscreteSubset(3)
sage: box = DiscreteBox([-5,5],[-5,5],[-5,5])
sage: I = D & box
sage: I.plot_points(color='green')      # optional long
```

A 2d plot of a 3d object:

```
sage: D = DiscreteSubset(3)
sage: box = DiscreteBox([-5,5],[-5,5],[-5,5])
sage: I = D & box
sage: I.plot_points(color='green', m='isometric')      # optional long
```

plot_points_at_distance(*k*, *roots=None*, *color='blue'*, *projmat=None*)

Plot points at distance k from the roots.

INPUT:

- `k` - integer

EXAMPLES:

```
sage: alpha = solve(x+x**2+x**3==1, x)[2].right()
sage: vv = vector((alpha, alpha+alpha**2, 1))
sage: omega = (1+alpha)**2 / 2
sage: from slabbe import DiscretePlane
sage: Pr = DiscretePlane(vv, omega, mu=pi, prec=200)
sage: Pr.plot_points_at_distance(200)               # optional long
sage: Pr.plot_points_at_distance(200, projmat='isometric') # optional long
```

projection_matrix(*m='isometric'*, *oblique=None*)

Return a projection matrix.

INPUT:

- `m` – projection matrix (default: `'isometric'`), it can be one of the following:
  - `'isometric'` - the isometric projection is used by default
  - matrix - a 2 x 3 matrix

–'belle' - shortcut for matrix(2, [1/3.0, 1, 0, 2/3.0, 0, 1])

–vector - defines the projection on the plane orthogonal to the vector.

•oblique – vector (default: None), vector perpendicular to the range space

EXAMPLES:

```
sage: from slabbe import DiscreteSubset
sage: d = DiscreteSubset(3)
sage: d.projection_matrix(vector((2,3,4))) # tolerance 0.00001
[  1.00000000000000  0.000000000000000 -0.500000000000000]
[ 0.000000000000000   1.00000000000000 -0.750000000000000]
sage: d.projection_matrix((2,3,4)) # tolerance 0.00001
[  1.00000000000000  0.000000000000000 -0.500000000000000]
[ 0.000000000000000   1.00000000000000 -0.750000000000000]
sage: d.projection_matrix()         # tolerance 0.00001
[-0.866025403784  0.866025403784             0.0]
[           -0.5            -0.5             1.0]
sage: d.projection_matrix(_) # tolerance 0.00001
[-0.866025403784439  0.866025403784439  0.000000000000000]
[-0.500000000000000 -0.500000000000000   1.00000000000000]
sage: d.projection_matrix('belle')    # tolerance 0.00001
[0.333333333333            1.0            0.0]
[0.666666666667            0.0            1.0]
```

**DiscreteSubset.tikz(projmat=[-0.866025403784439 0.866025403784439 0.000000000000000]
[-0.500000000000000 -0.500000000000000   1.00000000000000], scale=1, clip=[], contour=[], edges=Tru**

INPUT:

•projmat – (default: M3to2) 2 x dim projection matrix where dim is the dimensoin of self, the isometric projection is used by default

•scale – real number (default: 1), scaling constant for the whole figure

•clip - list (default: []), list of points whose convex hull describes a cliping path

•contour - list (default: []), list of points describing a contour path to be drawn

•edges - bool (default: True), whether to draw edges

•points - bool (default: True), whether to draw points

•axes - bool (default: False), whether to draw axes

•point_kwds - dict (default: {})

•edge_kwds - dict (default: {})

•axes_kwds - dict (default: {})

•extra_code – string (default: ''), extra tikz code to add

EXAMPLES:

```
sage: from slabbe import DiscretePlane
sage: p = DiscretePlane([2,3,5], 10)
sage: p.tikz(points=False, edges=False)
\begin{tikzpicture}
[scale=1]
\end{tikzpicture}
```

tikz_axes(*xshift=0*, *yshift=0*, *label='e'*, *projmat='isometric'*)

Return the tikz code for drawing axes.

INPUT:

•xshift - integer (default: 0), x shift

- `yshift` - integer (default: `0`), y shift
- `label` - string (default: `"e"`), label for base vectors
- `projmat` - matrix (default: `'isometric'`), projection matrix

OUTPUT:

string

EXAMPLES:

2d example:

```
sage: from slabbe import DiscreteSubset
sage: d = DiscreteSubset(2)
sage: d.tikz_axes()
%the axes
\begin{scope}[xshift=0cm,yshift=0cm]
\draw[->,>=latex, very thick, blue] (0,0) -- (1, 0);
\draw[->,>=latex, very thick, blue] (0,0) -- (0, 1);
\node at (1.40000000000000,0) {$e_1$};
\node at (0,1.40000000000000) {$e_2$};
\end{scope}
```

3d example:

```
sage: d = DiscreteSubset(3)
sage: d.tikz_axes(projmat='isometric')
%the axes
\begin{scope}
[x={(-0.866025cm,-0.500000cm)}, y={(0.866025cm,-0.500000cm)},
z={(0.000000cm,1.000000cm)}, scale=1,xshift=0,yshift=0]
\draw[fill=white] (2,0,0) rectangle (-1.8,.1,1);
\draw[->,>=latex, very thick, blue] (0,0,0) -- (1, 0, 0);
\draw[->,>=latex, very thick, blue] (0,0,0) -- (0, 1, 0);
\draw[->,>=latex, very thick, blue] (0,0,0) -- (0, 0, 1);
\node at (1.40000000000000,0,0) {$e_1$};
\node at (0,1.40000000000000,0) {$e_2$};
\node at (0,0,1.40000000000000) {$e_3$};
\end{scope}
```

tikz_edges(*style='very thick'*, *color='blue'*, *roots=None*, *projmat=None*)

Returns the mesh of the object. The mesh is the union of segments joining two adjacents points.

INPUT:

- `style` - string (default: `'dashed, very thick'`)
- `color` - string or callable (default: `'blue'`), the color of all edges or a function : (u,v) -> color of the edge (u,v)
- `roots` - list of some elements in self
- `projmat` - matrix (default: `None`), projection matrix, if None, no projection is done.

EXAMPLES:

```
sage: from slabbe import DiscretePlane
sage: p = DiscretePlane([2,3,5], 4)
sage: p.tikz_edges()
\draw[very thick, blue] (0, 0, 0) -- (1, 0, 0);
\draw[very thick, blue] (0, 0, 0) -- (0, 1, 0);
\draw[very thick, blue] (-1, 1, 0) -- (0, 1, 0);

sage: p.tikz_edges(color='orange')
\draw[very thick, orange] (0, 0, 0) -- (1, 0, 0);
```

```
\draw[very thick, orange] (0, 0, 0) -- (0, 1, 0);
\draw[very thick, orange] (-1, 1, 0) -- (0, 1, 0);

sage: c = lambda u,v: 'red' if u == 0 else 'blue'
sage: p.tikz_edges(color=c)
\draw[very thick, red] (0, 0, 0) -- (1, 0, 0);
\draw[very thick, red] (0, 0, 0) -- (0, 1, 0);
\draw[very thick, blue] (-1, 1, 0) -- (0, 1, 0);

sage: from slabbe.discrete_subset import M3to2
sage: p.tikz_edges(projmat=M3to2)
\draw[very thick, blue] (0.00000, 0.00000) -- (-0.86603, -0.50000);
\draw[very thick, blue] (0.00000, 0.00000) -- (0.86603, -0.50000);
\draw[very thick, blue] (1.73205, 0.00000) -- (0.86603, -0.50000);
```

tikz_noprojection(*projmat=None*, *scale=1*, *clip=[]*, *edges=True*, *points=True*, *axes=False*, *point_kwds={}*, *edge_kwds={}*, *axes_kwds={}*, *extra_code=''*)

Return the tikz code of self.

In this version, the points are not projected. If the points are in 3d, the tikz 3d picture is used.

INPUT:

- `projmat` – (default: None) 2*3 projection matrix for drawing unit faces, the isometric projection is used by default
- `scale` – real number (default: 1), scaling constant for the whole figure
- `clip` - list (default: `[]`), list of points describing a cliping path once projected. Works only if `self.dimension()` is 2.
- `edges` - bool (default: `True`), whether to draw edges
- `points` - bool (default: `True`), whether to draw points
- `axes` - bool (default: `False`), whether to draw axes
- `point_kwds` - dict (default: `{}`)
- `edge_kwds` - dict (default: `{}`)
- `axes_kwds` - dict (default: `{}`)
- `extra_code` – string (default: `''`), extra tikz code to add

EXAMPLES:

Object in 2d:

```
sage: from slabbe import DiscreteLine, DiscreteBox
sage: L = DiscreteLine([2,5], 2+5, mu=0)
sage: b = DiscreteBox([-5,5],[-5,5])
sage: I = L & b
sage: point_kwds = {'label':lambda p:2*p[0]+5*p[1],'label_pos':'above right'}
sage: tikz = I.tikz_noprojection(scale=0.5,point_kwds=point_kwds)
sage: len(tikz)
2659
sage: tikz
\begin{tikzpicture}
[scale=0.500000000000000]
\draw[very thick, blue] (0, 0) -- (1, 0);
\draw[very thick, blue] (0, 0) -- (0, 1);
...
\node[above right] at (-5, 2) {$0$};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (-5, 3) {};
\node[above right] at (-5, 3) {$5$};
\end{tikzpicture}
```

Object in 3d:

```
sage: from slabbe import DiscretePlane, DiscreteTube
sage: p = DiscretePlane([1,3,7], 11)
sage: d = DiscreteTube([-5,5],[-5,5])
sage: I = p & d
sage: s = I.tikz_noprojection()
sage: lines = s.splitlines()
sage: len(lines)
321
sage: print '\n'.join(lines[:4])
\begin{tikzpicture}
[x={(-0.866025cm,-0.500000cm)}, y={(0.866025cm,-0.500000cm)},
z={(0.000000cm,1.000000cm)}, scale=1]
\draw[very thick, blue] (0, 0, 0) -- (1, 0, 0);
```

tikz_points(*size='0.8mm'*, *label=None*, *label_pos='right'*, *fill='black'*, *options=''*, *roots=None*, *filter=None*, *projmat=None*)

INPUT:

- `size` - string (default: `'0.8mm'`), size of the points
- `label` - function (default: `None`), print some label next to the point
- `label_pos` - function (default: `'right'`), tikz label position
- `fill` - string (default: `'black'`), fill color
- `options` - string (default: `''`), author tikz node circle options
- `roots` - list of some elements in self
- `filter` - boolean function, if filter(p) is False, the point p is not drawn
- `projmat` - matrix (default: `None`), projection matrix, if None, no projection is done.

EXAMPLES:

```
sage: from slabbe import DiscreteBox
sage: p = DiscreteBox([0,3], [0,3], [0,3])
sage: s = p.tikz_points()
sage: lines = s.splitlines()
sage: lines[0]
'\\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (0, 0, 0) {};'
```

```
sage: from slabbe import DiscretePlane, DiscreteTube
sage: p = DiscretePlane([1,3,7], 11)
sage: d = DiscreteTube([-1,1],[-1,1])
sage: I = p & d
sage: I.tikz_points()
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (0, 0, 0) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (1, 0, 0) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (0, 1, 0) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (0, 0, 1) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (0, 1, 1) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (1, 1, 0) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (1, 0, 1) {};
```

Using a filter on the points:

```
sage: I.tikz_points(filter=lambda x:sum(x)==1)
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (1, 0, 0) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (0, 1, 0) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (0, 0, 1) {};
```

tikz_projection_scale(*projmat='isometric'*, *scale=1*, *extra=''*)

INPUT:

- `projmat` – (default: `'isometric'`) It can be one of the following:
  - `'isometric'` - the isometric projection is used by default
  - matrix - a 2 x 3 matrix
  - `'belle'` - shortcut for `matrix(2, [1/3.0, 1, 0, 2/3.0, 0, 1])`
  - vector - defines the projection on the plane orthogonal to the vector.
- `scale` – real number (default: 1), scaling constant for the whole figure
- `extra` – string (default: `''`)

EXAMPLES:

```
sage: from slabbe import DiscretePlane
sage: p = DiscretePlane([1,3,7], 11)
sage: p.tikz_projection_scale()
[x={(-0.866025cm,-0.500000cm)}, y={(0.866025cm,-0.500000cm)},
z={(0.000000cm,1.000000cm)}, scale=1]
sage: p.tikz_projection_scale(extra="xshift=4cm")
[x={(-0.866025cm,-0.500000cm)}, y={(0.866025cm,-0.500000cm)},
z={(0.000000cm,1.000000cm)}, scale=1,xshift=4cm]
```

**DiscreteTube(projmat=[-0.866025403784439 0.866025403784439 0.000000000000000]
[-0.500000000000000 -0.500000000000000 1.00000000000000], *args, **kwds)**

Bases: slabbe.discrete_subset.DiscreteSubset

Discrete Tube (preimage of a box by a projection matrix)

Subset of a discrete object such that its projection by a matrix is inside a certain box.

INPUT:

- `*args` - intervals, lists of size two : [min, max]
- `projmat` - matrix (default: `M3to2`), projection matrix

EXAMPLES:

```
sage: from slabbe import DiscreteTube
sage: DiscreteTube([-5,5],[-5,5])
DiscreteTube: Preimage of [-5, 5] x [-5, 5] by a 2 by 3 matrix

sage: m = matrix(3,4,range(12))
sage: DiscreteTube([2,10],[3,4],[6,7], projmat=m)
DiscreteTube: Preimage of [2, 10] x [3, 4] x [6, 7] by a 3 by 4 matrix
```

EXAMPLES:

```
sage: from slabbe import DiscretePlane, DiscreteTube
sage: p = DiscretePlane([1,pi,7], 1+pi+7, mu=0)
sage: tube = DiscreteTube([-5,5],[-5,5])
sage: I = p & tube
sage: I
Intersection of the following objects:
Set of points x in ZZ^3 satisfying: 0 <= (1, pi, 7) . x + 0 < pi + 8
DiscreteTube: Preimage of [-5, 5] x [-5, 5] by a 2 by 3 matrix
sage: len(list(I))
115
```

DiscreteTube.**clip**(*space=1*)

Return a good clip rectangle for this box.

INPUT:

- `space` – number (default: 1), inner space within the box

EXAMPLES:

```
sage: from slabbe import DiscreteTube
sage: tube = DiscreteTube([-6,6],[-4,3])
sage: tube.clip()
[(-5, -3), (5, -3), (5, 2), (-5, 2), (-5, -3)]
```

class slabbe.discrete_subset.**Intersection**(*objets*)

Bases: slabbe.discrete_subset.DiscreteSubset

Intersection

todo:

- Rendre l'heritage 3d automatique

INPUT:

- `objets` - un tuple d'objets discrets

EXAMPLES:

Intersection de deux plans:

```
sage: from slabbe import DiscretePlane, Intersection
sage: p = DiscretePlane([1,3,7],11)
sage: q = DiscretePlane([1,3,5],9)
sage: Intersection((p,q))
Intersection of the following objects:
Set of points x in ZZ^3 satisfying: 0 <= (1, 3, 7) . x + 0 < 11
Set of points x in ZZ^3 satisfying: 0 <= (1, 3, 5) . x + 0 < 9
```

Shortcut:

```
sage: p & q
Intersection of the following objects:
Set of points x in ZZ^3 satisfying: 0 <= (1, 3, 7) . x + 0 < 11
Set of points x in ZZ^3 satisfying: 0 <= (1, 3, 5) . x + 0 < 9
```

Intersection of a plane and a tube:

```
sage: from slabbe import DiscreteTube
sage: p = DiscretePlane([1,pi,7], 1+pi+7, mu=0)
sage: d = DiscreteTube([-5,5],[-5,5])
sage: I = p & d
sage: I
Intersection of the following objects:
Set of points x in ZZ^3 satisfying: 0 <= (1, pi, 7) . x + 0 < pi + 8
DiscreteTube: Preimage of [-5, 5] x [-5, 5] by a 2 by 3 matrix
sage: len(list(I))
115
```

Intersection of a line and a box:

```
sage: from slabbe import DiscreteLine, DiscreteBox
sage: L = DiscreteLine([2,5], 2+5, mu=0)
sage: b = DiscreteBox([-5,5],[-5,5])
sage: I = L & b
sage: I
Intersection of the following objects:
Set of points x in ZZ^2 satisfying: 0 <= (2, 5) . x + 0 < 7
[-5, 5] x [-5, 5]
```

TESTS:

Intersected objects must be of the same dimension:

```
sage: box = DiscreteBox([-5,5],[-5,5])
sage: p = DiscretePlane([1,pi,7], 1+pi+7)
sage: p & box
Traceback (most recent call last):
...
ValueError: Intersection not defined for objects not of the same dimension
```

an_element()

Returns an element in self.

EXAMPLES:

```
sage: from slabbe import DiscretePlane, DiscreteTube
sage: P = DiscretePlane([4,6,7], 17, mu=0)
sage: tube = DiscreteTube([-6.4, 6.4], [-5.2, 5.2])
sage: I = tube & P
sage: I.an_element()
(0, 0, 0)
sage: I.an_element() in I
True
```

TESTS:

```
sage: P = DiscretePlane([4,6,7], 17, mu=0)
sage: def contain(p): return 0 < P._v.dot_product(p) + P._mu <= P._omega
sage: P._predicate = contain
sage: tube = DiscreteTube([-6.4, 6.4], [-5.2, 5.2])
sage: I = tube & P
sage: I.an_element()
(0, 0, 0) not in the plane
trying similar points
(0, 0, 1)
```

has_edge(*p*, *s*)

Returns whether it has the edge (p, s) where s-p is a canonical vector.

INPUT:

- p - point in the space
- s - point in the space

EXAMPLES:

```
sage: from slabbe import DiscretePlane, DiscreteSubset
sage: d3 = DiscreteSubset(3)
sage: p = DiscretePlane([1,pi,7], 1+pi+7, mu=0)
sage: I = p & d3
sage: I.has_edge(vector((0,0,0)),vector((0,0,1)))
True
sage: I.has_edge(vector((0,0,0)),vector((0,0,-1)))
False
```

TESTS:

```
sage: from slabbe import DiscreteBox
sage: F = lambda p: (2*p[0]+5*p[1]) % 7
sage: edge_predicate = lambda p,s: F(p) < F(s)
sage: D = DiscreteSubset(dimension=2, edge_predicate=edge_predicate)
sage: b = DiscreteBox([-5,5],[-5,5])
sage: I = D & b
sage: all(I.has_edge(a,b) for a,b in I.edges_iterator())
True
```

```
sage: all(D.has_edge(a,b) for a,b in I.edges_iterator())
True

sage: from slabbe import ChristoffelGraph
sage: C = ChristoffelGraph((2,5))
sage: b = DiscreteBox([-5,5],[-5,5])
sage: I = C & b
sage: all(I.has_edge(a,b) for a,b in I.edges_iterator())
True
sage: all(C.has_edge(a,b) for a,b in I.edges_iterator())
True
```

slabbe.discrete_subset.**convex_boundary**(*L*)

EXAMPLES:

```
sage: from slabbe.discrete_subset import convex_boundary
sage: convex_boundary([(3,4), (1,2), (3,5)])
[(3, 5), (1, 2), (3, 4)]
```

1.2 Discrete Plane

Intersection of a plane and a tube:

```
sage: from slabbe import DiscretePlane, DiscreteTube
sage: p = DiscretePlane([1,pi,7], 1+pi+7, mu=0)
sage: d = DiscreteTube([-5,5],[-5,5])
sage: I = p & d
sage: I
Intersection of the following objects:
Set of points x in ZZ^3 satisfying: 0 <= (1, pi, 7) . x + 0 < pi + 8
DiscreteTube: Preimage of [-5, 5] x [-5, 5] by a 2 by 3 matrix
sage: len(list(I))
115
```

Intersection of a line and a box:

```
sage: from slabbe import DiscreteLine, DiscreteBox
sage: L = DiscreteLine([pi,sqrt(2)], pi+sqrt(2), mu=0)
sage: b = DiscreteBox([-5,5],[-5,5])
sage: I = L & b
sage: I
Intersection of the following objects:
Set of points x in ZZ^2 satisfying: 0 <= (pi, sqrt(2)) . x + 0 < pi + sqrt(2)
[-5, 5] x [-5, 5]
```

TODO:

- do some dimension checking for DiscreteLine and DiscretePlane

class slabbe.discrete_plane.**DiscreteHyperplane**(*v*, *omega*, *mu=0*, *prec=None*)

Bases: slabbe.discrete_subset.DiscreteSubset

This is the set of point p such that

$$0 \leq p \cdot v - mu < \omega$$

INPUT:

- **v** - normal vector
- omega - width

- mu - intercept (optional, default: 0)

EXAMPLES:

```
sage: from slabbe import DiscreteLine
sage: L = DiscreteLine([pi,sqrt(2)], pi+sqrt(2), mu=10)
sage: L
Set of points x in ZZ^2 satisfying: 0 <= (pi, sqrt(2)) . x + 10 < pi + sqrt(2)

sage: from slabbe import DiscretePlane
sage: p = DiscretePlane([1,pi,7], 1+pi+7, mu=0)
sage: p
Set of points x in ZZ^3 satisfying: 0 <= (1, pi, 7) . x + 0 < pi + 8

sage: from slabbe import DiscreteHyperplane
sage: p = DiscreteHyperplane([1,3,7,9], 20, mu=13)
sage: p
Set of points x in ZZ^4 satisfying: 0 <= (1, 3, 7, 9) . x + 13 < 20
```

TESTS:

```
sage: p = DiscretePlane([1,pi,7], 1+pi+7, mu=20)
sage: vector((0,0,0)) in p
False
sage: p = DiscretePlane([1,pi,7], 1+pi+7, mu=0)
sage: vector((0,0,0)) in p
True

sage: p = DiscreteHyperplane((2,3,4,5), 10)
sage: p.dimension()
4

sage: L = DiscreteLine([1,pi], 1+pi, mu=20)
sage: vector((0,0)) in L
False
sage: L = DiscreteLine([1,pi], 1+pi, mu=0)
sage: vector((0,0)) in L
True
```

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

Returns an element in self.

EXAMPLES:

```
sage: from slabbe import DiscreteHyperplane
sage: p = DiscreteHyperplane([1,pi,7], 1+pi+7, mu=10)
sage: p.an_element()
(0, 0, 0)

sage: from slabbe import DiscreteLine
sage: L = DiscreteLine([pi,sqrt(2)], pi+sqrt(2), mu=10)
sage: L.an_element()
(-2, -2)

sage: L = DiscreteLine([pi,sqrt(2)], pi+sqrt(2), mu=0)
sage: L.an_element()
(0, 0)
```

level_value(*p*)

Return the level value of a point p.

INPUT:

- p - point in the space

EXAMPLES:

```
sage: from slabbe import DiscreteHyperplane
sage: H = DiscreteHyperplane([1,3,7,9], 20, mu=13)
sage: p = H._space((1,2,3,4))
sage: H.level_value(p)
64
```

slabbe.discrete_plane.**DiscreteLine**
alias of DiscreteHyperplane

slabbe.discrete_plane.**DiscretePlane**
alias of DiscreteHyperplane

1.3 Discrete Lines

EXAMPLES:

```
sage: from slabbe import BilliardCube
sage: b = BilliardCube((1,pi,sqrt(2)))
sage: b
Cubic billiard of direction (1, pi, sqrt(2))
```

TODO:

- Rewrite some parts in cython because it is slow
- Should handle any direction
- Should use Forest structure for enumeration
- Should use +e_i only for children
- Fix documentation of class
- Fix issue with the assertion error in the step iterator
- not robust for non integral start point

class slabbe.billiard.**BilliardCube**(*v*, *start=(0, 0, 0)*)
Bases: slabbe.discrete_subset.Intersection

This is the set of point p such that

$$0 \leq p \cdot v - mu < \omega \text{ #fix me } 0 \leq p \cdot v - mu < \omega \text{ #fix me } 0 \leq p \cdot v - mu < \omega \text{ #fix me}$$

INPUT:

- **v** - directive vector
- **start** - initial point (default = (0,0,0))

EXAMPLES:

```
sage: from slabbe import BilliardCube
sage: b = BilliardCube((1,pi,sqrt(2)))
sage: b
Cubic billiard of direction (1, pi, sqrt(2))
```

```
sage: b = BilliardCube((1,pi,sqrt(2)))
sage: it = iter(b)
sage: [next(it) for _ in range(20)]
[(0, 0, 0),
 (0, 1, 0),
 (0, 1, 1),
 (0, 2, 1),
```

```
    (1, 2, 1),
    (1, 3, 1),
    (1, 3, 2),
    (1, 4, 2),
    (1, 5, 2),
    (2, 5, 2),
    (2, 6, 2),
    (2, 6, 3),
    (2, 7, 3),
    (2, 8, 3),
    (2, 8, 4),
    (3, 8, 4),
    (3, 9, 4),
    (3, 10, 4),
    (3, 10, 5),
    (3, 11, 5)]
```

::

```
    sage: b = BilliardCube((1,sqrt(2),pi), start=(11,13,14))
    sage: b.to_word()
    word: 32313233132332133231323312333213323132333...
```

an_element()

Returns an element in self.

EXAMPLES:

```
sage: from slabbe import BilliardCube
sage: b = BilliardCube((1,pi,sqrt(2)))
sage: b.an_element()
(0, 0, 0)
```

children(*p*)

Return the children of a point.

This method overwrites the methods slabbe.discrete_subset.DiscreteSubset.children(), because for billiard words, we go only in one direction in each axis.

EXAMPLES:

```
sage: from slabbe import BilliardCube
sage: b = BilliardCube((1,pi,sqrt(2)))
sage: list(b.children(vector((0,0,0))))
[(0, 1, 0)]
```

connected_component_iterator(*roots=None*)

Return an iterator over the connected component of the root.

This method overwrites the methods slabbe.discrete_subset.DiscreteSubset.connected_component_iterator(because for billiard words, we go only in one direction in each axis which allows to use a forest structure for the enumeration.

INPUT:

- roots - list of some elements immutable in self

EXAMPLES:

```
sage: from slabbe import BilliardCube
sage: p = BilliardCube([1,pi,sqrt(7)])
sage: root = vector((0,0,0))
sage: root.set_immutable()
sage: it = p.connected_component_iterator(roots=[root])
sage: [next(it) for _ in range(5)]
[(0, 0, 0), (0, 1, 0), (0, 1, 1), (0, 2, 1), (1, 2, 1)]
```

```
sage: p = BilliardCube([1,pi,7.45], start=(10.2,20.4,30.8))
sage: it = p.connected_component_iterator()
sage: [next(it) for _ in range(5)]
[(10.2000000000000, 20.4000000000000, 30.8000000000000),
 (10.2000000000000, 20.4000000000000, 31.8000000000000),
 (10.2000000000000, 21.4000000000000, 31.8000000000000),
 (10.2000000000000, 21.4000000000000, 32.8000000000000),
 (10.2000000000000, 21.4000000000000, 33.8000000000000)]
```

step_iterator()

Return an iterator coding the steps of the discrete line.

EXAMPLES:

```
sage: from slabbe import BilliardCube
sage: b = BilliardCube((1,pi,sqrt(2)))
sage: it = b.step_iterator()
sage: [next(it) for _ in range(5)]
[(0, 1, 0), (0, 0, 1), (0, 1, 0), (1, 0, 0), (0, 1, 0)]
```

TESTS:

Fix this:

```
sage: from slabbe import BilliardCube
sage: B = BilliardCube((1.1,2.2,3.3))
sage: B.to_word()
Traceback (most recent call last):
...
AssertionError: step(=(-1, 0, 1)) is not a canonical basis
vector.
```

to_word(*alphabet=[1, 2, 3]*)

Return the billiard word.

INPUT:

- `alphabet` - list

EXAMPLES:

```
sage: from slabbe import BilliardCube
sage: b = BilliardCube((1,pi,sqrt(2)))
sage: b.to_word()
word: 232123221232231223212322132223122321232 2...

sage: B = BilliardCube((sqrt(3),sqrt(5),sqrt(7)))
sage: B.to_word()
word: 321321323123213321321323123213231323 2...
```

1.4 Christoffel Graph

This module was developped for the article on a d-dimensional extension of Christoffel Words written with Christophe Reutenauer [LR2014].

EXAMPLES:

Christoffel graph in 2d (tikz code):

```
sage: from slabbe import ChristoffelGraph, DiscreteBox
sage: C = ChristoffelGraph((2,5))
```

```
sage: b = DiscreteBox([-5,5],[-5,5])
sage: I = C & b
sage: point_kwds = {'label':lambda p:C.level_value(p),'label_pos':'above right'}
sage: tikz = I.tikz_noprojection(scale=0.8,point_kwds=point_kwds)
```

Christoffel graph in 3d (tikz code):

```
sage: C = ChristoffelGraph((2,3,5))
sage: tikz = C.tikz_kernel()
```

TODO:

- Clean kernel_vector method of ChristoffelGraph

class slabbe.christoffel_graph.**ChristoffelGraph**(*v*, *mod=None*)

Bases: slabbe.discrete_subset.DiscreteSubset

Subset of a discrete object such that its projection by a matrix is inside a certain box.

INPUT:

- **v** - vector, normal vector

EXAMPLES:

```
sage: from slabbe import ChristoffelGraph
sage: ChristoffelGraph((2,5))
Christoffel set of edges for normal vector v=(2, 5)

sage: C = ChristoffelGraph((2,5))
sage: it = C.edges_iterator()
sage: it.next()
((0, 0), (1, 0))

sage: C = ChristoffelGraph((2,5,8))
sage: it = C.edges_iterator()
sage: it.next()
((0, 0, 0), (1, 0, 0))

sage: from slabbe import DiscreteBox
sage: C = ChristoffelGraph((2,5))
sage: b = DiscreteBox([-5,5],[-5,5])
sage: I = C & b
sage: point_kwds = {'label':lambda p:C.level_value(p),'label_pos':'above right'}
sage: tikz = I.tikz_noprojection(scale=0.8,point_kwds=point_kwds)
```

TEST:

This was once a bug. We make sure it is fixed:

```
sage: from slabbe import DiscreteSubset
sage: C = ChristoffelGraph((2,3,5))
sage: isinstance(C, DiscreteSubset)
True
```

has_edge(*p*, *s*)

Returns whether it has the edge (p, s) where s-p is a canonical vector.

INPUT:

- **p** - point in the space
- **s** - point in the space

EXAMPLES:

```
sage: from slabbe import ChristoffelGraph
sage: C = ChristoffelGraph((2,5,8))
sage: C.has_edge(vector((0,0,0)), vector((0,0,1)))
True
sage: C.has_edge(vector((0,0,0)), vector((0,0,2)))
False
sage: C.has_edge(vector((0,0,0)), vector((0,0,-1)))
False

sage: C = ChristoffelGraph((2,5))
sage: C.has_edge(vector((0,0)),vector((1,0)))
True
sage: C.has_edge(vector((0,0)),vector((-1,0)))
False
sage: C.has_edge(vector((-1,1)),vector((1,0)))
False
```

kernel_vector(*way='LLL'*, *verbose=False*)

todo: clean this

EXAMPLES:

```
sage: from slabbe import ChristoffelGraph
sage: C = ChristoffelGraph((2,5,7))
sage: C.kernel_vector()
[(-1, -1, 1), (3, -4, 0)]
```

level_value(*p*)

Return the level value of a point p.

INPUT:

- p - point in the space

EXAMPLES:

```
sage: from slabbe import ChristoffelGraph
sage: C = ChristoffelGraph((2,5,8))
sage: C.level_value(vector((2,3,4)))
6
sage: C.level_value(vector((1,1,1)))
0
```

**ChristoffelGraph.tikz_kernel(projmat=[-0.866025403784439 0.866025403784439 0.000000000000000]
[-0.500000000000000 -0.500000000000000 1.00000000000000], scale=1, edges=True, points=True, label**

INPUT:

- projmat – (default: M3to2) 2 x dim projection matrix where dim is the dimensoin of self, the isometric projection is used by default
- scale – real number (default: 1), scaling constant for the whole figure
- edges - bool (optional, default: True), whether to draw edges
- points - bool (optional, default: True), whether to draw points
- point_kwds - dict (default: {})
- edge_kwds - dict (default: {})
- extra_code – string (default: ''), extra tikz code to add
- way – string (default: 'LLL'), the way the base of the kernel is computed
- kernel_vector – list (default: None), the vectors, if None it uses kernel_vector() output.

EXAMPLES:

```
sage: from slabbe import ChristoffelGraph
sage: C = ChristoffelGraph((2,3,5))
sage: tikz = C.tikz_kernel()
sage: lines = tikz.splitlines()
sage: len(lines)
306
```

1.5 Double Square Tiles

If a polyomino P tiles the plane by translation, then there exists a regular tiling of the plane by P [WVL1984], i.e., where the set of translations forms a lattice. Such a polyomino was called *exact* by Wijshoff and van Leeuven. There are two types of regular tiling of the plane : square and hexagonal. These are characterized by the Beauquier-Nivat condition [BN1991]. Deciding whether a polyomino is exact can be done efficiently from the boundary and in linear time for square tiling [BFP2009]. Brlek, Fédou, Provençal also remarked that there exist polyominoes leading to more than one regular tilings but conjectured that any polyomino produces at most two regular square tilings. This conjecture was proved in [BBL2012]. In [BBGL2011], two infinite families of *double square tiles* were provided, that is polyominoes having exactly two distinct regular square tilings of the plane, namely the Christoffel tiles and the Fibonacci tiles. Finally, in [BGL2012], it was shown that any double square tile can be constructed using two simple combinatorial rules: EXTEND and SWAP.

This module is about double square tiles. Notations are chosen according to [BGL2012]. It allows to construct, study and show double square tiles. Operations TRIM, SWAP and EXTEND are implemented. Double square tiles can be shown using Sage 2D Graphics objects or using tikz.

REFERENCES:

AUTHORS:

- Sébastien Labbé, 2008: initial version
- Alexandre Blondin Massé, 2008: initial version
- Sébastien Labbé, March 2013: rewrite for inclusion into Sage

EXAMPLES:

Double Square tile from the boundary word of a known double square:

```
sage: from slabbe import DoubleSquare
sage: DoubleSquare(words.fibonacci_tile(2))
Double Square Tile
  w0 = 32303010   w4 = 10121232
  w1 = 30323      w5 = 12101
  w2 = 21232303   w6 = 03010121
  w3 = 23212      w7 = 01030
(|w0|, |w1|, |w2|, |w3|) = (8, 5, 8, 5)
(d0, d1, d2, d3)         = (10, 16, 10, 16)
(n0, n1, n2, n3)         = (0, 0, 0, 0)
```

```
sage: from slabbe import christoffel_tile
sage: DoubleSquare(christoffel_tile(4,7))
Double Square Tile
  w0 = 03                           w4 = 21
  w1 = 010301010301030101030301030  w5 = 232123232123212323232123212
  w2 = 10103010                     w6 = 32321232
  w3 = 1                            w7 = 3
(|w0|, |w1|, |w2|, |w3|) = (2, 25, 8, 1)
(d0, d1, d2, d3)         = (26, 10, 26, 10)
(n0, n1, n2, n3)         = (0, 2, 0, 0)
```

Double Square tile from the lengths of the w_i:

```
sage: DoubleSquare((4,7,4,7))
Double Square Tile
  w0 = 3232      w4 = 1010
  w1 = 1212323   w5 = 3030101
  w2 = 2121      w6 = 0303
  w3 = 0101212   w7 = 2323030
(|w0|, |w1|, |w2|, |w3|) = (4, 7, 4, 7)
(d0, d1, d2, d3)         = (14, 8, 14, 8)
(n0, n1, n2, n3)         = (0, 0, 0, 0)
```

DoubleSquare tile from the words (w_0, w_1, w_2, w_3):

```
sage: DoubleSquare(([3,2], [3], [0,3], [0,1,0,3,0]))
Double Square Tile
  w0 = 32              w4 = 10
  w1 = 3               w5 = 1
  w2 = 03              w6 = 21
  w3 = 01030           w7 = 23212
(|w0|, |w1|, |w2|, |w3|) = (2, 1, 2, 5)
(d0, d1, d2, d3)         = (6, 4, 6, 4)
(n0, n1, n2, n3)         = (0, 0, 0, 1)
```

Reduction of a double square tile:

```
sage: D = DoubleSquare(christoffel_tile(4,7))
sage: D.reduction()
['TRIM_1', 'TRIM_1', 'TRIM_2', 'TRIM_1', 'TRIM_0', 'TRIM_2']
sage: D.apply_reduction()
Double Square Tile
  w0 =     w4 =
  w1 = 0   w5 = 2
  w2 =     w6 =
  w3 = 1   w7 = 3
(|w0|, |w1|, |w2|, |w3|) = (0, 1, 0, 1)
(d0, d1, d2, d3)         = (2, 0, 2, 0)
(n0, n1, n2, n3)         = (0, NaN, 0, NaN)
```

The intermediate steps of the reduction of a double square tile:

```
sage: E,op = D.reduce()
sage: E
Double Square Tile
  w0 = 03                 w4 = 21
  w1 = 0103010103010 30   w5 = 232123232123212
  w2 = 10103010           w6 = 32321232
  w3 = 1                  w7 = 3
(|w0|, |w1|, |w2|, |w3|) = (2, 15, 8, 1)
(d0, d1, d2, d3)         = (16, 10, 16, 10)
(n0, n1, n2, n3)         = (0, 1, 0, 0)
sage: op
'TRIM_1'


sage: D.reduce_ntimes(3)
Double Square Tile
  w0 = 03      w4 = 21
  w1 = 01030   w5 = 23212
  w2 = 10      w6 = 32
  w3 = 1       w7 = 3
(|w0|, |w1|, |w2|, |w3|) = (2, 5, 2, 1)
(d0, d1, d2, d3)         = (6, 4, 6, 4)
(n0, n1, n2, n3)         = (0, 1, 0, 0)
```

Plot a double square tile and plot its reduction:

```
sage: D = DoubleSquare((34,21,34,21))
sage: D.plot()                 # long time (1s)
sage: D.plot_reduction()       # long time (1s)
```

It is not said clear enough in the articles, but double square reduction also works for double square tiles that are 8-connected polyominoes:

```
sage: D = DoubleSquare((55,34,55,34))
sage: D.plot()                 # long time (1s)
sage: D.plot_reduction()       # long time (1s)
```

class slabbe.double_square_tile.**DoubleSquare**(*data*, *rot180=None*, *steps=None*)

Bases: sage.structure.sage_object.SageObject

A double square tile.

We represent a double square tile by its boundary, that is a finite sequence on the alphabet $A = \{0, 1, 2, 3\}$ where 0 is a East step, 1 is a North step, 2 is a West step and 3 is a South step.

INPUT:

- data - can be one of the following:
  - word - word over over the alphabet A representing the boundary of a double square tile
  - tuple - tuple of 4 elements (w0,w1,w2,w3) or 8 elements (w0,w1,w2,w3,w4,w5,w6,w7) such that each wi is a sequence over the alphabet A. The condition $w_i w_{i+1} = hat(w_{i+4} w_{i+5})$ must be verified for all i modulo 8.
  - tuple - tuple of 4 integers, the lengths of (w0,w1,w2,w3)
- rot180 - WordMorphism (default: None), involution on the alphabet A and representing a rotation of 180 degrees. If None, the morphism 0->2, 1->3, 2->0, 3->1 is considered.
- steps - dict (default: None), mapping letters of A to steps in the plane. If None, the corresondance 0->(1,0), 1->(0,1), 2->(-1,0), 3->(0,-1) is considered.

EXAMPLES:

From a double square:

```
sage: from slabbe import DoubleSquare
sage: DoubleSquare(words.fibonacci_tile(1))
Double Square Tile
  w0 = 32   w4 = 10
  w1 = 3    w5 = 1
  w2 = 03   w6 = 21
  w3 = 0    w7 = 2
(|w0|, |w1|, |w2|, |w3|) = (2, 1, 2, 1)
(d0, d1, d2, d3)         = (2, 4, 2, 4)
(n0, n1, n2, n3)         = (1, 0, 1, 0)
sage: DoubleSquare(words.fibonacci_tile(2))
Double Square Tile
  w0 = 32303010   w4 = 10121232
  w1 = 30323      w5 = 12101
  w2 = 21232303   w6 = 03010121
  w3 = 23212      w7 = 01030
(|w0|, |w1|, |w2|, |w3|) = (8, 5, 8, 5)
(d0, d1, d2, d3)         = (10, 16, 10, 16)
(n0, n1, n2, n3)         = (0, 0, 0, 0)

sage: from slabbe import christoffel_tile
sage: DoubleSquare(christoffel_tile(9,7))
Double Square Tile
  w0 = 03                          w4 = 21
  w1 = 0101030101030101030         w5 = 2323212323212323212
```

```
  w2 = 10101030101030101030101010  w6 = 3232321232321232321232321232
  w3 = 1                           w7 = 3
(|w0|, |w1|, |w2|, |w3|) = (2, 19, 24, 1)
(d0, d1, d2, d3)         = (20, 26, 20, 26)
(n0, n1, n2, n3)         = (0, 0, 1, 0)
```

From the w_i:

```
sage: D = DoubleSquare(([],[],[0,1,0,1],[0,1]))
sage: D.rot180
WordMorphism: 0->2, 1->3, 2->0, 3->1
sage: D._steps
{0: (1, 0), 1: (0, 1), 2: (-1, 0), 3: (0, -1)}
sage: D
Double Square Tile
  w0 =         w4 =
  w1 =         w5 =
  w2 = 0101    w6 = 3232
  w3 = 01      w7 = 32
(|w0|, |w1|, |w2|, |w3|) = (0, 0, 4, 2)
(d0, d1, d2, d3)         = (2, 4, 2, 4)
(n0, n1, n2, n3)         = (0, 0, 2, 0)
```

One may also provide strings as long as other arguments are consistent:

```
sage: steps = {'0':(1,0), '1':(0,1), '2':(-1,0), '3': (0,-1)}
sage: rot180 = WordMorphism('0->2,2->0,3->1,1->3')
sage: DoubleSquare(('','','0101','01','','','3232','32'), rot180, steps)
Double Square Tile
  w0 =         w4 =
  w1 =         w5 =
  w2 = 0101    w6 = 3232
  w3 = 01      w7 = 32
(|w0|, |w1|, |w2|, |w3|) = (0, 0, 4, 2)
(d0, d1, d2, d3)         = (2, 4, 2, 4)
(n0, n1, n2, n3)         = (0, 0, 2, 0)
```

The first four words wi are sufficient:

```
sage: steps = {'0':(1,0), '1':(0,1), '2':(-1,0), '3': (0,-1)}
sage: rot180 = WordMorphism('0->2,2->0,3->1,1->3')
sage: DoubleSquare(('','','0101','01'), rot180, steps)
Double Square Tile
  w0 =         w4 =
  w1 =         w5 =
  w2 = 0101    w6 = 3232
  w3 = 01      w7 = 32
(|w0|, |w1|, |w2|, |w3|) = (0, 0, 4, 2)
(d0, d1, d2, d3)         = (2, 4, 2, 4)
(n0, n1, n2, n3)         = (0, 0, 2, 0)
```

alphabet()

Returns the python set of the letters that occurs in the boundary word.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.alphabet()
{0, 1, 2, 3}
```

apply(*L*)

Return the double square obtained after the application of a list of operations.

INPUT:

- L - list, list of strings

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.apply(['SWAP_0', 'EXTEND_3', 'TRIM_3'])
Double Square Tile
  w0 = 01030323      w4 = 23212101
  w1 = 21232303010   w5 = 03010121232
  w2 = 30323212      w6 = 12101030
  w3 = 10121232303   w7 = 32303010121
(|w0|, |w1|, |w2|, |w3|) = (8, 11, 8, 11)
(d0, d1, d2, d3)         = (22, 16, 22, 16)
(n0, n1, n2, n3)         = (0, 0, 0, 0)


sage: D.apply(D.reduction())
Double Square Tile
  w0 =     w4 =
  w1 = 3   w5 = 1
  w2 =     w6 =
  w3 = 2   w7 = 0
(|w0|, |w1|, |w2|, |w3|) = (0, 1, 0, 1)
(d0, d1, d2, d3)         = (2, 0, 2, 0)
(n0, n1, n2, n3)         = (0, NaN, 0, NaN)
```

apply_morphism(*m*)

INPUT:

- m - a WordMorphism

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: m = WordMorphism({0:[0],1:[1,0,1],2:[2],3:[3,2,3]})
sage: D.apply_morphism(m)
Double Square Tile
  w0 = 3232   w4 = 1010
  w1 = 323    w5 = 101
  w2 = 0323   w6 = 2101
  w3 = 0      w7 = 2
(|w0|, |w1|, |w2|, |w3|) = (4, 3, 4, 1)
(d0, d1, d2, d3)         = (4, 8, 4, 8)
(n0, n1, n2, n3)         = (1, 0, 1, 0)
```

apply_reduction()

Apply the reduction algorithm on self.

This is equivalent to `self.apply(self.reduction())`.

EXAMPLES:

```
sage: from slabbe import DoubleSquare, christoffel_tile
sage: D = DoubleSquare(christoffel_tile(9,7))
sage: D.apply_reduction()
Double Square Tile
  w0 =     w4 =
  w1 = 0   w5 = 2
  w2 =     w6 =
  w3 = 1   w7 = 3
(|w0|, |w1|, |w2|, |w3|) = (0, 1, 0, 1)
(d0, d1, d2, d3)         = (2, 0, 2, 0)
(n0, n1, n2, n3)         = (0, NaN, 0, NaN)


sage: D = DoubleSquare((5,7,4,13))
sage: D.apply_reduction()
```

```
Double Square Tile
  w0 =     w4 =
  w1 =     w5 =
  w2 = 1   w6 = 0
  w3 =     w7 =
(|w0|, |w1|, |w2|, |w3|) = (0, 0, 1, 0)
(d0, d1, d2, d3)         = (0, 1, 0, 1)
(n0, n1, n2, n3)         = (NaN, 0, NaN, 0)

sage: D = DoubleSquare((5,2,4,13))
sage: D.reduce_ntimes(3)
Double Square Tile
  w0 = 0    w4 = 0
  w1 = 12   w5 = 32
  w2 = 01   w6 = 03
  w3 = 2    w7 = 2
(|w0|, |w1|, |w2|, |w3|) = (1, 2, 2, 1)
(d0, d1, d2, d3)         = (3, 3, 3, 3)
(n0, n1, n2, n3)         = (0, 0, 0, 0)
sage: D.apply_reduction()
Traceback (most recent call last):
...
ValueError: not reducible, because self is nondegenerate and
d_0 == d_1 == 3. Also, the turning number (=0) must be +1 or -1
for the reduction to apply.
```

boundary_word()

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.boundary_word()
Path: 32303010303232123230323212101212321210 10...
```

d(i)

Return the integer d_i.

The value of d_i is defined as $d_i = |w_{i-1}| + |w_{i+1}|$.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: [D.d(i) for i in range(8)]
[10, 16, 10, 16, 10, 16, 10, 16]
```

extend(i)

Apply $EXTEND_i$ on self.

This adds a period of length d_i to w_i and w_{i+4}.

INPUT:

- **i** - integer

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.extend(3)
Double Square Tile
  w0 = 32303010                 w4 = 10121232
  w1 = 30323                    w5 = 12101
  w2 = 21232303                 w6 = 03010121
  w3 = 2321210121232303232 12   w7 = 0103032303010121010 30
(|w0|, |w1|, |w2|, |w3|) = (8, 5, 8, 21)
```

```
(d0, d1, d2, d3)          = (26, 16, 26, 16)
(n0, n1, n2, n3)          = (0, 0, 0, 1)
```

factorization_points()

Returns the eight factorization points of this configuration

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: D.factorization_points()
[0, 2, 3, 5, 6, 8, 9, 11]
```

hat()

Return the hat function returning the reversal of a word path.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.w(0)
Path: 32303010
sage: D.hat(D.w(0))
Path: 23212101
```

height()

Returns the width of this polyomino, i.e. the difference between its uppermost and lowermost coordinates

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.height()
9

sage: D = DoubleSquare((34,21,34,21))
sage: D.height()
23
```

is_degenerate()

Return whether self is degenerate.

A double square is *degenerate* if one of the w_i is empty.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: D.is_degenerate()
False

sage: rot180 = WordMorphism('0->2,2->0,3->1,1->3')
sage: D = DoubleSquare(('','0','10','1'), rot180)
sage: D.is_degenerate()
True
```

is_flat()

Return whether self is flat.

A double square is *flat* if one of the $w_i w_{i+1}$ is empty.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(1))
```

```
sage: D.is_flat()
False

sage: rot180 = WordMorphism('0->2,2->0,3->1,1->3')
sage: D = DoubleSquare(('','','0101','01'), rot180)
sage: D.is_flat()
True
```

is_morphic_pentamino()

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: D.is_morphic_pentamino()
True
```

is_singular()

Return whether self is singular.

A double square is *singular* if there exists i such that w_{i-1} and w_{i+1} are empty, equivalently if $d_i = 0$.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: D.is_singular()
False

sage: rot180 = WordMorphism('0->2,2->0,3->1,1->3')
sage: D = DoubleSquare(('','03010','','1011'), rot180)
sage: D.is_singular()
True
```

latex_8_tuple()

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.latex_8_tuple()
('{\\bf 32303010}', '{\\bf 30323}', '{\\bf 21232303}', '{\\bf 23212}',
 '{\\bf 10121232}', '{\\bf 12101}', '{\\bf 03010121}', '{\\bf 01030}')
```

latex_array()

Return a LaTeX array of self.

This code was used to create Table 1 in [BGL2012].

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: print D.latex_array()
\begin{array}{lllllll}
i & w_i & u_i & v_i & |w_i| & d_i & n_i
\\
\hline
0 & {\bf 32} & {\bf } & {\bf 32} & 2 & 2 & 1\\
1 & {\bf 3} & {\bf 3} & {\bf 032} & 1 & 4 & 0\\
2 & {\bf 03} & {\bf } & {\bf 03} & 2 & 2 & 1\\
3 & {\bf 0} & {\bf 0} & {\bf 103} & 1 & 4 & 0\\
4 & {\bf 10} & {\bf } & {\bf 10} & 2 & 2 & 1\\
5 & {\bf 1} & {\bf 1} & {\bf 210} & 1 & 4 & 0\\
6 & {\bf 21} & {\bf } & {\bf 21} & 2 & 2 & 1\\
7 & {\bf 2} & {\bf 2} & {\bf 321} & 1 & 4 & 0\\
\hline
\end{array}
```

latex_table()

Returns a Latex expression of a table containing the parameters of this double square.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.latex_table()
\begin{tabular}{|c|}
\hline
\\
\begin{tikzpicture}
[first/.style={circle,draw=black,fill=gray, inner sep=0pt, minimum size=3pt},
second/.style={rectangle,draw=black,fill=white, inner sep=0pt, minimum size=3pt}]
...
\end{tikzpicture} \\[1ex]
\hline\\
$(w_0,w_1,w_2,w_3) = (8,5,8,5)$ \\
$u_0 = 32303010$\quad $u_1 = 30323$$u_2 = 21232303$\quad $u_3 = 23212$\\
$v_0 = 30$\quad $v_1 = 2123230301O$$v_2 = 23$\quad $v_3 = 10121232303$\\
$(n_0,n_1,n_2,n_3) = (0,0,0,0)$ \\
Turning number = -1\\
Self-avoiding = True\\
\hline
\end{tabular}
```

n(*i*)

Return the integer n_i.

The value of n_i is defined as the quotient of $|w_i|$ by d_i.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: [D.n(i) for i in range(8)]
[0, 0, 0, 0, 0, 0, 0, 0]

sage: A = D.extend(1).extend(1).extend(1).extend(1)
sage: [A.n(i) for i in range(8)]
[0, 4, 0, 0, 0, 4, 0, 0]
```

If $d_i = 0$ then n_i is not defined:

```
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: B = D.reduce_ntimes(2)
sage: [B.n(i) for i in range(8)]
[0, NaN, 0, NaN, 0, NaN, 0, NaN]
```

plot(*pathoptions={'rgbcolor': 'black', 'thickness': 3}*, *fill=True*, *filloptions={'alpha': 0.2, 'rgbcolor': 'black'}*, *startpoint=True*, *startoptions={'pointsize': 100, 'rgbcolor': 'black'}*, *endarrow=True*, *arrowoptions={'width': 3, 'rgbcolor': 'black', 'arrowsize': 5}*, *gridlines=False*, *gridoptions={}*, *axes=False*)

Returns a 2d Graphics illustrating the double square tile associated to this configuration including the factorizations points.

INPUT:

- `pathoptions` - (dict, default:dict(rgbcolor='red',thickness=3)), options for the path drawing
- `fill` - (boolean, default: True), if fill is True and if the path is closed, the inside is colored
- `filloptions` - (dict, default:dict(rgbcolor='red',alpha=0.2)), ptions for the inside filling

- `startpoint` - (boolean, default: True), draw the start point?
- `startoptions` - (dict, default:dict(rgbcolor='red',pointsize=100)) options for the start point drawing
- `endarrow` - (boolean, default: True), draw an arrow end at the end?
- `arrowoptions` - (dict, default:dict(rgbcolor='red',arrowsize=20, width=3)) options for the end point arrow
- `gridlines`- (boolean, default: False), show gridlines?
- `gridoptions` - (dict, default: {}), options for the gridlines
- `axes` - (boolean, default: False), options for the axes

EXAMPLES:

The cross of area 5 together with its double square factorization points:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: D.plot()              # long time (1s)
```

plot_reduction(*ncols=3*, *options={}*)

Return a graphics array of the reduction.

INPUT:

- `ncols` - integer (default: 3), number of columns
- `options` - dict (default: {}), options given to the plot method of each double square

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.plot_reduction()          # long time (1s)
```

Using the color options:

```
sage: p = dict(rgbcolor='red', thickness=1)
sage: q = dict(rgbcolor='blue', alpha=1)
sage: options = dict(endarrow=False,startpoint=False,pathoptions=p,filloptions=q)
sage: D.plot_reduction(options=options)      # long time (1s)
```

reduce()

Reduces self by the application of TRIM or otherwise SWAP.

INPUT:

- `self` - non singular double square tile on the alphabet $0, 1, 2, 3$ such that its turning number is +1 or -1.

OUTPUT:

- DoubleSquare - the reduced double square
- string - the operation which was performed

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare((34,21,34,21))
sage: E,op = D.reduce()
sage: E
Double Square Tile
  w0 = 32303010                 w4 = 10121232
  w1 = 303232123230301030323    w5 = 12101030101212321210 1
```

```
  w2 = 21232303                w6 = 03010121
  w3 = 232121012123230323212   w7 = 0103032303012101030
(|w0|, |w1|, |w2|, |w3|) = (8, 21, 8, 21)
(d0, d1, d2, d3)         = (42, 16, 42, 16)
(n0, n1, n2, n3)         = (0, 1, 0, 1)
sage: op
'SWAP_1'

sage: D = DoubleSquare((1,2,2,1))
sage: D
Double Square Tile
  w0 = 1    w4 = 1
  w1 = 23   w5 = 03
  w2 = 12   w6 = 10
  w3 = 3    w7 = 3
(|w0|, |w1|, |w2|, |w3|) = (1, 2, 2, 1)
(d0, d1, d2, d3)         = (3, 3, 3, 3)
(n0, n1, n2, n3)         = (0, 0, 0, 0)
sage: D.reduce()
Traceback (most recent call last):
...
ValueError: not reducible, because self is nondegenerate and
d_0 == d_1 == 3. Also, the turning number (=0) must be +1 or -1
for the reduction to apply.
```

TESTS:

```
sage: D = DoubleSquare((5,4,3,4))
sage: D
Double Square Tile
  w0 = 90128   w4 = 40123
  w1 = 7659    w5 = 7654
  w2 = 012     w6 = 012
  w3 = 3765    w7 = 8765
(|w0|, |w1|, |w2|, |w3|) = (5, 4, 3, 4)
(d0, d1, d2, d3)         = (8, 8, 8, 8)
(n0, n1, n2, n3)         = (0, 0, 0, 0)
sage: D.reduce()
Traceback (most recent call last):
...
ValueError: not reducible, because self is nondegenerate and
d_0 == d_1 == 8. Also, the turning number (=-1) must be +1 or
-1 for the reduction to apply.
```

reduce_ntimes(*iteration=1*)

Reduces the double square self until it is singular.

INPUT:

- `iteration` - integer (default: 1), number of iterations to perform

OUTPUT:

DoubleSquare

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare((34,21,34,21))
sage: D.reduce_ntimes(10)
Double Square Tile
  w0 =     w4 =
  w1 = 3   w5 = 1
  w2 =     w6 =
  w3 = 2   w7 = 0
(|w0|, |w1|, |w2|, |w3|) = (0, 1, 0, 1)
```

```
(d0, d1, d2, d3)         = (2, 0, 2, 0)
(n0, n1, n2, n3)         = (0, NaN, 0, NaN)
```

reduction()

Return the list of operations to reduce self to a singular double square.

OUTPUT:

- list of strings

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare((34,21,34,21))
sage: D.reduction()
['SWAP_1', 'TRIM_1', 'TRIM_3', 'SWAP_1', 'TRIM_1', 'TRIM_3', 'TRIM_0', 'TRIM_2']

sage: from slabbe import christoffel_tile
sage: D = DoubleSquare(christoffel_tile(9,7))
sage: D.reduction()
['TRIM_2', 'TRIM_1', 'TRIM_1', 'TRIM_1', 'TRIM_0', 'TRIM_2', 'TRIM_2']
```

reverse()

Apply $REVERSE$ on self.

This reverses the words w_i.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D
Double Square Tile
  w0 = 32303010   w4 = 10121232
  w1 = 30323      w5 = 12101
  w2 = 21232303   w6 = 03010121
  w3 = 23212      w7 = 01030
(|w0|, |w1|, |w2|, |w3|) = (8, 5, 8, 5)
(d0, d1, d2, d3)         = (10, 16, 10, 16)
(n0, n1, n2, n3)         = (0, 0, 0, 0)
sage: D.reverse()
Double Square Tile
  w0 = 21232      w4 = 03010
  w1 = 30323212   w5 = 12101030
  w2 = 32303      w6 = 10121
  w3 = 01030323   w7 = 23212101
(|w0|, |w1|, |w2|, |w3|) = (5, 8, 5, 8)
(d0, d1, d2, d3)         = (16, 10, 16, 10)
(n0, n1, n2, n3)         = (0, 0, 0, 0)
sage: D.reverse().reverse() == D
True
```

shift()

Apply $SHIFT$ on self.

This replaces w_i by w_{i+1}.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D
Double Square Tile
  w0 = 32303010   w4 = 10121232
  w1 = 30323      w5 = 12101
  w2 = 21232303   w6 = 03010121
```

```
  w3 = 23212      w7 = 01030
(|w0|, |w1|, |w2|, |w3|) = (8, 5, 8, 5)
(d0, d1, d2, d3)         = (10, 16, 10, 16)
(n0, n1, n2, n3)         = (0, 0, 0, 0)
sage: D.shift()
Double Square Tile
  w0 = 30323      w4 = 12101
  w1 = 21232303   w5 = 03010121
  w2 = 23212      w6 = 01030
  w3 = 10121232   w7 = 32303010
(|w0|, |w1|, |w2|, |w3|) = (5, 8, 5, 8)
(d0, d1, d2, d3)         = (16, 10, 16, 10)
(n0, n1, n2, n3)         = (0, 0, 0, 0)
sage: D.shift().shift().shift().shift().shift().shift().shift().shift() == D
True
sage: D.shift().shift().shift().shift() == D
False
```

swap(*i*)

Apply $SWAP_i$ on self.

This replaces w_j by $\hat{w_{j+4}}$ for each $j = i, i+2, i+4, i+6$ and $w_j = (u_j * v_j)^{n_j} u_j$ by $(v_j * u_j)^{n_j} v_j$ for each $j = i+1, i+3, i+5, i+7$. This is an involution if the u_j are non empty.

INPUT:

- **i** - integer

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D
Double Square Tile
  w0 = 32303010   w4 = 10121232
  w1 = 30323      w5 = 12101
  w2 = 21232303   w6 = 03010121
  w3 = 23212      w7 = 01030
(|w0|, |w1|, |w2|, |w3|) = (8, 5, 8, 5)
(d0, d1, d2, d3)         = (10, 16, 10, 16)
(n0, n1, n2, n3)         = (0, 0, 0, 0)
sage: D.swap(1)
Double Square Tile
  w0 = 30      w4 = 12
  w1 = 32303   w5 = 10121
  w2 = 23      w6 = 01
  w3 = 21232   w7 = 03010
(|w0|, |w1|, |w2|, |w3|) = (2, 5, 2, 5)
(d0, d1, d2, d3)         = (10, 4, 10, 4)
(n0, n1, n2, n3)         = (0, 1, 0, 1)
```

tikz_boxed(*scale=1*, *boxsize=10*)

Return a tikzpicture of self included in a box.

INPUT:

- **scale** - number (default: 1), tikz scale
- **boxsize** - integer (default: 10), size of the box. If the width and height of the double square is less than the boxsize, then unit step are of size 1 and the (w_i) 8-tuple is added below the figure. Otherwise, if the width or height is larger than the boxsize, then the unit step are made smaller to fit the box and the (w_i) 8-tuple is not shown.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: D.tikz_boxed()
\begin{tabular}{c}
\begin{tikzpicture}
[scale=1]
\filldraw[-to, very thick, draw=black, fill=black!20] (0.000,
0.000) -- (0.000, -1.00) -- (-1.00, -1.00) -- (-1.00, -2.00)
-- (0.000, -2.00) -- (0.000, -3.00) -- (1.00, -3.00) -- (1.00,
-2.00) -- (2.00, -2.00) -- (2.00, -1.00) -- (1.00, -1.00) --
(1.00, 0.000) -- (0.000, 0.000);
\node[first] at (0.0000, 0.0000) {};
\node[first] at (-1.000, -2.000) {};
\node[first] at (1.000, -3.000) {};
\node[first] at (2.000, -1.000) {};
\node[second] at (-1.000, -1.000) {};
\node[second] at (0.0000, -3.000) {};
\node[second] at (2.000, -2.000) {};
\node[second] at (1.000, 0.0000) {};
\end{tikzpicture}
\\
$({\bf 32},{\bf 3},{\bf 03},{\bf 0},$ \\
$\phantom{({}{\bf 10},{\bf 1},{\bf 21},{\bf 2})$ \\
\end{tabular}
```

Smaller boxsize:

```
sage: D.tikz_boxed(boxsize=1.5)
\begin{tikzpicture}
[scale=1]
\filldraw[-to, very thick, draw=black, fill=black!20] (0.000, 0.000) --
(0.000, -0.500) -- (-0.500, -0.500) -- (-0.500, -1.00) -- (0.000, -1.00) --
(0.000, -1.50) -- (0.500, -1.50) -- (0.500, -1.00) -- (1.00, -1.00) --
(1.00, -0.500) -- (0.500, -0.500) -- (0.500, 0.000) -- (0.000, 0.000);
\node[first] at (0.0000, 0.0000) {};
\node[first] at (-0.5000, -1.000) {};
\node[first] at (0.5000, -1.500) {};
\node[first] at (1.000, -0.5000) {};
\node[second] at (-0.5000, -0.5000) {};
\node[second] at (0.0000, -1.500) {};
\node[second] at (1.000, -1.000) {};
\node[second] at (0.5000, 0.0000) {};
\end{tikzpicture}
```

tikz_commutative_diagram(*tile*, *N=1*, *scale=(1, 1)*, *labels=True*, *newcommand=True*)

Return a tikz commutative diagram for the composition.

INPUT:

- `tile` - WordMorphism, a square tile
- `N` - integer (default:`1`), length of the diagram
- `scale` - tuple of number (default:`(1,1)`), one for each line
- `labels` - arrow labels (default:`True`). It may take the following values:
  - `True` - prints TRIM, SWAP, etc.
  - `'T'` - prints T_i, etc.
  - `False` - print nothing
- `newcommand` - bool (default: `True`), whether newcommand which defines `\SWAP`, `\TRIM`, etc.

EXAMPLES:

The following command creates the tikz code for Figure 16 in [BGL2012]:

```
sage: from slabbe import DoubleSquare
sage: fibo2 = words.fibonacci_tile(2)
sage: S = WordMorphism({0:[0,0],1:[1,0,1],2:[2,2],3:[3,2,3]}, codomain=fibo2.parent())
sage: cfibo2 = DoubleSquare(fibo2)
sage: options = dict(tile=S,N=3,scale=(0.25,0.15),labels=True,newcommand=True)
sage: s = cfibo2.tikz_commutative_diagram(**options)     # long time (2s)
sage: s                                                  # long time
\newcommand{\TRIM}{\textsc{trim}}
\newcommand{\EXTEND}{\textsc{extend}}
\newcommand{\SWAP}{\textsc{swap}}
\newcommand{\SHIFT}{\textsc{shift}}
\newcommand{\REVERSE}{\textsc{reverse}}
\begin{tikzpicture}
[first/.style={circle,draw=black,fill=black, inner sep=0pt, minimum size=3pt},
second/.style={circle,draw=black,fill=white, inner sep=0pt, minimum size=3pt}]
\node (q0) at (0, 0) {\begin{tikzpicture}
[scale=0.250000000000000]
...
\end{tikzpicture}};
\path[thick, ->] (r0) edge node[midway, rectangle, fill=white, rotate=90] {$\SWAP_1$} (r1);
\path[thick, ->] (r1) edge node[midway, rectangle, fill=white, rotate=90] {$\TRIM_1$} (r2);
\path[thick, ->] (r2) edge node[midway, rectangle, fill=white, rotate=90] {$\TRIM_3$} (r3);
\path[thick, ->] (q0) edge node[midway, left] {$\varphi$} (r0);
\path[thick, ->] (q1) edge node[midway, left] {$\varphi$} (r1);
\path[thick, ->] (q2) edge node[midway, left] {$\varphi$} (r2);
\path[thick, ->] (q3) edge node[midway, left] {$\varphi$} (r3);
\end{tikzpicture}
```

tikz_reduction(*scale=1*, *ncols=3*, *gridstep=5*, *labels=True*, *newcommand=True*)

INPUT:

- `scale` - number
- `ncols` - integer, number of columns displaying the reduction
- `gridstep` - number (default: `5`), the gridstep for the snake node positions
- `labels` - arrow labels (default:`True`). It may take the following values:
  - `True` - prints TRIM, SWAP, etc.
  - `'T'` - prints T_i, etc.
  - `False` - print nothing
- `newcommand` - bool (default: `True`), whether newcommand which defines `\SWAP`, `\TRIM`, etc.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: fibo2 = words.fibonacci_tile(2)
sage: cfibo2 = DoubleSquare(fibo2)
sage: s = cfibo2.tikz_reduction(scale=0.5,ncols=4,labels=True)
sage: s
\newcommand{\TRIM}{\textsc{trim}}
\newcommand{\EXTEND}{\textsc{extend}}
\newcommand{\SWAP}{\textsc{swap}}
\newcommand{\SHIFT}{\textsc{shift}}
\newcommand{\REVERSE}{\textsc{reverse}}
\begin{tikzpicture}
[first/.style={circle,draw=black,fill=black, inner sep=0pt, minimum size=3pt},
second/.style={circle,draw=black,fill=white, inner sep=0pt, minimum size=3pt}]
\node (q0) at (0, 0) {
\begin{tikzpicture}
[scale=0.500000000000000]
...
\end{tikzpicture}
```

```
\\
$(\varepsilon,{\bf 3},\varepsilon,{\bf 2},$ \\
$\phantom{(}\varepsilon,{\bf 1},\varepsilon,{\bf 0})$ \\
\end{tabular}
};
\path[->] (q0) edge node[midway, rectangle, fill=white, rotate=90] {$\SWAP_1$} (q1);
\path[->] (q1) edge node[midway, rectangle, fill=white, rotate=90] {$\TRIM_1$} (q2);
\path[->] (q2) edge node[midway, rectangle, fill=white, rotate=90] {$\TRIM_3$} (q3);
\path[->] (q3) edge node[midway, rectangle, fill=white] {$\TRIM_0$} (q4);
\path[->] (q4) edge node[midway, rectangle, fill=white, rotate=90] {$\TRIM_2$} (q5);
\end{tikzpicture}
```

```
sage: S = WordMorphism({0:[0,0],1:[1,0,1],2:[2,2],3:[3,2,3]}, codomain=fibo2.parent())
sage: cSfibo2 = cfibo2.apply_morphism(S)
sage: s = cSfibo2.tikz_reduction(scale=0.15,ncols=4,labels='T')
sage: s
\newcommand{\TRIM}{\textsc{trim}}
\newcommand{\EXTEND}{\textsc{extend}}
\newcommand{\SWAP}{\textsc{swap}}
\newcommand{\SHIFT}{\textsc{shift}}
\newcommand{\REVERSE}{\textsc{reverse}}
\begin{tikzpicture}
[first/.style={circle,draw=black,fill=black, inner sep=0pt, minimum size=3pt},
second/.style={circle,draw=black,fill=white, inner sep=0pt, minimum size=3pt}]
\node (q0) at (0, 0) {
\begin{tikzpicture}
[scale=0.150000000000000]
...
\end{tikzpicture}
\\
$(\varepsilon,{\bf 323},\varepsilon,{\bf 22},$ \\
$\phantom{(}\varepsilon,{\bf 101},\varepsilon,{\bf 00})$ \\
\end{tabular}
};
\path[thick, ->] (q0) edge node[midway, above] {$T_1$} (q1);
\path[thick, ->] (q1) edge node[midway, above] {$T_2$} (q2);
\path[thick, ->] (q2) edge node[midway, above] {$T_3$} (q3);
\path[thick, ->] (q3) edge node[midway, above] {$T_4$} (q4);
\path[thick, ->] (q4) edge node[midway, above] {$T_5$} (q5);
\end{tikzpicture}
```

tikz_trajectory(*step=1*, *arrow='->'*)

Returns a tikz string describing the double square induced by this configuration together with its factorization points

The factorization points respectively get the tikz attribute 'first' and 'second' so that when including it in a tikzpicture environment, it is possible to modify the way those points appear.

INPUT:

- `step` - integer (default: 1)
- `arrow` - string (default: ->), tikz arrow shape

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: D.tikz_trajectory()
\filldraw[->, very thick, draw=black, fill=black!20] (0.000, 0.000)
-- (0.000, -1.00) -- (-1.00, -1.00) -- (-1.00, -2.00) -- (0.000, -2.00) --
(0.000, -3.00) -- (1.00, -3.00) -- (1.00, -2.00) -- (2.00, -2.00) -- (2.00,
-1.00) -- (1.00, -1.00) -- (1.00, 0.000) -- (0.000, 0.000); \node[first] at
(0.0000, 0.0000) {};
\node[first] at (-1.000, -2.000) {};
\node[first] at (1.000, -3.000) {};
```

```
\node[first] at (2.000, -1.000) {};
\node[second] at (-1.000, -1.000) {};
\node[second] at (0.0000, -3.000) {};
\node[second] at (2.000, -2.000) {};
\node[second] at (1.000, 0.0000) {};
```

trim(i)

Apply $TRIM_i$ on self.

This removes a period of length d_i to w_i and w_{i+4}.

INPUT:

- `i` - integer

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare((3,6,3,2))
sage: D.trim(1)
Double Square Tile
  w0 = 212   w4 = 030
  w1 =       w5 =
  w2 = 303   w6 = 121
  w3 = 03    w7 = 21
(|w0|, |w1|, |w2|, |w3|) = (3, 0, 3, 2)
(d0, d1, d2, d3)         = (2, 6, 2, 6)
(n0, n1, n2, n3)         = (1, 0, 1, 0)

sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.extend(3).trim(3)
Double Square Tile
  w0 = 32303010   w4 = 10121232
  w1 = 30323      w5 = 12101
  w2 = 21232303   w6 = 03010121
  w3 = 23212      w7 = 01030
(|w0|, |w1|, |w2|, |w3|) = (8, 5, 8, 5)
(d0, d1, d2, d3)         = (10, 16, 10, 16)
(n0, n1, n2, n3)         = (0, 0, 0, 0)
```

turning_number()

Return the turning number of self.

INPUT:

- `self` - double square defined on the alphabet of integers $\{0, 1, 2, 3\}$

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: D.turning_number()
1
sage: D.reverse().turning_number()
-1
```

Turning number of a degenerate double square:

```
sage: D = DoubleSquare(([],[0],[1,0],[1]))
sage: D.turning_number()
1
```

Turning number of a singular double square:

```
sage: D = DoubleSquare(([],[0,3,0,1,0],[],[1,0,1,1]))
sage: D.turning_number()
1
```

Turning number of a flat double square:

```
sage: D = DoubleSquare(([],[],[0,1,0,1],[0,1]))
sage: D.turning_number()
0
```

u(*i*)

Return the word u_i.

The word u_i is the unique word such that $w_i = (u_i * v_i)^{n_i} u_i$ where $0 \leq |u_i| < d_i$.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.u(1)
Path: 30323

sage: rot180 = WordMorphism('0->2,2->0,3->1,1->3')
sage: steps = {'0':(1,0), '1':(0,1), '2':(-1,0), '3': (0,-1)}
sage: D = DoubleSquare(('','03010','','1011'), rot180, steps)
sage: D.u(0)
word:
sage: D.u(1)
Traceback (most recent call last):
...
ValueError: u_1 is not defined when d_1 == 0
```

v(*i*)

Return the word v_i.

The word v_i is the unique word such that $w_i = (u_i * v_i)^{n_i} u_i$ where $0 \leq |u_i| < d_i$, $\hat{w_{i-3}} w_{i-1} = u_i v_i$ and $0 < |u_i| \leq d_i$.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.v(1)
Path: 21232303010

sage: rot180 = WordMorphism('0->2,2->0,3->1,1->3')
sage: steps = {'0':(1,0), '1':(0,1), '2':(-1,0), '3': (0,-1)}
sage: D = DoubleSquare(('','03010','','1011'), rot180, steps)
sage: D.v(0)
word: 030103323
sage: D.v(1)
Traceback (most recent call last):
...
ValueError: v_1 is not defined when d_1 == 0
```

verify_definition()

Checks that the input verifies the definition.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.verify_definition()

sage: DoubleSquare(([],[0],[0,1,0,1],[0,1]))
Traceback (most recent call last):
...
AssertionError: wiwi+1 = hat(wi+4,wi+5) is not verified for i=1
```

w(*i*)

Return the factor w_i

This corresponds to the new definition of configuration (solution).

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: [D.w(i) for i in range(8)]
[Path: 32, Path: 3, Path: 03, Path: 0, Path: 10, Path: 1, Path: 21, Path: 2]

sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: [D.w(i) for i in range(8)]
[Path: 32303010, Path: 30323, Path: 21232303, Path: 23212, Path: 10121232, Path: 12101, Path: 03010121, Path: 01030]
```

width()

Returns the width of this polyomino, i.e. the difference between its rightmost and leftmost coordinates

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.width()
9

sage: D = DoubleSquare((34,21,34,21))
sage: D.width()
23
```

slabbe.double_square_tile.**christoffel_tile**(*p*, *q*)

Returns the (p, q) Christoffel Tile [BBGL2011].

EXAMPLES:

sage: from slabbe import christoffel_tile sage: christoffel_tile(7,9) Path: 03010301010301010301010301030101030101030... sage: christoffel_tile(9,7) Path: 03010103010103010301010103010301030101030... sage: christoffel_tile(2,3) Path: 03010301010301012123212323212323 sage: christoffel_tile(0,1) Path: 03012123 sage: print christoffel_tile(4,5) 03010301010301010301010301012123212323212323212323212323

slabbe.double_square_tile.**double_square_from_boundary_word**(*ds*)

Creates a double square object from the boundary word of a double square tile.

INPUT:

- ds - word, the boundary of a double square. The parent alphabet is assumed to be in the order : East, North, West, South.

OUTPUT:

- tuple - tuple of 8 words over the alphabet A
- WordMorphism, involution on the alphabet A and representing a rotation of 180 degrees.
- dict - mapping letters of A to steps in the plane.

EXAMPLES:

```
sage: from slabbe.double_square_tile import double_square_from_boundary_word
sage: fibo = words.fibonacci_tile
sage: W, rot180, steps = double_square_from_boundary_word(fibo(1))
sage: map(len, W)
[2, 1, 2, 1, 2, 1, 2, 1]
sage: W, rot180, steps = double_square_from_boundary_word(fibo(2))
sage: map(len, W)
```

```
[8, 5, 8, 5, 8, 5, 8, 5]
sage: W, rot180, steps = double_square_from_boundary_word(fibo(3))  # long time (6s)
sage: map(len, W)                                                  # long time
[34, 21, 34, 21, 34, 21, 34, 21]
sage: rot180                                                       # long time
WordMorphism: 0->2, 1->3, 2->0, 3->1
```

slabbe.double_square_tile.**double_square_from_four_integers**(*l0*, *l1*, *l2*, *l3*)

Creates a double square from the lengths of the w_i.

INPUT:

- `l0` - integer, length of w_0
- `l1` - integer, length of w_1
- `l2` - integer, length of w_2
- `l3` - integer, length of w_3

OUTPUT:

- tuple - tuple of 8 words over alphabet A
- WordMorphism, involution on the alphabet A and representing a rotation of 180 degrees.
- dict - mapping letters of A to steps in the plane.

EXAMPLES:

```
sage: from slabbe.double_square_tile import double_square_from_four_integers
sage: w,rot180,steps = double_square_from_four_integers(2,1,2,1)
sage: w
(Path: 21, Path: 2, Path: 32, Path: 3, Path: 03, Path: 0, Path: 10, Path: 1)
sage: rot180
WordMorphism: 0->2, 1->3, 2->0, 3->1
sage: sorted(steps.items())
[(0, (1, 0)), (1, (0, 1)), (2, (-1, 0)), (3, (0, -1))]
```

If the input integers do not define a double square uniquely, the alphabet might be larger than 8:

```
sage: w,rot180,steps = double_square_from_four_integers(4,2,4,2)
sage: w
(Path: 7601,
 Path: 76,
 Path: 5476,
 Path: 54,
 Path: 2354,
 Path: 23,
 Path: 0123,
 Path: 01)
sage: rot180
WordMorphism: 0->4, 1->5, 2->6, 3->7, 4->0, 5->1, 6->2, 7->3
sage: sorted(steps.items())
[(0, (1, 0)),
 (1, (1/2*sqrt(2), 1/2*sqrt(2))),
 (2, (0, 1)),
 (3, (-1/2*sqrt(2), 1/2*sqrt(2))),
 (4, (-1, 0)),
 (5, (-1/2*sqrt(2), -1/2*sqrt(2))),
 (6, (0, -1)),
 (7, (1/2*sqrt(2), -1/2*sqrt(2)))]
```

slabbe.double_square_tile.**figure_11_BGL2012**(*scale=0.5*, *boxsize=10*, *newcommand=True*)

Return the tikz code of the Figure 11 for the article [BGL2012].

INPUT:

- scale - number (default: 0.5), tikz scale
- boxsize - integer (default: 10), size of box the double squares must fit in
- newcommand - bool (default: True), whether to include latex newcommand for TRIM, EXTEND and SWAP

EXAMPLES:

```
sage: from slabbe.double_square_tile import figure_11_BGL2012
sage: s = figure_11_BGL2012()
sage: s
\newcommand{\TRIM}{\textsc{trim}}
\newcommand{\EXTEND}{\textsc{extend}}
\newcommand{\SWAP}{\textsc{swap}}
\begin{tikzpicture}
[first/.style={circle,draw=black,fill=black, inner sep=0pt, minimum size=3pt},
second/.style={circle,draw=black,fill=white, inner sep=0pt, minimum size=3pt},
>=latex,
node distance=3cm]
\node (A) at (15,0)
{
\begin{tabular}{c}
\begin{tikzpicture}
[scale=0.5]
...
\end{tikzpicture}
};
\path[<-] (A) edge node[midway, rectangle, fill=white] {$\TRIM_2$} (B);
\path[<-] (B) edge node[midway, rectangle, fill=white] {$\TRIM_0$} (C);
\path[<-] (C) edge node[midway, rectangle, fill=white] {$\TRIM_1$} (D);
\path[<-] (D) edge node[midway, rectangle, fill=white] {$\TRIM_3$} (E);
\path[<-] (E) edge node[midway, rectangle, fill=white] {$\SWAP_1$} (F);
\path[<-] (F) edge node[midway, rectangle, fill=white] {$\TRIM_1$} (G);
\path[<-] (G) edge node[midway, rectangle, fill=white,rotate=90] {$\TRIM_3$} (H);
\path[<-] (H) edge node[midway, rectangle, fill=white,rotate=90] {$\SWAP_1$} (I);
\path[<-] (D) edge node[midway, rectangle, fill=white] {$\TRIM_2$} (E2);
\end{tikzpicture}
```

slabbe.double_square_tile.**find_square_factorisation**(*ds*, *factorisation=None*, *alternate=True*)

Return a square factorisation of the double square ds, distinct from the given factorisation.

INPUT:

- ds - word, the boundary word of a square tile
- factorisation - tuple (default: None), a known factorisation
- alternate - bool (default: True), if True the search for the second factorisation is restricted to those who alternates with the first factorisation

OUTPUT:

tuple of four positions of a square factorisation

EXAMPLES:

```
sage: from slabbe.double_square_tile import find_square_factorisation
sage: find_square_factorisation(words.fibonacci_tile(0))
(0, 1, 2, 3)
sage: find_square_factorisation(words.fibonacci_tile(1))
(0, 3, 6, 9)
sage: find_square_factorisation(words.fibonacci_tile(2))
(0, 13, 26, 39)
sage: find_square_factorisation(words.fibonacci_tile(3))
(0, 55, 110, 165)
```

```
sage: f = find_square_factorisation(words.fibonacci_tile(3))
sage: f
(0, 55, 110, 165)
sage: find_square_factorisation(words.fibonacci_tile(3),f)          # long time (6s)
(34, 89, 144, 199)
sage: find_square_factorisation(words.fibonacci_tile(3),f,False)    # long time (11s)
(34, 89, 144, 199)

sage: from slabbe import christoffel_tile
sage: find_square_factorisation(christoffel_tile(4,5))
(0, 7, 28, 35)
sage: find_square_factorisation(christoffel_tile(4,5),_)
(2, 27, 30, 55)
```

TESTS:

```
sage: find_square_factorisation(Words('abcd')('aaaaaa'))
Traceback (most recent call last):
...
ValueError: no square factorization found
sage: find_square_factorisation(Words('abcd')('aaaaaa'),(1,2,3,4))
Traceback (most recent call last):
...
ValueError: no second square factorization found
```

slabbe.double_square_tile.**snake**(*i*, *ncols=2*)

Return the coordinate of the ith node of a snake.

This is used for the tikz drawing of a double square reduction.

INPUT:

- `i` - integer, the ith node
- `ncols` - integer (default 2), number of columns

EXAMPLES:

```
sage: from slabbe.double_square_tile import snake
sage: for i in range(8): snake(i, 3)
(0, 0)
(1, 0)
(2, 0)
(2, -1)
(1, -1)
(0, -1)
(0, -2)
(1, -2)
```

slabbe.double_square_tile.**triple_square_example**(*i*)

Return a triple square factorisation example.

These words having three square factorisations were provided by Xavier Provençal.

INPUT:

- `i` - integer, accepted values are 1, 2 or 3.

EXAMPLES:

```
sage: from slabbe.double_square_tile import triple_square_example
sage: triple_square_example(1)
Path: abaBAbabaBAbabaBAbabABABabABABabABAB
sage: triple_square_example(2)
Path: abaBABaabaBABaabaBABaabABAAbabABAAbabABA...
sage: triple_square_example(3)
Path: aabAAbaabAAbaabAAbaaBAABaaBAABaaBAAB
```

Triple square tile do not exist. Hence the example provided by Xavier Provençal can not be the boundary word of a tile. One can see it by ploting it or by the fact that the turning number is zero:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(triple_square_example(1))
sage: D
Double Square Tile
  w0 = a          w4 = a
  w1 = baBA       w5 = bABA
  w2 = babaB      w6 = BabAB
  w3 = AbabaBAb   w7 = ABabABAB
(|w0|, |w1|, |w2|, |w3|) = (1, 4, 5, 8)
(d0, d1, d2, d3)         = (12, 6, 12, 6)
(n0, n1, n2, n3)         = (0, 0, 0, 1)
sage: D.turning_number()
0
```

CHAPTER

TWO

COMBINATORICS ON WORDS

2.1 Kolakoski Word

The classical Kolakoski sequence [K65] was first studied by Oldenburger [O39], where it appears as the unique solution to the problem of a trajectory on the alphabet $\{1, 2\}$ which is identical to its exponent trajectory.

See http://en.wikipedia.org/wiki/Kolakoski_sequence

It uses cython implementation inspired from the 10 lines of C code written by Dominique Bernardi and shared during Sage Days 28 in Orsay, France, in January 2011. The alphabet must be `(1,2)`.

EXAMPLES:

```
sage: from slabbe import KolakoskiWord
sage: K = KolakoskiWord()
sage: K
word: 122112122122112112212112122112112122122 1...
```

The cython implementation is much faster than the python one:

```
sage: K = KolakoskiWord()
sage: K[10^6]      # takes 0.02 seconds
2
sage: K = words.KolakoskiWord()
sage: K[10^6]      # not tested : takes too long
2
```

TESTS:

We make sure both implementation correspond for the prefix of length 100:

```
sage: Kb = KolakoskiWord()
sage: Kp = words.KolakoskiWord()
sage: Kp[:100] == Kb[:100]
True
```

REFERENCES:

AUTHORS:

- Sébastien Labbé (February 2011) - Added a Cython implementation which was easily translated from the 10 lines of C code written by Dominique Bernardi and shared during Sage Days 28 in Orsay. Needs review at #13346 since too long time...

class slabbe.kolakoski_word.**KolakoskiWord**(*parent=None*)

Bases: slabbe.kolakoski_word_pyx.WordDatatype_Kolakoski, sage.combinat.words.infinite_word.InfiniteWo

Constructor. See documentation of WordDatatype_Kolakoski for more details.

EXAMPLES:

```
sage: from slabbe import KolakoskiWord
sage: K = KolakoskiWord()
sage: K
word: 122112122122112112212112122112112122122112...
```

TESTS:

Pickle is supported:

```
sage: K = KolakoskiWord()
sage: loads(dumps(K))
word: 122112122122112112212112122112112122122112...
```

class slabbe.kolakoski_word_pyx.**WordDatatype_Kolakoski**

Bases: object

Word datatype class for the Kolakoski word.

This is a cython implementation inspired from the 10 lines of C code written by Dominique Bernardi and shared during Sage Days 28 in Orsay, France, in January 2011.

INPUT:

- parent - the parent

EXAMPLES:

```
sage: from slabbe.kolakoski_word_pyx import WordDatatype_Kolakoski
sage: parent = Words([1,2])
sage: K = WordDatatype_Kolakoski(parent)
sage: K
<slabbe.kolakoski_word_pyx.WordDatatype_Kolakoski object at ...>
sage: K[0]
1
sage: K[1]
2
```

2.2 Factor complexity and Bispecial Extension Type

This module was developped for the article on the factor complexity of infinite sequences genereated by substitutions written with Valérie Berthé [BL2014].

EXAMPLES:

The extension type of an ordinary bispecial factor:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E
  E(w)   1   2   3
   1             X
   2             X
   3     X   X   X
 m(w)=0, ordinary
sage: E.is_ordinaire()
True
```

Creation of a strong-weak pair of bispecial words from a neutral not ordinairy word:

```
sage: m = WordMorphism({1:[1,2,3],2:[2,3],3:[3]})
sage: E = ExtensionType1to1([(1,2),(2,3),(3,1),(3,2),(3,3)], [1,2,3])
sage: E
  E(w)   1   2   3
   1         X
   2             X
   3     X   X   X
 m(w)=0, not ord.
sage: E1, E2 = E.apply(m)
sage: E1
  E(3w)   1   2   3
    1
    2         X   X
    3     X   X   X
 m(w)=1, not ord.
sage: E2
  E(23w)   1   2   3
    1          X
    2
    3              X
 m(w)=-1, not ord.
```

TODO:

- use __classcall_private__ stuff for ExtensionType ?
- rename ExtensionType2to1 to ExtendedExtensionType ?
- export tikz to pdf using view instead of tikz2pdf ?
- fix bug of apply for ExtensionType2to1 when the word appears in the image of a letter
- add the bispecial word to the attribute of extension type
- use this to compute the factor complexity function

class slabbe.bispecial_extension_type.**ExtensionType**

Bases: object

cardinality()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, [1,2,3])
sage: E.cardinality()
5
```

equivalence_class()

EXAMPLES:

```
sage: from slabbe import ExtensionType2to1
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionType2to1(L, (1,2,3))
sage: len(E.equivalence_class())
6
```

static from_factor(*bispecial*, *word*)

EXAMPLES:

```
sage: from slabbe import ExtensionType
sage: W = Words([0,1,2])
sage: ExtensionType.from_factor(W(), W([0,1,1,2,0]))
  E(w)   0   1   2
   0         X
   1         X   X
```

```
  2     X
m(w)=-1, not ord.
```

static from_factor2(*bispecial*, *word*)

EXAMPLES:

```
sage: from slabbe import ExtensionType
sage: W = Words([0,1,2])
sage: ExtensionType.from_factor2(W(), W([0,1,1,2,0]))
  E(w)   0   1   2
   0         X
   1         X   X
   2     X
m(w)=-1, not ord.
```

static from_morphism(*m*)

Return the extension type of the empty word in the language defined by the image of the free monoid under the morphism m.

INPUT:

- m - endomorphim

EXAMPLES:

```
sage: from slabbe import ExtensionType
sage: ar = WordMorphism({1:[1,3],2:[2,3],3:[3]})
sage: ExtensionType.from_morphism(ar)
  E(w)   1   2   3
   1             X
   2             X
   3     X   X   X
m(w)=0, ordinary
```

```
sage: p = WordMorphism({1:[1,2,3],2:[2,3],3:[3]})
sage: ExtensionType.from_morphism(p)
  E(w)   1   2   3
   1         X
   2             X
   3     X   X   X
m(w)=0, not ord.
```

```
sage: b12 = WordMorphism({1:[1,2],2:[2],3:[3]})
sage: ExtensionType.from_morphism(b12)
  E(w)   1   2   3
   1         X
   2     X   X   X
   3     X   X   X
m(w)=2, not ord.
```

image(*m*)

EXAMPLES:

```
sage: from slabbe import ExtensionType2to1
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:          2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionType2to1(L, (1,2,3))
sage: b23 = WordMorphism({1:[1],2:[2,3],3:[3]})
sage: E.image(b23)
  E(w)   1   2   3
   31        X
   12            X
   32            X
   23    X   X   X
   33    X
```

```
m(w)=0, not ord., empty
```

is_bispecial()

EXAMPLES:

```
sage: from slabbe import ExtensionType2to1
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionType2to1(L, (1,2,3))
sage: E.is_bispecial()
True
```

is_empty()

Return whether the word associated to this extension type is empty.

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E.is_empty()
False
```

```
sage: from slabbe import ExtensionType2to1
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:          2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionType2to1(L, (1,2,3))
sage: E.is_empty()
True
```

```
sage: E = ExtensionType2to1(L, (1,2,3), empty=False)
sage: E.is_empty()
False
```

is_equivalent(*other*)

EXAMPLES:

```
sage: from slabbe import ExtensionType2to1
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionType2to1(L, (1,2,3))
sage: E.is_equivalent(E)
True
```

is_neutral()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, [1,2,3])
sage: E.is_neutral()
True
```

is_ordinaire()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E.is_ordinaire()
True
```

left_extensions()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E.left_extensions()
{1, 2, 3}
```

left_right_projection()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,2), (2,2), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E.left_right_projection()
(Counter({3: 3, 1: 1, 2: 1}), Counter({2: 3, 1: 1, 3: 1}))
```

```
sage: from slabbe import ExtensionType2to1
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:          2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionType2to1(L, (1,2,3))
sage: left, right = E.left_right_projection()
sage: sorted(left.iteritems())
[(word: 12, 3), (word: 21, 1), (word: 22, 1), (word: 23, 1), (word: 31, 1)]
sage: sorted(right.iteritems())
[(word: 1, 3), (word: 2, 3), (word: 3, 1)]
```

left_valence()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, [1,2,3])
sage: E.left_valence()
3
```

life_graph(*substitutions*, *substitutions_dict=None*)

Return the graph of extension types generated under a sequence of substitutions.

INPUT:

- `substitutions` - list of substitutions keys
- `substitutions_dict` - dict of substitutions, if None then it gets replaced by `common_substitutions_dict` defined in the module.

EXAMPLES:

From an ordinaire word:

```
sage: from slabbe import ExtensionType1to1
sage: e = ExtensionType1to1([(1,3),(2,3),(3,1),(3,2),(3,3)], [1,2,3])
sage: e.life_graph(['p23'])
Looped multi-digraph on 2 vertices
sage: e.life_graph(['p32'])
Looped multi-digraph on 3 vertices
sage: e.life_graph(['p32','p13','ar2'])
Looped multi-digraph on 5 vertices
```

2to1:

```
sage: from slabbe import ExtensionType2to1
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:     2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionType2to1(L, (1,2,3))
sage: E.life_graph(['b12','b21','b12'])
Looped multi-digraph on 8 vertices
```

life_graph_save_tikz(*filename*, *substitutions*, ***kwds*)

INPUT:

- "filename" - string
- "format" - string, default: 'tkz_graph' – either 'dot2tex' or 'tkz_graph'.

If format is 'dot2tex', then all the LaTeX generation will be delegated to "dot2tex" (which must be installed).

For the 'dot2tex' format, the possible option names and associated values are given below:

- "prog" – the program used for the layout. It must be a string corresponding to one of the software of the graphviz suite: 'dot', 'neato', 'twopi', 'circo' or 'fdp'.

See this for more options:

```
sage: g = graphs.PetersenGraph()
sage: opts = g.latex_options()
sage: opts.set_option?        # not tested
```

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: e = ExtensionType1to1([(1,3),(2,3),(3,1),(3,2),(3,3)], [1,2,3])
sage: e.life_graph_save_tikz('a.tikz', ['p32','p13','ar2'])    # not tested
```

```
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: from slabbe import ExtensionType2to1
sage: E = ExtensionType2to1(L, (1,2,3))
sage: E.life_graph_save_tikz('b.tikz', ['b12','b21','b12']) # not tested
Creation of file b.tikz
Using template '/Users/slabbe/.tikz2pdf.tex'.
tikz2pdf: calling pdflatex...
tikz2pdf: Output written to 'b.pdf'.
```

multiplicity()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, [1,2,3])
sage: E.multiplicity()
0
```

```
sage: from slabbe import ExtensionType2to1
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionType2to1(L, (1,2,3))
sage: E.multiplicity()
0
```

right_extensions()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E.right_extensions()
{1, 2, 3}
```

right_valence()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, [1,2,3])
sage: E.right_valence()
3
```

class slabbe.bispecial_extension_type.**ExtensionType1to1**(*L*, *alphabet*, *chignons=('', '')*)

Bases: slabbe.bispecial_extension_type.ExtensionType

INPUT:

- L - list of pairs of letters
- alphabet - the alphabet
- chignons - optional (default: None), pair of words added to the left and to the right of the image of the previous bispecial

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E
  E(w)   1   2   3
   1             X
   2             X
   3     X   X   X
 m(w)=0, ordinary
```

With chignons:

```
sage: E = ExtensionType1to1(L, [1,2,3], ('a','b'))
sage: E
  E(awb)   1   2   3
    1              X
    2              X
    3      X   X   X
 m(w)=0, ordinary
```

apply(*m*)

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, [1,2,3])
sage: E
  E(w)   1   2   3
   1             X
   2             X
   3     X   X   X
 m(w)=0, ordinary
sage: ar = WordMorphism({1:[1,3],2:[2,3],3:[3]})
sage: E.apply(ar)
(  E(3w)   1   2   3
    1             X
    2             X
    3     X   X   X
 m(w)=0, ordinary,)

sage: ar = WordMorphism({1:[3,1],2:[3,2],3:[3]})
sage: E.apply(ar)
(  E(w3)   1   2   3
    1             X
    2             X
```

```
    3     X   X   X
 m(w)=0, ordinary,)
```

Creation of a pair of ordinaire bispecial words from an **ordinaire** word:

```
sage: e = ExtensionType1to1([(1,3),(2,3),(3,1),(3,2),(3,3)], [1,2,3])
sage: p0 = WordMorphism({1:[1,2,3],2:[2,3],3:[3]})
sage: e.apply(p0)
(  E(3w)   1   2   3
    1
    2             X
    3     X   X   X
 m(w)=0, ordinary,)
sage: p3 = WordMorphism({1:[1,3,2],2:[2],3:[3,2]})
sage: e.apply(p3)
(  E(2w)   1   2   3
    1
    2             X
    3     X   X   X
 m(w)=0, ordinary,
   E(32w)   1   2   3
    1              X
    2      X   X   X
    3
 m(w)=0, ordinary)
```

Creation of a strong-weak pair of bispecial words from a neutral **not ordinaire** word:

```
sage: p0 = WordMorphism({1:[1,2,3],2:[2,3],3:[3]})
sage: e = ExtensionType1to1([(1,2),(2,3),(3,1),(3,2),(3,3)], [1,2,3])
sage: e.apply(p0)
(  E(3w)   1   2   3
    1
    2         X   X
    3     X   X   X
 m(w)=1, not ord.,
   E(23w)   1   2   3
    1          X
    2
    3              X
 m(w)=-1, not ord.)
```

Creation of a pair of ordinaire bispecial words from an **not ordinaire** word:

```
sage: p1 = WordMorphism({1:[1,2],2:[2],3:[3,1,2]})
sage: e = ExtensionType1to1([(1,2),(2,3),(3,1),(3,2),(3,3)], [1,2,3])
sage: e.apply(p1)
(  E(2w)   1   2   3
    1     X   X   X
    2             X
    3
 m(w)=0, ordinary,
   E(12w)   1   2   3
    1
    2          X
    3      X   X   X
 m(w)=0, ordinary)
```

This result is now fixed:

```
sage: e = ExtensionType1to1([(1,2), (3,3)], [1,2,3])
sage: p3 = WordMorphism({1:[1,3,2],2:[2],3:[3,2]})
sage: e.apply(p3)
(  E(32w)   1   2   3
    1          X
    2              X
    3
```

```
 m(w)=-1, not ord.,)

sage: e = ExtensionType1to1([(2,2),(2,3),(3,1),(3,2),(3,3)], [1,2,3])
sage: e.apply(p3)
(  E(2w)   1   2   3
    1
    2         X   X
    3     X   X   X
 m(w)=1, not ord.,)
```

This result is now fixed:

```
sage: e = ExtensionType1to1([(2,2),(2,3),(3,1),(3,2),(3,3)], [1,2,3])
sage: p2 = WordMorphism({1:[1],2:[2,3,1],3:[3,1]})
sage: e.apply(p2)
(  E(31w)   1   2   3
    1      X   X   X
    2          X   X
    3
 m(w)=1, not ord.,)

sage: e = ExtensionType1to1([(1,2),(3,3)], [1,2,3])
sage: e.apply(p2)
(  E(1w)   1   2   3
    1         X
    2
    3             X
 m(w)=-1, not ord.,)
```

TESTS:

```
sage: L = [(1,3)]
sage: E = ExtensionType1to1(L, [1,2,3])
sage: E
  E(w)   1   2   3
   1             X
   2
   3
 m(w)=0, not ord.
sage: ar = WordMorphism({1:[1,3],2:[2,3],3:[3]})
sage: E.apply(ar)
()
```

POSSIBLE BUG:

```
sage: from slabbe import ExtensionType
sage: b23 = WordMorphism({1:[1],2:[2,3],3:[3]})
sage: b13 = WordMorphism({1:[1,3],2:[2],3:[3]})
sage: b31 = WordMorphism({1:[1],2:[2],3:[3,1]})
sage: e = ExtensionType.from_morphism(b23)
sage: r = e.apply(b23)[0]
sage: r.apply(b13)
()
sage: r.apply(b31)
()
```

cardinality()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, [1,2,3])
sage: E.cardinality()
5
```

chignons_multiplicity_tuple()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, [1,2,3], ('a', 'b'))
sage: E.chignons_multiplicity_tuple()
('a', 'b', 0)
```

is_ordinaire()

EXAMPLES:

ordinary:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E
  E(w)   1   2   3
   1             X
   2             X
   3     X   X   X
 m(w)=0, ordinary
sage: E.is_ordinaire()
True
```

strong:

```
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3), (1,1)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E.is_ordinaire()
False
```

neutral but not ordinary:

```
sage: L = [(1,1), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E
  E(w)   1   2   3
   1     X
   2             X
   3     X   X   X
 m(w)=0, not ord.
sage: E.is_neutral()
True
sage: E.is_ordinaire()
False
```

not neutral, not ordinaire:

```
sage: L = [(1,1), (2,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E
  E(w)   1   2   3
   1     X
   2     X
   3         X   X
 m(w)=-1, not ord.
sage: E.is_neutral()
False
sage: E.is_ordinaire()
False
```

left_extensions()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, [1,2,3])
sage: E.left_extensions()
{1, 2, 3}
```

right_extensions()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, [1,2,3])
sage: E.right_extensions()
{1, 2, 3}
```

table()

return a table representation of self.

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E.table()
  E(w)   1   2   3
   1             X
   2             X
   3     X   X   X

sage: E = ExtensionType1to1(L, alphabet=(1,2,3), chignons=('a', 'b'))
sage: E.table()
  E(awb)   1   2   3
    1              X
    2              X
    3      X   X   X
```

class slabbe.bispecial_extension_type.**ExtensionType2to1**(*L*, *alphabet*, *chignons=('', '')*, *factors_length_2=None*, *empty=None*)

Bases: slabbe.bispecial_extension_type.ExtensionType

Generalized to words.

INPUT:

- L - list of pairs of *words*
- alphabet - the alphabet
- chignons - optional (default: None), pair of words added to the left and to the right of the image of the previous bispecial
- factors_length_2 - list of factors of length 2. If None, they are computed from the provided extension assuming the bispecial factor is *empty*.
- empty - bool, (optional, default: None), if None, then it is computed from the chignons and takes value True iff the chignons are empyt.

EXAMPLES:

```
sage: from slabbe import ExtensionType2to1
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:          2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionType2to1(L, (1,2,3))
sage: E
  E(w)   1   2   3
   21        X
```

```
   31        X
   12    X   X   X
   22    X
   23    X
m(w)=0, not ord., empty
```

apply(*m*, *l=2*, *r=1*)

The code works for Brun here because we take length 2 on the left and length 1 on the right.

On utilise les facteurs de longueur 2 pour completer l'info qui peut manquer.

TODO: bien corriger les facteurs de longueurs 2 de l'image!!!

INPUT:

- `m` - substitution
- `l` - integer, length of left extension
- `r` - integer, length of right extension

OUTPUT:

list of Extension type of the bispecial images

POSSIBLE BUG:

```
sage: from slabbe import ExtensionType
sage: b23 = WordMorphism({1:[1],2:[2,3],3:[3]})
sage: b13 = WordMorphism({1:[1,3],2:[2],3:[3]})
sage: b31 = WordMorphism({1:[1],2:[2],3:[3,1]})
sage: e = ExtensionType.from_morphism(b23)
sage: r = e.apply(b23)[0]
sage: r.apply(b13)
()
sage: r.apply(b31)
()
```

On a le meme bug (ca se corrige avec de plus grandes extensions a gauche):

```
sage: from slabbe import ExtensionType2to1
sage: E = ExtensionType2to1((([a],[b]) for a,b in e), (1,2,3))
sage: E.apply(b23)[0].apply(b13)
(  E(w)   1   2   3
   31             X
   32             X
   3      X   X   X
   13     X   X   X
   23             X
 m(w)=0, ordinary, empty,)
sage: E.apply(b23)[0].apply(b31)
(  E(w)   1   2   3
    11    X   X   X
    31    X   X   X
    12            X
    13    X
    23    X
  m(w)=0, not ord., empty,
  E(1w)   1   2   3
     1     X   X   X
     3     X   X   X
    23             X
  m(w)=2, not ord.)
```

EXAMPLES:

On imagine qu'on vient de faire b12:

```
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:      2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionType2to1(L, (1,2,3))
sage: E
  E(w)   1   2   3
   21        X
   31        X
   12    X   X   X
   22    X
   23    X
 m(w)=0, not ord., empty
sage: b12 = WordMorphism({1:[1,2],2:[2],3:[3]})
sage: E.apply(b12)
(  E(w)   1   2   3
    21        X
    31        X
    12        X
    22    X   X   X
    23    X
  m(w)=0, not ord., empty,
   E(2w)   1   2   3
    21         X
    31         X
    12     X   X   X
    22     X
  m(w)=0, ordinary)

sage: b21 = WordMorphism({1:[1],2:[2,1],3:[3]})
sage: E.apply(b21)
(  E(w)   1   2   3
    11        X
    21    X   X   X
    31        X
    12    X
    13    X
  m(w)=0, ordinary, empty,
   E(1w)   1   2   3
    21         X
    12     X   X   X
    13         X
  m(w)=0, ordinary)
sage: b23 = WordMorphism({1:[1],2:[2,3],3:[3]})
sage: E.apply(b23)
(  E(23w)   1   2   3
     31     X   X   X
     23     X
  m(w)=0, ordinary,
   E(w)   1   2   3
    31        X
    12            X
    32            X
    23    X   X   X
    33    X
  m(w)=0, not ord., empty,
   E(3w)   1   2   3
    12     X   X   X
    32     X
    23     X
  m(w)=0, ordinary)
```

cardinality()

EXAMPLES:

```
sage: from slabbe import ExtensionType2to1
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
```

```
sage: E = ExtensionType2to1(L, (1,2,3))
sage: E.cardinality()
5
```

chignons_multiplicity_tuple()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, [1,2,3], ('a', 'b'))
sage: E.chignons_multiplicity_tuple()
('a', 'b', 0)
```

extension_type_1to1()

EXAMPLES:

```
sage: from slabbe import ExtensionType2to1
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionType2to1(L, (1,2,3))
sage: E.extension_type_1to1()
  E(w)   1   2   3
   1         X
   2     X   X   X
   3     X
 m(w)=0, not ord.
```

factors_length_2()

Returns the set of factors of length 2 of the language.

This is computed from the extension type if it was not provided at the construction.

EXAMPLES:

```
sage: from slabbe import ExtensionType2to1
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionType2to1(L, (1,2,3))
sage: sorted(E.factors_length_2())
[(1, 2), (2, 1), (2, 2), (2, 3), (3, 1)]
```

is_chignons_empty()

EXAMPLES:

```
sage: from slabbe import ExtensionType2to1
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:          2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionType2to1(L, (1,2,3))
sage: E.is_chignons_empty()
True
```

is_ordinaire()

EXAMPLES:

```
sage: from slabbe import ExtensionType2to1
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:          2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionType2to1(L, (1,2,3))
sage: E.extension_type_1to1()
  E(w)   1   2   3
   1         X
   2     X   X   X
   3     X
 m(w)=0, not ord.
sage: E.is_ordinaire()
```

```
False
```

is_valid()

Return whether self is valid, i.e, each left and right extension is non empty.

EXAMPLES:

```
sage: from slabbe import ExtensionType2to1
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:          2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionType2to1(L, (1,2,3))
sage: E.is_valid()
True
```

left_extensions()

EXAMPLES:

```
sage: from slabbe import ExtensionType2to1
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionType2to1(L, (1,2,3))
sage: E.left_extensions()
{1, 2, 3}
```

left_word_extensions()

EXAMPLES:

```
sage: from slabbe import ExtensionType2to1
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:          2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionType2to1(L, (1,2,3))
sage: sorted(E.left_word_extensions())
[word: 12, word: 21, word: 22, word: 23, word: 31]
```

letters_before_and_after()

Returns a pair of dict giving the possible letters that goes before or after a letter.

Computed from the set of factors of length 2 of the language.

EXAMPLES:

```
sage: from slabbe import ExtensionType2to1
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:      2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionType2to1(L, (1,2,3))
sage: sorted(E.factors_length_2())
[(1, 2), (2, 1), (2, 2), (2, 3), (3, 1)]
sage: E.letters_before_and_after()
(defaultdict(<type 'set'>, {1: set([2, 3]), 2: set([1, 2]), 3: set([2])}),
defaultdict(<type 'set'>, {1: set([2]), 2: set([1, 2, 3]), 3: set([1])}))
```

right_extensions()

EXAMPLES:

```
sage: from slabbe import ExtensionType2to1
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionType2to1(L, (1,2,3))
sage: E.right_extensions()
{1, 2, 3}
```

right_word_extensions()

EXAMPLES:

```
sage: from slabbe import ExtensionType2to1
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:          2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionType2to1(L, (1,2,3))
sage: sorted(E.right_word_extensions())
[word: 1, word: 2, word: 3]
```

table()

return a table representation of self.

EXAMPLES:

```
sage: from slabbe import ExtensionType2to1
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:          2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionType2to1(L, (1,2,3))
sage: E
  E(w)   1   2   3
   21        X
   31        X
   12    X   X   X
   22    X
   23    X
 m(w)=0, not ord., empty
```

slabbe.bispecial_extension_type.**factors_length_2_from_morphism_and_factors_length_2**(*m*, *F*)

Return the set of factors of lengths two in the image by a morphism of a set of factors of length 2.

INPUT:

- **m** - endomorphim
- **F** - set of factors of length 2

EXAMPLES:

```
sage: from slabbe.bispecial_extension_type import factors_length_2_from_morphism_and_factors_length_2
sage: b12 = WordMorphism({1:[1,2],2:[2],3:[3]})
sage: sorted(factors_length_2_from_morphism_and_factors_length_2(b12, []))
[]
sage: sorted(factors_length_2_from_morphism_and_factors_length_2(b12, [(1,1)]))
[(1, 2), (2, 1)]
sage: b23 = WordMorphism({1:[1],2:[2,3],3:[3]})
sage: sorted(factors_length_2_from_morphism_and_factors_length_2(b23, [(1,1)]))
[(1, 1)]
```

slabbe.bispecial_extension_type.**longest_common_prefix**(*L*)

Return the longest common prefix of a list of words.

EXAMPLES:

```
sage: from slabbe.bispecial_extension_type import longest_common_prefix
sage: longest_common_prefix((Word('ab'), Word('abc'), Word('abd')))
word: ab
```

slabbe.bispecial_extension_type.**longest_common_suffix**(*L*)

Return the longest common suffix of a list of words.

EXAMPLES:

```
sage: from slabbe.bispecial_extension_type import longest_common_suffix
sage: longest_common_suffix((Word('abc'), Word('bc'), Word('xabc')))
word: bc
```

2.3 Finite words

Methods that are not in Sage (for now!)

AUTHORS:

- Sébastien Labbé, 2015

EXAMPLES:

```
sage: from slabbe.finite_word import discrepancy
sage: w = words.ChristoffelWord(5,8)
sage: discrepancy(w)
12/13
```

slabbe.finite_word.**discrepancy**(*self*)

Return the discrepancy of the word.

This is a distance to the euclidean line defined in [T1980].

EXAMPLES:

```
sage: from slabbe.finite_word import discrepancy
sage: w = words.ChristoffelWord(5,8)
sage: w
word: 0010010100101
sage: discrepancy(w)
12/13

sage: for c in w.conjugates(): print c, discrepancy(c)
0010010100101 12/13
0100101001010 7/13
1001010010100 10/13
0010100101001 10/13
0101001010010 7/13
1010010100100 12/13
0100101001001 8/13
1001010010010 9/13
0010100100101 11/13
0101001001010 6/13
1010010010100 11/13
0100100101001 9/13
1001001010010 8/13
```

REFERENCES:

2.4 Languages

EXAMPLES:

Language over all finite words on an alphabet:

```
sage: from slabbe.language import Language
sage: Language(alphabet=['a', 'b'])
Language of finite words over alphabet ['a', 'b']
```

Finite language:

```
sage: from slabbe.language import FiniteLanguage
sage: S = ['a', 'ab', 'aab', 'aaab']
sage: FiniteLanguage(alphabet=['a', 'b'], words=S)
Finite language of cardinality 4 over alphabet ['a', 'b']
```

Regular language:

```
sage: from slabbe.language import RegularLanguage
sage: alphabet = ['a', 'b']
sage: trans = [(0, 1, 'a'), (1, 2, 'b'), (2, 3, 'b'), (3, 4, 'a')]
sage: automaton = Automaton(trans, initial_states=[0], final_states=[4])
sage: RegularLanguage(alphabet, automaton)
Regular language over ['a', 'b']
defined by: Automaton with 5 states
```

Predefined languages:

```
sage: from slabbe.language import languages
sage: languages.ARP()
Regular language over [1, 2, 3, 123, 132, 213, 231, 312, 321]
defined by: Automaton with 7 states
```

AUTHORS:

- Sébastien Labbé, initial clean and full doctested version, October 2015

class slabbe.language.**FiniteLanguage**(*alphabet*, *words*)

Bases: slabbe.language.Language

Finite language

INPUT:

- `alphabet` – iterable of letters
- `words` – finite iterable of words

EXAMPLES:

```
sage: from slabbe.language import FiniteLanguage
sage: L = ['a', 'aa', 'aaa']
sage: FiniteLanguage(alphabet=['a'], words=L)
Finite language of cardinality 3 over alphabet ['a']
```

automaton()

Return the automaton recognizing this finite language.

EXAMPLES:

```
sage: from slabbe.language import FiniteLanguage
sage: L = ['a', 'aa', 'aaa']
sage: F = FiniteLanguage(alphabet=['a'], words=L)
sage: F.automaton()
Automaton with 7 states
```

minimal_automaton()

Return the minimal automaton recognizing this finite language.

Note: One of the state is not final. You may want to remove it...

EXAMPLES:

```
sage: from slabbe.language import FiniteLanguage
sage: L = ['a', 'aa', 'aaa']
sage: F = FiniteLanguage(alphabet=['a'], words=L)
sage: F.minimal_automaton()
Automaton with 5 states
```

number_of_states()

EXAMPLES:

```
sage: from slabbe.language import FiniteLanguage
sage: L = ['a', 'aa', 'aaa']
sage: F = FiniteLanguage(alphabet=['a'], words=L)
sage: F.number_of_states()
5
```

class slabbe.language.**Language**(*alphabet*)

Bases: object

Language of finite words

INPUT:

- alphabet – iterable of letters

EXAMPLES:

```
sage: from slabbe.language import Language
sage: Language(alphabet=['a', 'b'])
Language of finite words over alphabet ['a', 'b']
```

complexity(*length*)

Returns the number of words of given length.

Note: This method is defined from words_of_length_iterator().

INPUT:

- length – integer

EXAMPLES:

```
sage: from slabbe.language import Language
sage: F = Language(alphabet=['a', 'b'])
sage: map(F.complexity, range(5))
[1, 2, 4, 8, 16]
```

words_of_length_iterator(*length*)

Return an iterator over words of given length.

INPUT:

- length – integer

EXAMPLES:

```
sage: from slabbe.language import Language
sage: F = Language(alphabet=['a', 'b'])
sage: it = F.words_of_length_iterator(2)
sage: list(it)
[word: aa, word: ab, word: ba, word: bb]
```

class slabbe.language.**LanguageGenerator**

Bases: object

ARP()

Return the Arnoux-Rauzy-Poincaré regular language.

sage: from slabbe.language import languages sage: L = languages.ARP() sage: L Regular language over [1, 2, 3, 123, 132, 213, 231, 312, 321] defined by: Automaton with 7 states sage: map(L.complexity, range(4)) [1, 9, 57, 345]

Brun()

Return the Brun regular language.

EXAMPLES:

```
sage: from slabbe.language import languages
sage: L = languages.Brun()
sage: L
Regular language over [123, 132, 213, 231, 312, 321]
defined by: Automaton with 6 states
sage: map(L.complexity, range(4))
[1, 6, 18, 54]
sage: list(L.words_of_length_iterator(2))
[word: 123,123,
 word: 123,132,
 word: 123,312,
 word: 132,123,
 word: 132,132,
 word: 132,213,
 word: 213,213,
 word: 213,231,
 word: 213,321,
 word: 231,123,
 word: 231,213,
 word: 231,231,
 word: 312,231,
 word: 312,312,
 word: 312,321,
 word: 321,132,
 word: 321,312,
 word: 321,321]
```

Selmer()

Return the Selmer regular language.

EXAMPLES:

```
sage: from slabbe.language import languages
sage: L = languages.Selmer()
sage: L
Regular language over [123, 132, 213, 231, 312, 321]
defined by: Automaton with 6 states
sage: map(L.complexity, range(4))
[1, 6, 12, 24]
sage: list(L.words_of_length_iterator(2))
[word: 123,132,
 word: 123,312,
 word: 132,123,
 word: 132,213,
 word: 213,231,
 word: 213,321,
 word: 231,123,
 word: 231,213,
 word: 312,231,
 word: 312,321,
 word: 321,132,
 word: 321,312]
```

class slabbe.language.**RegularLanguage**(*alphabet*, *automaton*)

Bases: slabbe.language.Language

Regular language

INPUT:

- `alphabet` – iterable of letters
- `automaton` – finite state automaton

EXAMPLES:

```
sage: from slabbe.language import RegularLanguage
sage: alphabet = ['a', 'b']
sage: trans = [(0, 1, 'a'), (1, 2, 'b'), (2, 3, 'b'), (3, 4, 'a')]
sage: automaton = Automaton(trans, initial_states=[0], final_states=[4])
sage: RegularLanguage(alphabet, automaton)
Regular language over ['a', 'b']
defined by: Automaton with 5 states
```

words_of_length_iterator(*length*)

Return an iterator over words of given length.

INPUT:

- length – integer

EXAMPLES:

```
sage: from slabbe.language import RegularLanguage
sage: alphabet = ['a', 'b']
sage: trans = [(0, 1, 'a'), (1, 2, 'b'), (2, 3, 'b'), (3, 4, 'a')]
sage: automaton = Automaton(trans, initial_states=[0], final_states=[4])
sage: R = RegularLanguage(alphabet, automaton)
sage: [list(R.words_of_length_iterator(i)) for i in range(6)]
[[], [], [], [], [word: abba], []]
```

CHAPTER

THREE

COMBINATORICS

3.1 Joyal Bijection

Problem suggested by Doron Zeilberger during a talk done at CRM, Montreal, May 11th, 2012 to compare code in different languages. This is a implementation of the Joyal's Bijection using Sage. It will not win for the most brief code, but it is object oriented, documented, reusable, testable and allows introspection.

AUTHOR:

- Sébastien Labbé, May 12th 2012

TODO:

- Base Endofunction class on sage's FiniteSetMap classes (for both element and parent)

EXAMPLES:

3.1.1 Creation of an endofunction

```
sage: from slabbe import Endofunction
sage: L = [7, 0, 6, 1, 4, 7, 2, 1, 5]
sage: f = Endofunction(L)
sage: f
Endofunction:
[0..8] -> [7, 0, 6, 1, 4, 7, 2, 1, 5]
```

3.1.2 Creation of a double rooted tree

```
sage: from slabbe import DoubleRootedTree
sage: L = [(0,6),(2,1),(3,1),(4,2),(5,7),(6,4),(7,0),(8,5)]
sage: D = DoubleRootedTree(L, 1, 7)
sage: D
Double rooted tree:
Edges: [(0, 6), (2, 1), (3, 1), (4, 2), (5, 7), (6, 4), (7, 0), (8, 5)]
RootA: 1
RootB: 7
```

3.1.3 Joyal's bijection

From the endofunction `f`, we get a double rooted tree:

```
sage: f.to_double_rooted_tree()
Double rooted tree:
Edges: [(0, 6), (2, 1), (3, 1), (4, 2), (5, 7), (6, 4), (7, 0), (8, 5)]
RootA: 1
RootB: 7
```

From the double rooted tree `D`, we get an endofunction:

```
sage: D.to_endofunction()
Endofunction:
[0..8] -> [7, 0, 6, 1, 4, 7, 2, 1, 5]
```

In fact, we got `D` from `f` and vice versa:

```
sage: D == f.to_double_rooted_tree()
True
sage: f == D.to_endofunction()
True
```

3.1.4 Endofunctions are defined on the set [0, 1, ..., n-1]

As of now, the code supports only endofunctions defined on the set [0, 1, ..., n-1]

```
sage: L = [1, 0, 3, 4, 5, 7, 1]
sage: f = Endofunction(L)
Traceback (most recent call last):
...
ValueError: images of [0..6] must be 0 <= i < 7
```

3.1.5 Another example

From a list `L`, we create an endofunction `f`

```
sage: L = [12, 7, 8, 3, 3, 11, 11, 9, 5, 12, 0, 10, 9]
sage: f = Endofunction(L)
sage: f
Endofunction:
[0..12] -> [12, 7, 8, 3, 3, 11, 11, 9, 5, 12, 0, 10, 9]
```

From `f`, we create a double rooted tree `D`:

```
sage: D = f.to_double_rooted_tree(); D
Double rooted tree:
Edges: [(0, 12), (1, 7), (2, 8), (3, 12), (4, 3), (5, 11),
(6, 11), (7, 9), (8, 5), (10, 0), (11, 10), (12, 9)]
RootA: 9
RootB: 3
```

And from `D`, we create an endofunction:

```
sage: D.to_endofunction()
Endofunction:
[0..12] -> [12, 7, 8, 3, 3, 11, 11, 9, 5, 12, 0, 10, 9]
```

We test that we recover the initial endofunction `f`:

```
sage: f == f.to_double_rooted_tree().to_endofunction()
True
```

3.1.6 A random example

We define the set of all endofunctions on [0..7]:

```
sage: from slabbe import Endofunctions
sage: E = Endofunctions(8)
sage: E
Endofunctions of [0..7]
```

We choose a random endofunction on the set [0..7]:

```
sage: f = E.random_element()
sage: f                                # random
Endofunction:
[0..7] -> [5, 5, 0, 4, 5, 0, 1, 1]
```

We construct a double rooted tree from it:

```
sage: f.to_double_rooted_tree()        # random
Double rooted tree:
Edges: [(1, 5), (2, 0), (3, 4), (4, 5), (5, 0), (6, 1), (7, 1)]
RootA: 0
RootB: 5
```

We recover an endofunction from the double rooted tree:

```
sage: f.to_double_rooted_tree().to_endofunction()   # random
Endofunction:
[0..7] -> [5, 5, 0, 4, 5, 0, 1, 1]
```

Finally, we check the bijection:

```
sage: f == f.to_double_rooted_tree().to_endofunction()
True
```

3.1.7 Large random example

```
sage: E = Endofunctions(1000)
sage: f = E.random_element()
sage: f == f.to_double_rooted_tree().to_endofunction()
True
```

TESTS:

We test the limit cases:

```
sage: f = Endofunction([0])
sage: f == f.to_double_rooted_tree().to_endofunction()
True
sage: f = Endofunction([0,1])
sage: f == f.to_double_rooted_tree().to_endofunction()
True
sage: f = Endofunction([1,0])
sage: f == f.to_double_rooted_tree().to_endofunction()
True
```

More extensively:

```
sage: E = Endofunctions(1)
sage: all(f == f.to_double_rooted_tree().to_endofunction() for f in E)
True
```

```
sage: E = Endofunctions(2)
sage: all(f == f.to_double_rooted_tree().to_endofunction() for f in E)
True
sage: E = Endofunctions(3)
sage: all(f == f.to_double_rooted_tree().to_endofunction() for f in E)
True
sage: E = Endofunctions(4)
sage: all(f == f.to_double_rooted_tree().to_endofunction() for f in E)
True
```

TIMING TESTS:

When the extension of the file is .sage:

```
sage: E = Endofunctions(3)
sage: time all(f == f.to_double_rooted_tree().to_endofunction() for f in E) # not tested
True
Time: CPU 0.02 s, Wall: 0.02 s
sage: E = Endofunctions(4)
sage: time all(f == f.to_double_rooted_tree().to_endofunction() for f in E) # not tested
True
Time: CPU 0.22 s, Wall: 0.22 s
sage: E = Endofunctions(5)
sage: time all(f == f.to_double_rooted_tree().to_endofunction() for f in E) # not tested
True
Time: CPU 2.82 s, Wall: 2.82 s
sage: E = Endofunctions(6)
sage: time all(f == f.to_double_rooted_tree().to_endofunction() for f in E) # not tested
True
Time: CPU 45.66 s, Wall: 45.74 s
```

When the extension of the file is .spyx:

```
sage: E = Endofunctions(3)
sage: time all(f == f.to_double_rooted_tree().to_endofunction() for f in E) # not tested
True
Time: CPU 0.02 s, Wall: 0.02 s
sage: E = Endofunctions(4)
sage: time all(f == f.to_double_rooted_tree().to_endofunction() for f in E) # not tested
True
Time: CPU 0.21 s, Wall: 0.21 s
sage: E = Endofunctions(5)
sage: time all(f == f.to_double_rooted_tree().to_endofunction() for f in E) # not tested
True
Time: CPU 2.71 s, Wall: 2.72 s
sage: E = Endofunctions(6)
sage: time all(f == f.to_double_rooted_tree().to_endofunction() for f in E) # not tested
True
Time: CPU 44.08 s, Wall: 44.17 s
```

When the extension of the file is .sage:

```
sage: E = Endofunctions(1000)
sage: f = E.random_element()
sage: time f == f.to_double_rooted_tree().to_endofunction() # not tested
True
Time: CPU 0.09 s, Wall: 0.09 s
sage: E = Endofunctions(10000)
sage: f = E.random_element()
sage: time f == f.to_double_rooted_tree().to_endofunction()  # not tested
True
Time: CPU 2.23 s, Wall: 2.24 s
```

When the extension of the file is .spyx:

```
sage: E = Endofunctions(1000)
sage: f = E.random_element()
sage: time f == f.to_double_rooted_tree().to_endofunction() # not tested
True
Time: CPU 0.11 s, Wall: 0.11 s
sage: E = Endofunctions(10000)
sage: f = E.random_element()
sage: time f == f.to_double_rooted_tree().to_endofunction() # not tested
True
Time: CPU 2.91 s, Wall: 2.93 s
```

class slabbe.joyal_bijection.**DoubleRootedTree**(*edges*, *rootA*, *rootB*)

Bases: object

Returns a double rooted tree.

INPUT:

- edges - list of edges
- rootA - root A
- rootB - root B

EXAMPLES:

```
sage: from slabbe import DoubleRootedTree
sage: edges = [(0,5),(1,2),(2,6),(3,2),(4,1),(5,7),(7,1),(8,2),(9,4)]
sage: D = DoubleRootedTree(edges, 6, 0)
sage: D
Double rooted tree:
Edges: [(0, 5), (1, 2), (2, 6), (3, 2), (4, 1), (5, 7), (7, 1), (8, 2), (9, 4)]
RootA: 6
RootB: 0
```

graph()

EXAMPLES:

```
sage: from slabbe import DoubleRootedTree
sage: edges = [(0,5),(1,2),(2,6),(3,2),(4,1),(5,7),(7,1),(8,2),(9,4)]
sage: D = DoubleRootedTree(edges, 6, 0)
sage: D.graph()
Graph on 10 vertices
```

skeleton()

EXAMPLES:

```
sage: from slabbe import DoubleRootedTree
sage: edges = [(0,5),(1,2),(2,6),(3,2),(4,1),(5,7),(7,1),(8,2),(9,4)]
sage: D = DoubleRootedTree(edges, 6, 0)
sage: D.skeleton()
[0, 5, 7, 1, 2, 6]
```

skeleton_cycles()

EXAMPLES:

```
sage: from slabbe import DoubleRootedTree
sage: edges = [(0,5),(1,2),(2,6),(3,2),(4,1),(5,7),(7,1),(8,2),(9,4)]
sage: D = DoubleRootedTree(edges, 6, 0)
sage: D.skeleton()
[0, 5, 7, 1, 2, 6]
sage: D.skeleton_cycles()
[(0,), (1, 5), (2, 7, 6)]
```

to_endofunction()

EXAMPLES:

```
sage: from slabbe import DoubleRootedTree
sage: edges = [(0,5),(1,2),(2,6),(3,2),(4,1),(5,7),(7,1),(8,2),(9,4)]
sage: D = DoubleRootedTree(edges, 6, 0)
sage: D.to_endofunction()
Endofunction:
[0..9] -> [0, 5, 7, 2, 1, 1, 2, 6, 2, 4]
```

TESTS:

```
sage: D = DoubleRootedTree([], 0, 0)
sage: D.to_endofunction()
Endofunction:
[0..0] -> [0]
```

class slabbe.joyal_bijection.**Endofunction**(*L*)

Bases: object

Returns an endofunction.

INPUT:

- L - list of length n containing images of the integers from 0 to n-1 where the images belong to the integers from 0 to n-1.

EXAMPLES:

```
sage: from slabbe import Endofunction
sage: L = [0, 5, 7, 2, 1, 1, 2, 6, 2, 4]
sage: f = Endofunction(L)
sage: f
Endofunction:
[0..9] -> [0, 5, 7, 2, 1, 1, 2, 6, 2, 4]
```

cycle_elements()

Returns the list of all elements in a cycle for this endofunction.

OUTPUT:

list

EXAMPLES:

```
sage: from slabbe import Endofunction
sage: L = [6, 5, 7, 2, 1, 1, 2, 6, 2, 4]
sage: f = Endofunction(L)
sage: f.cycle_elements()   # random order
[0, 6, 7, 2, 1, 5]
```

Note: G.cycle_basis() is not implemented for directed or multiedge graphs (in Networkx). Hence, the cycle_basis method is missing the 2-cycles.

---

skeleton()

Return the skeleton of the endofunction.

OUTPUT:

list

EXAMPLES:

```
sage: from slabbe import Endofunction
sage: L = [0, 5, 7, 2, 1, 1, 2, 6, 2, 4]
sage: f = Endofunction(L)
sage: f.skeleton()
[0, 5, 7, 1, 2, 6]
```

to_double_rooted_tree()

Return the double rooted tree following André Joyal Bijection.

OUTPUT:

Double rooted tree

EXAMPLES:

```
sage: from slabbe import Endofunction
sage: L = [0, 5, 7, 2, 1, 1, 2, 6, 2, 4]
sage: f = Endofunction(L)
sage: f.to_double_rooted_tree()
Double rooted tree:
Edges: [(0, 5), (1, 2), (2, 6), (3, 2), (4, 1), (5, 7), (7, 1), (8, 2), (9, 4)]
RootA: 6
RootB: 0
```

two_cycle_elements()

Iterator over elements in a two-cycle.

EXAMPLES:

```
sage: from slabbe import Endofunction
sage: L = [0, 5, 7, 2, 1, 1, 2, 6, 2, 4]
sage: f = Endofunction(L)
sage: list(f.two_cycle_elements())
[1, 5]
```

class slabbe.joyal_bijection.**Endofunctions**(*n*)

Bases: object

Returns the set of all endofunction on the set [0..n-1].

INPUT:

- n - positive integer

EXAMPLES:

```
sage: from slabbe import Endofunctions
sage: Endofunctions(10)
Endofunctions of [0..9]
```

random_element()

Return a random endofunction on [0..n-1].

EXAMPLES:

```
sage: from slabbe import Endofunctions
sage: E = Endofunctions(10)
sage: E.random_element()          # random
Endofunction:
[0..9] -> [2, 8, 7, 0, 0, 6, 2, 3, 5, 9]
sage: E.random_element()          # random
Endofunction:
[0..9] -> [8, 7, 7, 5, 4, 1, 0, 3, 8, 6]
```

3.2 Percolation in lattices

This is an implementation of bond percolation. See Chapter 3 of [POG]. See also my blog post Percolation and self-avoiding walks related to this code.

AUTHORS:

- Sebastien Labbe (2012-12-17): initial version, for pog lecture group

REFERENCES:

EXAMPLES:

3.2.1 Bond percolation sample

We construct a bond percolation sample in dimension d=2 with probability of open edges p=0.3:

```
sage: from slabbe import BondPercolationSample
sage: S = BondPercolationSample(p=0.3, d=2)
sage: S
Bond percolation sample d=2 p=0.300
```

An edge is defined uniquely as a starting point in Z^d and an axis direction given by an integer i such that 1 <= i <= d. One may ask if a given edge is in the sample S:

```
sage: ((34,56), 2) in S     # random
False
```

The result is cached so the same answer is returned again:

```
sage: ((34,56), 2) in S     # random
False
sage: ((34,56), 2) in S     # random
False
```

The cluster containing the point zero is returned as an iterator:

```
sage: S.cluster()
<generator object at ...>
```

It may be finite of infinite. If you believe it is finite, you may compute its cardinality. If the cluster is infinite, it will not halt:

```
sage: S.cluster_cardinality() # not tested, might not halt
13
```

For larger values of p, the cluster might be larger if not infinite. In this case you may want to stop the computation at a certain point determined in advance. The following method does this. And it returns the cardinality if it is smaller than the stop value:

```
sage: S = BondPercolationSample(p=0.45, d=2)
sage: S.cluster_cardinality_stop_at(stop=10)          # random
'>=10'
sage: S.cluster_cardinality_stop_at(stop=100)          # random
'>=100'
sage: S.cluster_cardinality_stop_at(stop=1000)          # random
625
```

3.2.2 Bond percolation samples

Construction of 20 bond percolation samples. For each of them, compute the cardinality of the open cluster containing zero:

```
sage: from slabbe import BondPercolationSamples
sage: S20 = BondPercolationSamples(p=0.4, d=2, n=20)
sage: S20.cluster_cardinality(stop=100)               # random
[4, 2, 1, 4, 1, 10, 62, 71, 1, 25, 19, 2, 2, 42, '>=100', 1, 18, 2, '>=100', 20]
```

By considering "larger than 100" to be an infinite cluster, this gives a value of 2/20 = 0.10 for the percolation probability:

```
sage: S20.percolation_probability(stop=100)        # random
0.100
```

By increasing the stop value, the computations can be redone again *on the same samples*. In this case, by using a stop value of 1000, we get a value of 0 for the percolation probability of p=0.4:

```
sage: S20.cluster_cardinality(stop=1000)             # random
[4, 2, 1, 4, 1, 10, 62, 71, 1, 25, 19, 2, 2, 42, 176, 1, 18, 2, 186, 20]
sage: S20.percolation_probability(stop=1000)         # random
0.000
```

3.2.3 Percolation probability

One can define the percolation probability function for a given dimension d. It will generate n samples and consider the cluster to be infinite if its cardinality is larger than the given stop value:

```
sage: from slabbe import PercolationProbability
sage: T = PercolationProbability(d=2, n=10, stop=100)
sage: T
Percolation Probability $\theta(p)$
d = dimension = 2
n = # samples = 10
stop counting at = 100
```

Compute the value for a certain probability p:

```
sage: T(0.4534)             # random
0.300
```

Of course, this value will change for another equal percolation probability since it depends on the samples:

```
sage: T = PercolationProbability(d=2, n=10, stop=100)
sage: T(0.4534)              # random
0.600
```

Anyway, it is usefull to draw the plot of the percolation probability:

```
sage: T = PercolationProbability(d=2, n=10, stop=100)
sage: T.return_plot((0,1), adaptive_recursion=4, plot_points=4)     # optional long
Graphics object consisting of 2 graphics primitives
```

Here we use Sage adaptative recursion algorithm for drawing plots which finds the particular important intervals to ask for more values of the function. See help section of plot function for details. Because T might be long to compute we start with only 4 points

3.2.4 TODO

- Make it 100% doctested (presently 21/24 = 87%)
- Base it on DiscreteSubset code

- Fix tikz2pdf use

Do we want to use?:

```
sage: from sage.sets.set_from_iterator import EnumeratedSetFromIterator
sage: from itertools import count
sage: S = EnumeratedSetFromIterator(count)
sage: S
{0, 1, 2, 3, 4, ...}
```

and ?:

```
sage: M = FiniteSetMaps(["a", "b"], [3, 4, 5]); M
Maps from {'a', 'b'} to {3, 4, 5}
```

3.2.5 Methods and classes

class slabbe.bond_percolation.**BondPercolationSample**(*p*, *d=2*)

Bases: sage.structure.sage_object.SageObject

Let $L^d = (Z^d, E^d)$ be the hypercubic lattice.

A sample contained in the set {0,1}^{E^d}.

An edge e in E is open (=1) in the sample with probability p.

Cached __contains__ method below does the job of memory.

children(*pt*)

Return an iterator over open neighbors of the point pt.

INPUT:

- pt - tuple, point in Z^d
- m - integer, limit

EXAMPLES:

The result is consistent:

```
sage: from slabbe import BondPercolationSample
sage: S = BondPercolationSample(0.5)
sage: list(S.children((0,0)))        # random
[(1, 0), (-1, 0), (0, -1)]
```

Might be different for another sample:

```
sage: S = BondPercolationSample(0.5)
sage: list(S.children((0,0)))        # random
[(-1, 0)]
```

In dimension 3:

```
sage: S = BondPercolationSample(0.5,3)
sage: list(S.children((0,0,0)))        # random
[(1, 0, 0), (0, -1, 0), (0, 0, -1)]
sage: S = BondPercolationSample(0.5,3)
sage: list(S.children((0,0,0)))        # random
[(1, 0, 0), (-1, 0, 0)]
sage: list(S.children((0,0,0)))        # random
[(1, 0, 0), (-1, 0, 0)]
```

```
sage: S = BondPercolationSample(1,2)
sage: list(S.children((0,0)))        # random
[(1, 0), (-1, 0), (0, 1), (0, -1)]
sage: S = BondPercolationSample(1,3)
sage: list(S.children((0,0,0)))        # random
[(1, 0, 0), (-1, 0, 0), (0, 1, 0), (0, -1, 0), (0, 0, 1), (0, 0, -1)]
```

cluster(*pt=None*)

Return an iterator over the open cluster containing the point pt.

INPUT:

- `pt` - tuple, point in Z^d. If None, pt=zero is considered.

EXAMPLES:

sage: from slabbe import BondPercolationSample sage: S = BondPercolationSample(0.5) sage: it = S.cluster() sage: next(it) (0, 0)

cluster_cardinality(*pt=None*)

INPUT:

- `pt` - tuple, point in Z^d. If None, pt=zero is considered.

EXAMPLES:

```
sage: from slabbe import BondPercolationSample
sage: BondPercolationSample(0.01).cluster_cardinality() # random
1
sage: BondPercolationSample(0.4).cluster_cardinality()  # random
28
```

cluster_cardinality_stop_at(*stop*, *pt=None*)

Return the cardinality of the cluster or the strin ">=STOP" if the size is larger than stop value.

INPUT:

- `stop` - integer
- `pt` - tuple, point in Z^d. If None, pt=zero is considered.

EXAMPLES:

```
sage: from slabbe import BondPercolationSample
sage: S = BondPercolationSample(0.3,2)
sage: S.cluster_cardinality()                # random
13
sage: S.cluster_cardinality_stop_at(1000)    # random
13
sage: S.cluster_cardinality_stop_at(100)     # random
13
sage: S.cluster_cardinality_stop_at(10)      # random
'>=10'
```

cluster_in_box(*m*, *pt=None*)

Return the cluster (as a list) in the primal box [-m,m]^d containing the point pt.

INPUT:

- `m` - integer
- `pt` - tuple, point in Z^d. If None, pt=zero is considered.

EXAMPLES:

```
sage: from slabbe import BondPercolationSample
sage: S = BondPercolationSample(0.3,2)
sage: S.cluster_in_box(2)         # random
[(-2, -2), (-2, -1), (-1, -2), (-1, -1), (-1, 0), (0, 0)]
```

edges_in_box(*m*)

Return an iterator over all edges in the primal box [-m,m]^d.

INPUT:

- `m` - integer

EXAMPLES:

```
sage: from slabbe import BondPercolationSample
sage: S = BondPercolationSample(1,2)
sage: for a in S.edges_in_box(1): print a
((-1, -1), (0, -1))
((-1, -1), (-1, 0))
((-1, 0), (0, 0))
((-1, 0), (-1, 1))
((0, -1), (1, -1))
((0, -1), (0, 0))
((0, 0), (1, 0))
((0, 0), (0, 1))
```

neighbor(*pt*, *d*)

Return the neighbors of the point pt in direction d.

INPUT:

- `pt` - tuple, point in Z^d
- `direction` - integer, possible values are 1, 2, ..., d and -1, -2, ..., -d.

EXAMPLES:

sage: from slabbe import BondPercolationSample sage: S = BondPercolationSample(0.5,2) sage: S.neighbor((2,3),1) (3, 3) sage: S.neighbor((2,3),2) (2, 4) sage: S.neighbor((2,3),-1) (1, 3) sage: S.neighbor((2,3),-2) (2, 2)

plot(*m*, *pointsize=100*, *thickness=3*, *axes=False*)

Return 2d graphics object contained in the primal box [-m,m]^d.

INPUT:

- `pointsize`, integer (default:`100`),
- `thickness`, integer (default:`3`),
- `axes`, bool (default:`False`),

EXAMPLES:

```
sage: from slabbe import BondPercolationSample
sage: S = BondPercolationSample(0.5,2)
sage: S.plot(2)           # optional long
```

It works in 3d!!:

```
sage: S = BondPercolationSample(0.5,3)
sage: S.plot(3, pointsize=10, thickness=1)     # optional long
Graphics3d Object
```

save_pdf(*m*)

EXAMPLES:

```
sage: from slabbe import BondPercolationSample
sage: BondPercolationSample(0.3, d=2).save_pdf(20)    # optional long
Creation du fichier tikz_sample_d2_p300_m20.tikz
Using template '/Users/slabbe/.tikz2pdf.tex'.
tikz2pdf: calling pdflatex...
tikz2pdf: Output written to 'tikz_sample_d2_p300_m20.pdf'.
```

tikz(*m*)

Return tikz code.

EXAMPLES:

```
sage: from slabbe import BondPercolationSample
sage: S = BondPercolationSample(0.5,2)
sage: S.tikz(2)
\begin{tikzpicture}
[inner sep=0pt,thick,
 reddot/.style={fill=red,draw=red,circle,minimum size=5pt}]
\clip (-2.4, -2.4) rectangle (2.4, 2.4);
\draw (..., ...) -- (..., ...);
...
\node[circle,fill=none,draw=red,minimum size=0.8cm,ultra thick,inner sep=0pt] at (0,0) {};
\node[above right] at (0,0) {$(0, 0)$};
\end{tikzpicture}
```

zero()

EXAMPLES:

```
sage: from slabbe import BondPercolationSample
sage: S = BondPercolationSample(0.5,3)
sage: S.zero()
(0, 0, 0)

sage: S = BondPercolationSample(0.5,5)
sage: S.zero()
(0, 0, 0, 0, 0)
```

class slabbe.bond_percolation.**BondPercolationSamples**(*p*, *d*, *n*)

Bases: sage.structure.sage_object.SageObject

Return a list of n BondPercolationSample of given parameter p and dimension d.

EXAMPLES:

```
sage: from slabbe import BondPercolationSamples
sage: BondPercolationSamples(0.2,2,3)
<class 'slabbe.bond_percolation.BondPercolationSamples'>
```

cluster_cardinality(*stop*)

Return the list of cardinality of the cluster for each sample or +Infinity if the size is larger than stop value.

INPUT:

- stop - integer

EXAMPLES:

```
sage: from slabbe import BondPercolationSamples
sage: S20 = BondPercolationSamples(p=0.2, d=2, n=20)
sage: S20.cluster_cardinality(100)       # random
[1, 4, 1, 2, 2, 6, 5, 1, 5, 9, 2, 2, 2, 1, 2, 4, 4, 3, 2, 1]

sage: d = 2
sage: n = 5
sage: for p in srange(0,1,0.1): print p,BondPercolationSamples(p,d,n).cluster_cardinality(100) # optional long
```

```
0.000000000000000 [1, 1, 1, 1, 1]
0.100000000000000 [5, 1, 2, 1, 2]
0.200000000000000 [3, 1, 4, 1, 1]
0.300000000000000 [11, 7, 2, 1, 3]
0.400000000000000 [3, 3, 35, 10, '>=100']
0.500000000000000 ['>=100', '>=100', '>=100', '>=100', 26]
0.600000000000000 ['>=100', '>=100', '>=100', '>=100', '>=100']
0.700000000000000 ['>=100', '>=100', '>=100', '>=100', '>=100']
0.800000000000000 ['>=100', '>=100', '>=100', '>=100', '>=100']
0.900000000000000 ['>=100', '>=100', '>=100', '>=100', '>=100']
```

ntimes_over_size(*stop*)

EXAMPLES:

```
sage: from slabbe import BondPercolationSamples
sage: S = BondPercolationSamples(0.2,2,20)
sage: S.ntimes_over_size(100)    # random
0
sage: S = BondPercolationSamples(0.4,2,20)
sage: S.ntimes_over_size(100)    # random
1
sage: S = BondPercolationSamples(0.5,2,20)
sage: S.ntimes_over_size(100)    # random
17
```

percolation_probability(*stop*)

class slabbe.bond_percolation.**PercolationProbability**(*d*, *n*, *stop*, *verbose=False*)

Bases: sage.structure.sage_object.SageObject

EXAMPLES:

```
sage: from slabbe import PercolationProbability
sage: f = PercolationProbability(d=2, n=10, stop=100)
sage: f
Percolation Probability $\theta(p)$
d = dimension = 2
n = # samples = 10
stop counting at = 100
```

return_plot(*interval=(0, 1)*, *adaptive_recursion=4*, *plot_points=4*, *adaptive_tolerance=0.1*)

Return a plot of percolation probability using basic sage plot settings.

INPUT:

- interval, default=(0,1)
- adaptive_recursion, default=0
- plot_points, default=10
- adaptive_tolerance default=0.10

EXAMPLES:

```
sage: from slabbe import PercolationProbability
sage: T = PercolationProbability(d=2, n=10, stop=100)
sage: T.return_plot()           # optional long
Graphics object consisting of 1 graphics primitive
```

slabbe.bond_percolation.**compute_percolation_probability**(*range_p*, *d*, *n*, *stop*)

EXAMPLES:

```
sage: from slabbe.bond_percolation import compute_percolation_probability
sage: compute_percolation_probability(srange(0,0.8,0.1), d=2, n=5, stop=100) # random
d = 2, n = number of samples = 5
```

```
stop counting at = 100
p=0.0000, Theta=0.000, if |C|< 100 then max|C|=1
p=0.1000, Theta=0.000, if |C|< 100 then max|C|=1
p=0.2000, Theta=0.000, if |C|< 100 then max|C|=5
p=0.3000, Theta=0.000, if |C|< 100 then max|C|=6
p=0.4000, Theta=0.000, if |C|< 100 then max|C|=31
p=0.5000, Theta=1.00, if |C|< 100 then max|C|=-Infinity
p=0.6000, Theta=1.00, if |C|< 100 then max|C|=-Infinity
p=0.7000, Theta=1.00, if |C|< 100 then max|C|=-Infinity

sage: range_p = srange(0,0.8,0.1)
sage: compute_percolation_probability(range_p, d=2, n=5, stop=100) # not tested
d = 2, n = number of samples = 5
stop counting at = 100
p=0.0000, Theta=0.000, if |C|< 100 then max|C|=1
p=0.1000, Theta=0.000, if |C|< 100 then max|C|=1
p=0.2000, Theta=0.000, if |C|< 100 then max|C|=5
p=0.3000, Theta=0.000, if |C|< 100 then max|C|=6
p=0.4000, Theta=0.000, if |C|< 100 then max|C|=31
p=0.5000, Theta=1.00, if |C|< 100 then max|C|=-Infinity
p=0.6000, Theta=1.00, if |C|< 100 then max|C|=-Infinity
p=0.7000, Theta=1.00, if |C|< 100 then max|C|=-Infinity

sage: range_p = srange(0.45,0.55,0.01)
sage: compute_percolation_probability(range_p, d=2, n=10, stop=1000) # not tested
d = 2, n = number of samples = 10
stop counting at = 1000
p=0.4500, Theta=0.000, if |C|< 1000 then max|C|=378
p=0.4600, Theta=0.000, if |C|< 1000 then max|C|=475
p=0.4700, Theta=0.000, if |C|< 1000 then max|C|=514
p=0.4800, Theta=0.100, if |C|< 1000 then max|C|=655
p=0.4900, Theta=0.700, if |C|< 1000 then max|C|=274
p=0.5000, Theta=0.700, if |C|< 1000 then max|C|=975
p=0.5100, Theta=0.700, if |C|< 1000 then max|C|=16
p=0.5200, Theta=0.700, if |C|< 1000 then max|C|=125
p=0.5300, Theta=0.900, if |C|< 1000 then max|C|=4
p=0.5400, Theta=0.700, if |C|< 1000 then max|C|=6

sage: range_p = srange(0.475,0.485,0.001)
sage: compute_percolation_probability(range_p, d=2, n=10, stop=1000) # not tested
d = 2, n = number of samples = 10
stop counting at = 1000
p=0.4750, Theta=0.200, if |C|< 1000 then max|C|=718
p=0.4760, Theta=0.200, if |C|< 1000 then max|C|=844
p=0.4770, Theta=0.200, if |C|< 1000 then max|C|=566
p=0.4780, Theta=0.500, if |C|< 1000 then max|C|=257
p=0.4790, Theta=0.200, if |C|< 1000 then max|C|=566
p=0.4800, Theta=0.300, if |C|< 1000 then max|C|=544
p=0.4810, Theta=0.300, if |C|< 1000 then max|C|=778
p=0.4820, Theta=0.500, if |C|< 1000 then max|C|=983
p=0.4830, Theta=0.300, if |C|< 1000 then max|C|=473
p=0.4840, Theta=0.500, if |C|< 1000 then max|C|=411

sage: range_p = srange(0.47,0.48,0.001)
sage: compute_percolation_probability(range_p, d=2, n=20, stop=2000)  # not tested
d = 2, n = number of samples = 20
stop counting at = 2000
p=0.4700, Theta=0.0500, if |C|< 2000 then max|C|=1666
p=0.4710, Theta=0.100, if |C|< 2000 then max|C|=1665
p=0.4720, Theta=0.000, if |C|< 2000 then max|C|=1798
p=0.4730, Theta=0.0500, if |C|< 2000 then max|C|=1717
p=0.4740, Theta=0.150, if |C|< 2000 then max|C|=1924
p=0.4750, Theta=0.150, if |C|< 2000 then max|C|=1893
p=0.4760, Theta=0.150, if |C|< 2000 then max|C|=1458
p=0.4770, Theta=0.150, if |C|< 2000 then max|C|=1573
p=0.4780, Theta=0.200, if |C|< 2000 then max|C|=1762
```

```
p=0.4790, Theta=0.250, if |C|< 2000 then max|C|=951
```

slabbe.bond_percolation.**percolation_graphics_array**(*range_p*, *d*, *m*, *ncols=3*)

EXAMPLES:

```
sage: from slabbe.bond_percolation import percolation_graphics_array
sage: percolation_graphics_array(srange(0.1,1,0.1), d=2, m=5)    # optional long
sage: P = percolation_graphics_array(srange(0.45,0.55,0.01), d=2, m=5)  # optional long
sage: P.save('array_p45_p55_m5.png')     # not tested
sage: P = percolation_graphics_array(srange(0.45,0.55,0.01), d=2, m=10) # optional long
sage: P.save('array_p45_p55_m10.png')    # not tested
```

3.3 Dyck Word in 3d

Generalisation of Dyck Word to Surface of cubes in the n x n x n cube above the plane x+y+z=2n.

EXAMPLES:

```
sage: from slabbe.dyck_3d import DyckBlocks3d
sage: L = [len(DyckBlocks3d(i)) for i in range(1, 7)]    # not tested
[1, 2, 9, 96, 2498, 161422]
```

```
sage: L = [1, 2, 9, 96, 2498, 161422]
sage: oeis.find_by_subsequence(L)                        # not tested internet
0: A115965: Number of planar subpartitions of size n pyramidal planar partition.
```

AUTHOR:

- Sébastien Labbé, 31 october 2014

slabbe.dyck_3d.**DyckBlocks3d**(*n*)

EXAMPLES:

```
sage: from slabbe.dyck_3d import DyckBlocks3d
sage: L = [len(DyckBlocks3d(i)) for i in range(1, 6)]
sage: L
[1, 2, 9, 96, 2498]
```

slabbe.dyck_3d.**Possible**(*n*)

Possible stack of DyckWords inside a n x n cube.

EXAMPLES:

```
sage: from slabbe.dyck_3d import Possible
sage: Possible(1)
The cartesian product of ({[1, 0]},)
sage: Possible(2)
The cartesian product of ({[1, 1, 0, 0]}, {[1, 0, 1, 0], [1, 1, 0, 0]})
sage: Possible(3).list()
[([1, 1, 1, 0, 0, 0], [1, 1, 0, 1, 0, 0], [1, 0, 1, 0, 1, 0]),
 ([1, 1, 1, 0, 0, 0], [1, 1, 0, 1, 0, 0], [1, 0, 1, 1, 0, 0]),
 ([1, 1, 1, 0, 0, 0], [1, 1, 0, 1, 1, 0], [1, 1, 0, 0, 1, 0]),
 ([1, 1, 1, 0, 0, 0], [1, 1, 0, 1, 1, 0], [1, 1, 0, 1, 0, 0]),
 ([1, 1, 1, 0, 0, 0], [1, 1, 0, 1, 0, 0], [1, 1, 1, 0, 0, 0]),
 ([1, 1, 1, 0, 0, 0], [1, 1, 1, 0, 0, 0], [1, 0, 1, 0, 1, 0]),
 ([1, 1, 1, 0, 0, 0], [1, 1, 1, 0, 0, 0], [1, 0, 1, 1, 0, 0]),
 ([1, 1, 1, 0, 0, 0], [1, 1, 1, 0, 0, 0], [1, 1, 0, 0, 1, 0]),
 ([1, 1, 1, 0, 0, 0], [1, 1, 1, 0, 0, 0], [1, 1, 0, 1, 0, 0]),
 ([1, 1, 1, 0, 0, 0], [1, 1, 1, 0, 0, 0], [1, 1, 1, 0, 0, 0])]
```

slabbe.dyck_3d.**is_larger_than**(*x*, *y*)

EXAMPLES:

```
sage: from slabbe.dyck_3d import is_larger_than
sage: w = DyckWord([1,1,1,0,0,0])
sage: w.heights()
(0, 1, 2, 3, 2, 1, 0)
sage: z = DyckWord([1,0,1,0,1,0])
sage: is_larger_than(w,z)
True
sage: is_larger_than(z,w)
False
sage: is_larger_than(w,w)
True
```

CHAPTER

FOUR

DYNAMICAL SYSTEMS

4.1 Matrix Cocycles

EXAMPLES:

The 1-cylinders of ARP transformation given as matrices:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: ARP = cocycles.ARP()
sage: zip(*ARP.n_cylinders_iterator(1))
[(word: 1,
  word: 2,
  word: 3,
  word: 123,
  word: 132,
  word: 213,
  word: 231,
  word: 312,
  word: 321),
 (
[1 1 1]  [1 0 0]  [1 0 0]  [1 0 1]  [1 1 0]  [1 1 1]  [2 1 1]  [1 1 1]
[0 1 0]  [1 1 1]  [0 1 0]  [1 1 1]  [1 2 1]  [0 1 1]  [1 1 0]  [1 2 1]
[0 0 1], [0 0 1], [1 1 1], [1 1 2], [1 1 1], [1 1 2], [1 1 1], [0 1 1],

[2 1 1]
[1 1 1]
[1 0 1]
)]
```

Ces calculs illustrent le bounded distorsion de ratio=4 pour ARP multiplicatif (2 avril 2014):

```
sage: T = cocycles.Sorted_ARPMulti(2)
sage: T.distorsion_max(1, p=oo)
5
sage: T.distorsion_max(2, p=oo)
7
sage: T.distorsion_max(3, p=oo)
22/3
sage: T.distorsion_max(4, p=oo)    # long time (4s)
62/17


sage: T = cocycles.Sorted_ARPMulti(3)
sage: T.distorsion_max(1, p=oo)
7
sage: T.distorsion_max(2, p=oo)
9
sage: T.distorsion_max(3, p=oo)
19/2
sage: T.distorsion_max(4, p=oo)  # long time (47s)
161/43
```

Todo

- Fix the semi_norm_v bug using linear programming.

class slabbe.matrix_cocycle.**MatrixCocycle**(*gens*, *cone=None*, *language=None*)

Bases: object

Matrix cocycle

INPUT:

- gens – list, tuple or dict; the matrices. Keys 0,...,n-1 are used for list and tuple.
- cone – dict or matrix or None (default: None); the cone for each matrix generators. If it is a matrix, then it serves as the cone for all matrices. The cone is defined by the columns of the matrix. If None, then the cone is the identity matrix.
- language – regular language or None (default: None); if None, the language is the full shift.

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import MatrixCocycle
sage: B1 = matrix(3, [1,0,0, 0,1,0, 0,1,1])
sage: B2 = matrix(3, [1,0,0, 0,0,1, 0,1,1])
sage: B3 = matrix(3, [0,1,0, 0,0,1, 1,0,1])
sage: gens = {'1':B1, '2':B2, '3':B3}
sage: cone = matrix(3, [1,1,1,0,1,1,0,0,1])
sage: MatrixCocycle(gens, cone)
Cocycle with 3 gens over Language of finite words over alphabet ['1', '2', '3']
```

cone(*key*)

cone_dict()

distorsion_argmax(*n*, *p=1*)

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: ARP = cocycles.Sorted_ARP()
sage: ARP.distorsion_argmax(1)
(
          [1 0 0]
          [1 1 0]
word: A1, [3 2 1]
)
```

distorsion_max(*n*, *p=1*)

EXAMPLES:

Non borné:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: T = cocycles.Sorted_ARP()
sage: T.distorsion_max(1, p=oo)
1
sage: T.distorsion_max(2, p=oo)
3
sage: T.distorsion_max(3, p=oo)
5
sage: T.distorsion_max(4, p=oo)
7
```

gens()

identity_matrix()

EXAMPLES:

```
sage: class Foo:
....:     def __init__(self, x):
....:         self._x = x
....:     @cached_method
....:     def f(self):
....:         return self._x^2
sage: a = Foo(2)
sage: print a.f.get_cache()
None
sage: a.f()
4
sage: a.f.get_cache()
4
```

is_pisot(*w*)

language()

n_cylinders_edges(*n*)

Return the set of edges of the n-cylinders.

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: ARP = cocycles.ARP()
sage: ARP.n_cylinders_edges(1)
{frozenset({(1, 1, 0), (1, 1, 1)}),
 frozenset({(0, 1, 0), (1, 1, 0)}),
 frozenset({(1, 1, 1), (2, 1, 1)}),
 frozenset({(0, 0, 1), (1, 0, 1)}),
 frozenset({(0, 1, 0), (0, 1, 1)}),
 frozenset({(0, 1, 1), (1, 0, 1)}),
 frozenset({(1, 0, 0), (1, 1, 0)}),
 frozenset({(1, 1, 0), (2, 1, 1)}),
 frozenset({(1, 0, 1), (1, 1, 2)}),
 frozenset({(1, 1, 0), (1, 2, 1)}),
 frozenset({(1, 0, 1), (2, 1, 1)}),
 frozenset({(0, 0, 1), (0, 1, 1)}),
 frozenset({(1, 0, 1), (1, 1, 1)}),
 frozenset({(0, 1, 1), (1, 2, 1)}),
 frozenset({(0, 1, 1), (1, 1, 2)}),
 frozenset({(1, 0, 0), (1, 0, 1)}),
 frozenset({(1, 1, 1), (1, 2, 1)}),
 frozenset({(1, 0, 1), (1, 1, 0)}),
 frozenset({(0, 1, 1), (1, 1, 1)}),
 frozenset({(0, 1, 1), (1, 1, 0)}),
 frozenset({(1, 1, 1), (1, 1, 2)})}
```

n_cylinders_iterator(*n*)

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: C = cocycles.ARP()
sage: it = C.n_cylinders_iterator(1)
sage: for w,cyl in it: print "{}\n{}".format(w,cyl)
1
[1 1 1]
[0 1 0]
[0 0 1]
2
[1 0 0]
[1 1 1]
[0 0 1]
3
[1 0 0]
```

```
[0 1 0]
[1 1 1]
123
[1 0 1]
[1 1 1]
[1 1 2]
132
[1 1 0]
[1 2 1]
[1 1 1]
213
[1 1 1]
[0 1 1]
[1 1 2]
231
[2 1 1]
[1 1 0]
[1 1 1]
312
[1 1 1]
[1 2 1]
[0 1 1]
321
[2 1 1]
[1 1 1]
[1 0 1]
```

n_matrices_distorsion_iterator(*n*, *p=1*)

Return the the distorsion of the n-cylinders.

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: T = cocycles.Sorted_ARP()
sage: it =T.n_matrices_distorsion_iterator(1)
sage: list(it)
[(word: A1, 2),
 (word: A2, 2),
 (word: A3, 2),
 (word: P1, 3),
 (word: P2, 3),
 (word: P3, 3)]
```

n_matrices_eigenvalues_iterator(*n*)

Return the eigenvalues of the matrices of level n.

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: ARP = cocycles.ARP()
sage: list(ARP.n_matrices_eigenvalues_iterator(1))
[(word: 1, [1, 1, 1]),
 (word: 2, [1, 1, 1]),
 (word: 3, [1, 1, 1]),
 (word: 123, [1, 1, 1]),
 (word: 132, [1, 1, 1]),
 (word: 213, [1, 1, 1]),
 (word: 231, [1, 1, 1]),
 (word: 312, [1, 1, 1]),
 (word: 321, [1, 1, 1])]

sage: B = cocycles.Sorted_Brun()
sage: list(B.n_matrices_eigenvalues_iterator(1))
[(word: 1, [1, 1, 1]),
 (word: 2, [1, -0.618033988749895?, 1.618033988749895?]),
 (word: 3, [1.465571231876768?,
```

```
-0.2327856159383841? - 0.7925519925154479?*I,
-0.2327856159383841? + 0.7925519925154479?*I])]
```

n_matrices_eigenvectors(*n*, *verbose=False*)

Return the left and right eigenvectors of the matrices of level n.

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: C = cocycles.ARP()
sage: C.n_matrices_eigenvectors(1)
[(word: 1, (1.0, 0.0, 0.0), (0.0, 0.0, 1.0)),
 (word: 2, (0.0, 1.0, 0.0), (1.0, 0.0, 0.0)),
 (word: 3, (0.0, 0.0, 1.0), (1.0, 0.0, 0.0)),
 (word: 123, (0.0, 0.0, 1.0), (1.0, 0.0, 0.0)),
 (word: 132, (0.0, 1.0, 0.0), (1.0, 0.0, 0.0)),
 (word: 213, (0.0, 0.0, 1.0), (0.0, 1.0, 0.0)),
 (word: 231, (1.0, 0.0, 0.0), (0.0, 1.0, 0.0)),
 (word: 312, (0.0, 1.0, 0.0), (0.0, 0.0, 1.0)),
 (word: 321, (1.0, 0.0, 0.0), (0.0, 0.0, 1.0))]
```

n_matrices_iterator(*n*)

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: ARP = cocycles.Sorted_ARP()
sage: A,B = zip(*list(ARP.n_matrices_iterator(1)))
sage: A
(word: A1, word: A2, word: A3, word: P1, word: P2, word: P3)
sage: B
(
[1 0 0]  [1 0 0]  [0 1 0]  [0 1 0]  [0 0 1]  [0 0 1]
[0 1 0]  [0 0 1]  [0 0 1]  [0 1 1]  [1 0 1]  [0 1 1]
[1 1 1], [1 1 1], [1 1 1], [1 1 1], [1 1 1], [1 1 1]
)
```

n_matrices_non_pisot(*n*, *verbose=False*)

Return the list of non pisot matrices (as list of indices of base matrices).

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: ARP = cocycles.Sorted_ARP()
sage: ARP.n_matrices_non_pisot(1)
[word: A1, word: A2]
sage: ARP.n_matrices_non_pisot(2)   # long time (1s)
[word: A1,A1, word: A1,A2, word: A2,A1, word: A2,A2]
sage: ARP.n_matrices_non_pisot(3)   # long time (11s)
[word: A1,A1,A1,
 word: A1,A1,A2,
 word: A1,A2,A1,
 word: A1,A2,A2,
 word: A2,A1,A1,
 word: A2,A1,A2,
 word: A2,A2,A1,
 word: A2,A2,A2]
sage: len(ARP.n_matrices_non_pisot(4))  # long time
16
```

```
sage: from slabbe.matrix_cocycle import cocycles
sage: B = cocycles.Sorted_Brun()
sage: B.n_matrices_non_pisot(2)
[word: 11, word: 12, word: 21, word: 22]
sage: B.n_matrices_non_pisot(3)
[word: 111,
 word: 112,
```

```
 word: 121,
 word: 122,
 word: 211,
 word: 212,
 word: 221,
 word: 222]
```

n_matrices_semi_norm_iterator(*n*, *p=2*)

EXAMPLES:

For the 1-norm, all matrices contracts the hyperplane:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: C = cocycles.ARP()
sage: it = C.n_matrices_semi_norm_iterator(1, p=1)
sage: for _ in range(5): print next(it) # tolerance 0.0001
(word: 1, 1.0, False)
(word: 2, 1.0, False)
(word: 3, 1.0, False)
(word: 123, 0.9999885582839877, False)
(word: 132, 0.9999854006354785, False)
```

For the 2-norm, AR matrices do not contract:

```
sage: it = C.n_matrices_semi_norm_iterator(1, p=2)
sage: for w,s,b in it: print w,s,b   # long time (6s)
A1 1.30656296488 False
A2 1.30656296486 False
A3 1.30656296475 False
P12 0.99999999996 False
P13 0.999999999967 False
P21 0.999999999967 False
P23 0.999999999997 False
P31 0.999999999769 False
P32 0.999999999839 False
```

When, the 1-norm is < 1, the product is pisot:

```
sage: it = C.n_matrices_semi_norm_iterator(2, p=1)
sage: for w,s,b in it: print w,s,b   # long time
A1,A1 1.0 False
A1,A2 1.0 False
A1,A3 1.0 False
A1,P12 0.999998922557 False
A1,P13 0.999997464905 False
A1,P21 0.999993244882 False
A1,P23 0.999999150973 True
A1,P31 0.999994030522 False
A1,P32 0.999998046513 True
A2,A1 1.0 False
A2,A2 1.0 False
A2,A3 1.0 False
A2,P12 0.99999375291 False
A2,P13 0.999995591588 True
...
P31,A3 0.999988326888 False
P31,P12 0.749998931902 True
P31,P23 0.799999157344 True
P31,P32 0.749993104833 True
P32,A1 0.999997170005 True
P32,A3 0.99999420509 False
P32,P13 0.666665046248 True
P32,P21 0.666665629351 True
P32,P31 0.666664488371 True
```

n_words_iterator(*n*)

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: ARP = cocycles.Sorted_ARP()
sage: list(ARP.n_words_iterator(1))
[word: A1, word: A2, word: A3, word: P1, word: P2, word: P3]
```

non_pisot_automaton(*n*)

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: C = cocycles.ARP()
sage: A = C.non_pisot_automaton(2)
sage: A
Automaton with 2 states
sage: A.graph().plot(edge_labels=True)   # not tested
```

plot_n_cylinders(*n*, *labels=True*)

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: C = cocycles.Sorted_ARP()
sage: G = C.plot_n_cylinders(3)
```

plot_n_matrices_eigenvectors(*n*, *side='right'*, *color_index=0*, *draw_line=False*)

INPUT:

- `n` – integer, length
- `side` – `'left'` or `'right'`, drawing left or right eigenvectors
- `color_index` – 0 for first letter, -1 for last letter
- `draw_line` – boolean

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: ARP = cocycles.ARP()
sage: G = ARP.plot_n_matrices_eigenvectors(2)
```

plot_pisot_conjugates(*n*)

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: B = cocycles.Sorted_Brun()
sage: G = B.plot_pisot_conjugates(5)   # long time (8s)
```

Image envoyee a Timo (6 mai 2014):

```
sage: G = sum(B.plot_pisot_conjugates(i) for i in [1..6])  #not tested
```

tikz_n_cylinders(*n*, *labels=None*, *scale=1*)

INPUT:

- `labels` – None, True or False (default: None), if None, it takes value True if n is 1.

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: ARP = cocycles.ARP()
sage: t = ARP.tikz_n_cylinders(1, labels=True, scale=4)
sage: t
\documentclass[tikz]{standalone}
\begin{document}
\begin{tikzpicture}
```

```
[scale=4]
\draw (0.0000, -0.5000) -- (0.0000, 0.0000);
\draw (0.0000, -0.5000) -- (0.8660, -0.5000);
\draw (0.0000, 0.0000) -- (-0.2165, -0.1250);
...
... 23 lines not printed (1317 characters in total) ...
...
\node at (-0.1443, 0.1667) {$213$};
\node at (-0.2165, 0.0417) {$231$};
\node at (0.0722, -0.2083) {$312$};
\node at (-0.0722, -0.2083) {$321$};
\end{tikzpicture}
\end{document}
```

```
sage: from sage.misc.temporary_file import tmp_filename
sage: filename = tmp_filename('temp','.pdf')
sage: _ = t.pdf(filename)
```

word_to_matrix(*w*)

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: C = cocycles.Sorted_ARP()
sage: C.word_to_matrix(Word())
[1 0 0]
[0 1 0]
[0 0 1]
```

class slabbe.matrix_cocycle.**MatrixCocycleGenerator**

Bases: object

ARP()

ArnouxRauzy()

Brun()

Cassaigne()

FullySubtractive()

Poincare()

Reverse()

Selmer()

Sorted_ARP()

Sorted_ARPMulti(*order=3*)

Sorted_Brun()

slabbe.matrix_cocycle.**arp_polyhedron**(*d=3*)

Return the d-dimensional 1-cylinders of the ARP algorithm.

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import arp_polyhedron
sage: A,P,L = arp_polyhedron(3)
sage: A.vertices_list()
[[0, 0, 0], [1/2, 1/2, 0], [1/2, 1/4, 1/4], [1, 0, 0]]
sage: P.vertices_list()
[[0, 0, 0], [1/2, 1/2, 0], [1/2, 1/4, 1/4], [1/3, 1/3, 1/3]]
```

```
sage: A,P,L = arp_polyhedron(4)
sage: A.vertices_list()
[[0, 0, 0, 0],
 [1/2, 1/2, 0, 0],
 [1/2, 1/6, 1/6, 1/6],
 [1/2, 1/4, 1/4, 0],
 [1, 0, 0, 0]]
sage: P.vertices_list()
[[0, 0, 0, 0],
 [1/2, 1/2, 0, 0],
 [1/2, 1/4, 1/4, 0],
 [1/2, 1/6, 1/6, 1/6],
 [1/4, 1/4, 1/4, 1/4],
 [1/3, 1/3, 1/3, 0]]

sage: A,P,L = arp_polyhedron(5)
sage: A.vertices_list()
[[0, 0, 0, 0, 0],
 [1/2, 1/2, 0, 0, 0],
 [1/2, 1/8, 1/8, 1/8, 1/8],
 [1/2, 1/6, 1/6, 1/6, 0],
 [1/2, 1/4, 1/4, 0, 0],
 [1, 0, 0, 0, 0]]
sage: P.vertices_list()
[[0, 0, 0, 0, 0],
 [1/2, 1/2, 0, 0, 0],
 [1/2, 1/6, 1/6, 1/6, 0],
 [1/2, 1/8, 1/8, 1/8, 1/8],
 [1/2, 1/4, 1/4, 0, 0],
 [1/3, 1/3, 1/3, 0, 0],
 [1/5, 1/5, 1/5, 1/5, 1/5],
 [1/4, 1/4, 1/4, 1/4, 0]]
```

slabbe.matrix_cocycle.**cassaigne_polyhedron**(*d=3*)

Return the d-dimensional 1-cylinders of the Cassaigne algorithm.

(of the dual!)

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cassaigne_polyhedron
sage: L,La,Lb = cassaigne_polyhedron(3)
sage: L.vertices_list()
[[0, 0, 0], [0, 1/2, 1/2], [1/3, 1/3, 1/3], [1/2, 1/2, 0]]
sage: La.vertices_list()
[[0, 0, 0], [0, 1/2, 1/2], [1/3, 1/3, 1/3], [1/4, 1/2, 1/4]]
sage: Lb.vertices_list()
[[0, 0, 0], [1/3, 1/3, 1/3], [1/2, 1/2, 0], [1/4, 1/2, 1/4]]

sage: L,La,Lb = cassaigne_polyhedron(4)
sage: L.vertices_list()
[[0, 0, 0, 0],
 [0, 1/3, 1/3, 1/3],
 [1/3, 1/3, 1/3, 0],
 [1/4, 1/4, 1/4, 1/4],
 [1/5, 2/5, 1/5, 1/5],
 [1/5, 1/5, 2/5, 1/5]]

sage: L,La,Lb = cassaigne_polyhedron(5)
sage: L.vertices_list()
[[0, 0, 0, 0, 0],
 [0, 1/4, 1/4, 1/4, 1/4],
 [1/4, 1/4, 1/4, 1/4, 0],
 [1/6, 1/6, 1/3, 1/6, 1/6],
 [1/5, 1/5, 1/5, 1/5, 1/5],
 [1/6, 1/3, 1/6, 1/6, 1/6],
```

```
 [1/7, 2/7, 2/7, 1/7, 1/7],
 [1/7, 2/7, 1/7, 2/7, 1/7],
 [1/7, 1/7, 2/7, 2/7, 1/7],
 [1/6, 1/6, 1/6, 1/3, 1/6]]
```

slabbe.matrix_cocycle.**distorsion**(*M*, *p=1*)

1 Avril 2014. L'ancien ratio n'était pas le bon. Je n'utilisais pas les bonnes normes.

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import distorsion
sage: M = matrix(3, (1,2,3,4,5,6,7,8,9))
sage: M
[1 2 3]
[4 5 6]
[7 8 9]
sage: distorsion(M)
3/2
sage: (3+6+9) / (1+4+7)
3/2
sage: distorsion(M, p=oo)
9/7
```

slabbe.matrix_cocycle.**is_pisot**(*m*)

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import is_pisot
sage: M = matrix(3, (1,2,3,4,5,6,7,8,9))
sage: is_pisot(M)
False
```

slabbe.matrix_cocycle.**perron_right_eigenvector**(*M*)

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import perron_right_eigenvector
sage: m = matrix(2,[-11,14,-26,29])
sage: perron_right_eigenvector(m)           # tolerance 0.00001
(15.0000000000000, (0.35, 0.6499999999999999))
```

slabbe.matrix_cocycle.**projection_matrix**(*dim_from=3*, *dim_to=2*)

Return a projection matrix from R^d to R^l.

INPUT:

- dim_from‘ -- integer (default: ‘‘3)
- dim_to‘ -- integer (default: ‘‘2)

OUTPUT:

matrix

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import projection_matrix
sage: projection_matrix(3,2)
[-0.866025403784439  0.866025403784439  0.000000000000000]
[-0.500000000000000 -0.500000000000000   1.00000000000000]
sage: projection_matrix(2,3)
[-0.577350269189626 -0.333333333333333]
[ 0.577350269189626 -0.333333333333333]
[ 0.000000000000000  0.666666666666667]
```

slabbe.matrix_cocycle.**rounded_string_vector**(*v*, *digits=4*)

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import rounded_string_vector
sage: v = (-0.144337567297406, 0.166666666666667)
sage: rounded_string_vector(v)
'(-0.1443, 0.1667)'
sage: rounded_string_vector(v, digits=6)
'(-0.144338, 0.166667)'
```

slabbe.matrix_cocycle.**semi_norm_cone**(*M*, *cone*, *p=2*, *verbose=False*)

Return the semi norm on the hyperplane orthogonal to v where v lives in the cone.

EXAMPLES:

For Arnoux-Rauzy, only the 1-norm works:

```
sage: from slabbe.matrix_cocycle import semi_norm_cone
sage: A1 = matrix(3, [1,1,1, 0,1,0, 0,0,1])
sage: cone = A1
sage: semi_norm_cone(A1.transpose(), cone, p=1)    # tolerance 0.00001
0.9999999999999998
sage: semi_norm_cone(A1.transpose(), cone, p=oo)   # tolerance 0.0001
1.9999757223144654
sage: semi_norm_cone(A1.transpose(), cone, p=2)   # tolerance 0.00001
1.3065629648763757
```

For Poincaré, all norms work:

```
sage: P21 = matrix(3, [1,1,1, 0,1,1, 0,0,1])
sage: H21 = matrix(3, [1,0,0, 0,1,0, 1,0,1])
sage: cone = P21 * H21
sage: semi_norm_cone(P21.transpose(), cone, p=1)   # tolerance 0.00001
0.9999957276014074
sage: semi_norm_cone(P21.transpose(), cone, p=oo)   # tolerance 0.00001
1.0
sage: semi_norm_cone(P21.transpose(), cone, p=2)   # tolerance 0.00001
0.9999999999670175
```

For Poincaré on the whole cone, it works for some norms:

```
sage: P21 = matrix(3, [1,1,1, 0,1,1, 0,0,1])
sage: cone = P21
sage: semi_norm_cone(P21.transpose(), cone, p=1)   # tolerance 0.0001
1.9999675644077723
sage: semi_norm_cone(P21.transpose(), cone, p=2)   # tolerance 0.00001
1.6180339887021953
sage: semi_norm_cone(P21.transpose(), cone, p=oo)   # tolerance 0.00001
1.0
```

For a product, all norms work:

```
sage: A1 = matrix(3, [1,1,1, 0,1,0, 0,0,1])
sage: P21 = matrix(3, [1,1,1, 0,1,1, 0,0,1])
sage: H21 = matrix(3, [1,0,0, 0,1,0, 1,0,1])
sage: M = A1 * P21
sage: cone = A1 * P21 * H21
sage: semi_norm_cone(M.transpose(), cone, p=1)   # tolerance 0.00001
0.999993244882415
sage: semi_norm_cone(M.transpose(), cone, p=oo)   # tolerance 0.00001
0.9999935206958908
sage: semi_norm_cone(M.transpose(), cone, p=2)   # tolerance 0.00001
0.7529377601317161
```

slabbe.matrix_cocycle.**semi_norm_v**(*M*, *v*, *p=2*, *verbose=False*)

Return the semi norm on the hyperplane orthogonal to v.

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import semi_norm_v
sage: A1 = matrix(3, [1,-1,-1, 0,1,0, 0,0,1]).inverse()
sage: semi_norm_v(A1, vector( (1,1,1)))      # tolerance 0.0001
0.9999999999890247
sage: semi_norm_v(A1, vector( (1,1,1)), p=1)   # tolerance 0.0001
0.9999394820959548
sage: semi_norm_v(A1, vector( (1,1,1)), p=oo)   # tolerance 0.0001
1.0
```

4.2 Multidimensional Continued Fraction Algorithms

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Brun
sage: algo = Brun()
```

Orbit in the cone (with dual coordinates):

```
sage: algo.cone_orbit_list((10,23,15), 6)
[(10.0, 8.0, 15.0, 1.0, 1.0, 2.0, 132),
 (10.0, 8.0, 5.0, 3.0, 1.0, 2.0, 213),
 (2.0, 8.0, 5.0, 3.0, 4.0, 2.0, 321),
 (2.0, 3.0, 5.0, 3.0, 4.0, 6.0, 132),
 (2.0, 3.0, 2.0, 3.0, 10.0, 6.0, 123),
 (2.0, 1.0, 2.0, 3.0, 10.0, 16.0, 132)]
```

Orbit in the simplex:

```
sage: algo.simplex_orbit_list((10,23,15), 3)
[(0.30303030303030304,
  0.24242424242424246,
  0.45454545454545453,
  0.25,
  0.25,
  0.5,
  132),
 (0.43478260869565216,
  0.3478260869565218,
  0.21739130434782603,
  0.5,
  0.16666666666666666,
  0.3333333333333333,
  213),
 (0.13333333333333328,
  0.5333333333333334,
  0.3333333333333333,
  0.33333333333333337,
  0.4444444444444445,
  0.22222222222222224,
  321)]
```

Drawing the natural extension:

```
sage: fig = algo.natural_extension_plot(3000, norm_xyz='1', axis_off=True)
sage: fig
<matplotlib.figure.Figure object at ...>
sage: fig.savefig('a.png')  # not tested
```

Drawing the invariant measure:

```
sage: fig = algo.invariant_measure_wireframe_plot(10^6, 50)
sage: fig
<matplotlib.figure.Figure object at ...>
sage: fig.savefig('a.png')  # not tested
```

Word with given frequencies:

```
sage: algo.s_adic_word((1,e,pi))
word: 123232312323323123233231232312323231232 3...
```

Construction of the same s-adic word from the substitutions and the coding iterator:

```
sage: from itertools import repeat
sage: D = algo.substitutions()
sage: it = algo.coding_iterator((1,e,pi))
sage: words.s_adic(it, repeat(1), D)
word: 123232312323323123233231232312323231232 3...
```

Todo

- Ajout les algo de reuteneaour, nogueira, autres?
- Allow 2d, 1d, 4d, algorithms
- utilise les vecteurs pour les plus grandes dimensions?
- Replace method `_natural_extention_dict` by `simplex_orbit_filtered_list`
- Use `simplex_orbit_filtered_list` for `invariant_measure` ?
- Essayer d'utiliser @staticmethod pour call pour que ARP puisse apeler Poincare. Without the cython Error: Cannot call a static method on an instance variable. https://groups.google.com/d/topic/sage-support/DRI_s31D8ks/discussion

---

Question:

- Comment factoriser le code sans utiliser les yield?
- Comment faire un appel de fonction rapide (pour factoriser le code)

AUTHORS:

- Sébastien Labbé, Invariant measures, Lyapounov exponents and natural extensions for a dozen of algorithms, October 2013.
- Sébastien Labbé, Cleaning the code, Fall 2015

class slabbe.mult_cont_frac.**ARP**

Bases: slabbe.mult_cont_frac.MCFAlgorithm

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import ARP
sage: algo = ARP()
sage: TestSuite(algo).run()
sage: algo._test_dual_substitution_definition()
sage: algo._test_coherence()
sage: algo._test_definition()
```

dual_substitutions(**args*, ***kwds*)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import ARP
sage: ARP().dual_substitutions()
{1: WordMorphism: 1->123, 2->2, 3->3,
```

```
 2: WordMorphism: 1->1, 2->231, 3->3,
 3: WordMorphism: 1->1, 2->2, 3->312,
 123: WordMorphism: 1->1, 2->21, 3->321,
 132: WordMorphism: 1->1, 2->231, 3->31,
 213: WordMorphism: 1->12, 2->2, 3->312,
 231: WordMorphism: 1->132, 2->2, 3->32,
 312: WordMorphism: 1->13, 2->213, 3->3,
 321: WordMorphism: 1->123, 2->23, 3->3}
```

name(**args*, ***kwds*)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import ARP
sage: ARP().name()
"Arnoux-Rauzy-Poincar\\'e"
```

substitutions(**args*, ***kwds*)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import ARP
sage: ARP().substitutions()
{1: WordMorphism: 1->1, 2->21, 3->31,
 2: WordMorphism: 1->12, 2->2, 3->32,
 3: WordMorphism: 1->13, 2->23, 3->3,
 123: WordMorphism: 1->123, 2->23, 3->3,
 132: WordMorphism: 1->132, 2->2, 3->32,
 213: WordMorphism: 1->13, 2->213, 3->3,
 231: WordMorphism: 1->1, 2->231, 3->31,
 312: WordMorphism: 1->12, 2->2, 3->312,
 321: WordMorphism: 1->1, 2->21, 3->321}
```

class slabbe.mult_cont_frac.**ArnouxRauzy**

Bases: slabbe.mult_cont_frac.MCFAlgorithm

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import ArnouxRauzy
sage: algo = ArnouxRauzy()
sage: TestSuite(algo).run()
sage: algo._test_dual_substitution_definition()
sage: algo._test_coherence()
sage: algo._test_definition()
```

dual_substitutions(**args*, ***kwds*)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import ArnouxRauzy
sage: ArnouxRauzy().dual_substitutions()
{1: WordMorphism: 1->123, 2->2, 3->3,
 2: WordMorphism: 1->1, 2->231, 3->3,
 3: WordMorphism: 1->1, 2->2, 3->312}
```

substitutions(**args*, ***kwds*)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import ArnouxRauzy
sage: ArnouxRauzy().substitutions()
{1: WordMorphism: 1->1, 2->21, 3->31,
 2: WordMorphism: 1->12, 2->2, 3->32,
 3: WordMorphism: 1->13, 2->23, 3->3}
```

class slabbe.mult_cont_frac.**Brun**

Bases: slabbe.mult_cont_frac.MCFAlgorithm

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Brun
sage: algo = Brun()
sage: TestSuite(algo).run()
sage: algo._test_dual_substitution_definition()
sage: algo._test_coherence()
sage: algo._test_definition()
```

dual_substitutions(**args*, ***kwds*)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Brun
sage: Brun().dual_substitutions()
{123: WordMorphism: 1->1, 2->2, 3->32,
 132: WordMorphism: 1->1, 2->23, 3->3,
 213: WordMorphism: 1->1, 2->2, 3->31,
 231: WordMorphism: 1->13, 2->2, 3->3,
 312: WordMorphism: 1->1, 2->21, 3->3,
 321: WordMorphism: 1->12, 2->2, 3->3}
```

substitutions(**args*, ***kwds*)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Brun
sage: Brun().substitutions()
{123: WordMorphism: 1->1, 2->23, 3->3,
 132: WordMorphism: 1->1, 2->2, 3->32,
 213: WordMorphism: 1->13, 2->2, 3->3,
 231: WordMorphism: 1->1, 2->2, 3->31,
 312: WordMorphism: 1->12, 2->2, 3->3,
 321: WordMorphism: 1->1, 2->21, 3->3}
```

class slabbe.mult_cont_frac.**Cassaigne**

Bases: slabbe.mult_cont_frac.MCFAlgorithm

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Cassaigne
sage: algo = Cassaigne()
sage: TestSuite(algo).run()
sage: algo._test_dual_substitution_definition()
sage: algo._test_coherence()
sage: algo._test_definition()
```

dual_substitutions(**args*, ***kwds*)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Cassaigne
sage: Cassaigne().dual_substitutions()
{1: WordMorphism: 1->12, 2->3, 3->2,
 2: WordMorphism: 1->2, 2->1, 3->23}
```

substitutions(**args*, ***kwds*)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Cassaigne
sage: Cassaigne().substitutions()
{1: WordMorphism: 1->1, 2->13, 3->2,
 2: WordMorphism: 1->2, 2->13, 3->3}
```

class slabbe.mult_cont_frac.**FullySubtractive**

Bases: slabbe.mult_cont_frac.MCFAlgorithm

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import FullySubtractive
sage: algo = FullySubtractive()
sage: TestSuite(algo).run()
sage: algo._test_dual_substitution_definition()
sage: algo._test_coherence()
sage: algo._test_definition()
```

dual_substitutions(**args*, ***kwds*)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import FullySubtractive
sage: FullySubtractive().dual_substitutions()
{1: WordMorphism: 1->1, 2->21, 3->31,
 2: WordMorphism: 1->12, 2->2, 3->32,
 3: WordMorphism: 1->13, 2->23, 3->3}
```

name(**args*, ***kwds*)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import FullySubtractive
sage: FullySubtractive().name()
'Fully Subtractive'
```

substitutions(**args*, ***kwds*)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import FullySubtractive
sage: FullySubtractive().substitutions()
{1: WordMorphism: 1->123, 2->2, 3->3,
 2: WordMorphism: 1->1, 2->231, 3->3,
 3: WordMorphism: 1->1, 2->2, 3->312}
```

class slabbe.mult_cont_frac.**JacobiPerron**

Bases: slabbe.mult_cont_frac.MCFAlgorithm

class slabbe.mult_cont_frac.**JacobiPerronAdditif**

Bases: slabbe.mult_cont_frac.MCFAlgorithm

class slabbe.mult_cont_frac.**JacobiPerronAdditifv2**

Bases: slabbe.mult_cont_frac.MCFAlgorithm

class slabbe.mult_cont_frac.**MCFAlgorithm**

Bases: object

branches(**args*, ***kwds*)

Returns the branches labels of the algorithm.

This method is an heuristic and should be implemented in the inherited classes.

EXAMPLES:

```
sage: import slabbe.mult_cont_frac as mcf
sage: mcf.Brun().branches()
{123, 132, 213, 231, 312, 321}
sage: mcf.ARP().branches()
{1, 2, 3, 123, 132, 213, 231, 312, 321}
```

class_name(**args*, ***kwds*)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Reverse, Brun
sage: Reverse().class_name()
'Reverse'
```

```
sage: Brun().class_name()
'Brun'
```

coding_iterator(**args*, ***kwds*)

INPUT:

- `start` – iterable of three real numbers

OUTPUT:

iterator

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import ARP
sage: it = ARP().coding_iterator((1,e,pi))
sage: [next(it) for _ in range(20)]
[123, 2, 1, 123, 1, 231, 3, 3, 3, 3, 123, 1, 1, 1, 231, 2, 321, 2, 3, 312]
```

cone_orbit_iterator(**args*, ***kwds*)

INPUT:

- `start` - initial vector (default: `None`), if None, then initial point is random

NOTE:

This iterator is 10x slower because of the yield statement. So avoid using this when writing fast code. Just copy paste the loop or use simplex_orbit_list or simplex_orbit_filtered_list method.

OUTPUT:

iterator

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Brun
sage: it = Brun().cone_orbit_iterator((13,17,29))
sage: for _ in range(10): next(it)
((13.0, 17.0, 12.0), (1.0, 2.0, 1.0), 123)
((13.0, 4.0, 12.0), (3.0, 2.0, 1.0), 312)
((1.0, 4.0, 12.0), (3.0, 2.0, 4.0), 231)
((1.0, 4.0, 8.0), (3.0, 6.0, 4.0), 123)
((1.0, 4.0, 4.0), (3.0, 10.0, 4.0), 123)
((1.0, 4.0, 0.0), (3.0, 14.0, 4.0), 123)
((1.0, 3.0, 0.0), (17.0, 14.0, 4.0), 312)
((1.0, 2.0, 0.0), (31.0, 14.0, 4.0), 312)
((1.0, 1.0, 0.0), (45.0, 14.0, 4.0), 312)
((1.0, 0.0, 0.0), (59.0, 14.0, 4.0), 312)
```

cone_orbit_list(**args*, ***kwds*)

INPUT:

- `start` - initial vector (default: `None`), if None, then initial point is random
- `n_iterations` - integer, number of iterations

OUTPUT:

list

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Brun
sage: L = Brun().cone_orbit_list((10, 21, 37), 20)
sage: L[-1]
(1.0, 0.0, 0.0, 68.0, 55.0, 658.0, 231)
```

Todo

Check for a fixed point stop then.

discrepancy_statistics(**args*, ***kwds*)

Return the discrepancy of words of given length.

INPUT:

- `length` – integer

OUTPUT:

dict

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Brun
sage: Brun().discrepancy_statistics(5)
{[1, 1, 3]: 6/5,
 [1, 2, 2]: 4/5,
 [1, 3, 1]: 4/5,
 [2, 1, 2]: 4/5,
 [2, 2, 1]: 4/5,
 [3, 1, 1]: 4/5}
```

dual_substitutions(**args*, ***kwds*)

This method must be implemented in the inherited classes.

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Brun
sage: Brun().dual_substitutions()
{123: WordMorphism: 1->1, 2->2, 3->32,
 132: WordMorphism: 1->1, 2->23, 3->3,
 213: WordMorphism: 1->1, 2->2, 3->31,
 231: WordMorphism: 1->13, 2->2, 3->3,
 312: WordMorphism: 1->1, 2->21, 3->3,
 321: WordMorphism: 1->12, 2->2, 3->3}
```

e_one_star_patch(**args*, ***kwds*)

Return the n-th iterated patch of normal vector v.

INPUT:

- `v` – vector, the normal vector
- `n` – integer

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import FullySubtractive
sage: FullySubtractive().e_one_star_patch((1,e,pi), 4)
Patch of 21 faces
```

invariant_measure_contour_plot(**args*, ***kwds*)

Return a matplotlib graph of the invariant measure.

INPUT:

- `n_iterations` - integer, number of iterations
- `ndvis` - integer, number of divisions per dimension
- `norm` – string (default: `'1'`), either `'sup'` or `'1'`, the norm used for the orbit points

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Reverse, Brun
sage: Reverse().invariant_measure_contour_plot(1000000, 80)
<matplotlib.figure.Figure object at ...>
sage: Brun().invariant_measure_contour_plot(1000000, 40, norm='1')
<matplotlib.figure.Figure object at ...>
```

invariant_measure_wireframe_plot(**args*, ***kwds*)

Return a matplotlib graph of the invariant measure.

INPUT:

- `n_iterations` - integer, number of iterations
- `ndvis` - integer, number of divisions per dimension
- `norm` – string (default: `'1'`), either `'sup'` or `'1'`, the norm used for the orbit points

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Reverse, Brun
sage: Reverse().invariant_measure_wireframe_plot(1000000, 80)
<matplotlib.figure.Figure object at ...>
sage: Brun().invariant_measure_wireframe_plot(1000000, 40, norm='1')
<matplotlib.figure.Figure object at ...>
```

lyapunov_exponents(**args*, ***kwds*)

Return the lyapunov exponents (theta1, theta2, 1-theta2/theta1)

See also the module `slabbe.lyapunov` for parallel computations.

INPUT:

- `start` - initial vector (default: `None`), if None, then initial point is random
- `n_iterations` – integer
- `verbose` – bool (default: `False`)

OUTPUT:

tuple of the first two liapounov exponents and the uniform approximation exponent:

(theta1, theta2, 1-theta2/theta1)

Note: the code of this method was translated from C to cython. The C version is from Vincent Delecroix.

EXAMPLES:

Some benchmarks (on my machine):

```
sage: from slabbe.mult_cont_frac import Brun
sage: Brun().lyapunov_exponents(n_iterations=1000000)  # 68.6 ms # tolerance 0.003
(0.3049429393152174, -0.1120652699014143, 1.367495867105725)
```

Cython code on liafa is as fast as C on my machine:

```
sage: Brun().lyapunov_exponents(n_iterations=67000000) # 3.71s # tolerance 0.001
(0.30452120021265766, -0.11212586210856369, 1.36820379674801734)
```

Cython code on my machine is almost as fast as C on my machine:

```
sage: Brun().lyapunov_exponents(n_iterations=67000000) # 4.58 s # tolerance 0.001
(0.30456433843239084, -0.1121770192467067, 1.36831961293987303)
```

matrix_cocycle(**args*, ***kwds*)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import ARP
sage: ARP().matrix_cocycle()
Cocycle with 9 gens over Regular language over [1, 2, 3,
123, 132, 213, 231, 312, 321]
defined by: Automaton with 7 states

sage: from slabbe.mult_cont_frac import Sorted_Brun
sage: Sorted_Brun().matrix_cocycle()
Cocycle with 3 gens over Language of finite words over alphabet
[1, 2, 3]

sage: from slabbe.mult_cont_frac import Brun
sage: Brun().matrix_cocycle()
Cocycle with 6 gens over Regular language over
[123, 132, 213, 231, 312, 321]
defined by: Automaton with 6 states
```

measure_evaluation(**args*, ***kwds*)

Return the measure of a box according to an orbit.

INPUT:

- `n_iterations` - integer, number of iterations
- `draw` – string (default: `'image_right'`), possible values are:
  - `'domain_left'` - use x and y ranges
  - `'domain_right'` - use u and v ranges
  - `'image_left'` - use x and y ranges
  - `'image_right'` - use u and v ranges
- `norm_xyz` – string (default: `'1'`), either `'sup'` or `'1'`, the norm used for the orbit points
- `norm_uvw` – string (default: `'1'`), either `'sup'` or `'1'`, the norm used for the dual orbit points
- `xrange` – tuple (default: `(-.866, .866)`), interval of values for x
- `yrange` – tuple (default: `(-.5, 1.)`), interval of values for y
- `urange` – tuple (default: `(-.866, .866)`), interval of values for u
- `vrange` – tuple (default: `(-.5, 1.)`), interval of values for v
- `ndivs` – int (default: `1024`), number of pixels
- `verbose` – string (default: `False`)

BENCHMARK:

```
sage: from slabbe.mult_cont_frac import ARP
sage: opt = dict(urange=(-.15,.25), vrange=(-.05,.05))
sage: ARP().measure_evaluation(10^8, draw='right', ndivs=100, **opt) # optional long
0.435...
sage: ARP().measure_evaluation(10^8, draw='right', ndivs=1000, **opt) # optional long
0.357...
sage: ARP().measure_evaluation(10^8, draw='right', ndivs=2000, **opt) # optional long
0.293...
sage: ARP().measure_evaluation(10^8, draw='right', ndivs=4000, **opt) # optional long
0.177...
```

n_matrix(**args*, ***kwds*)

Return the n-matrix associated to the direction v.

INPUT:

- `start` – iterable of three real numbers
- `n_iterations` - integer, number of iterations

OUTPUT:

matrix

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import ARP
sage: ARP().n_matrix((1,e,pi), 10)
[ 31  40   7]
[ 84 109  19]
[ 97 126  22]
```

name(**args*, ***kwds*)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Reverse, Brun
sage: Reverse().name()
'Reverse'
sage: Brun().name()
'Brun'
```

natural_extension_part_png(**args*, ***kwds*)

Return a png or some part of an orbit in the natural extension.

INPUT:

- `n_iterations` - integer, number of iterations
- `draw` – string (default: `'image_right'`), possible values are:
  - `'domain_left'` - use x and y ranges
  - `'domain_right'` - use u and v ranges
  - `'image_left'` - use x and y ranges
  - `'image_right'` - use u and v ranges
- `norm_xyz` – string (default: `'1'`), either `'sup'` or `'1'`, the norm used for the orbit points
- `norm_uvw` – string (default: `'1'`), either `'sup'` or `'1'`, the norm used for the dual orbit points
- `xrange` – tuple (default: `(-.866, .866)`), interval of values for x
- `yrange` – tuple (default: `(-.5, 1.)`), interval of values for y
- `urange` – tuple (default: `(-.866, .866)`), interval of values for u
- `vrange` – tuple (default: `(-.5, 1.)`), interval of values for v
- `color_dict` – dict (default: `None`), dict from branches int to color (as RGB tuples)
- `branch_order` – list (default: `None`), list of branches int
- `ndivs` – int (default: `1024`), number of pixels
- `verbose` – string (default: `False`)

BENCHMARK:

A minute (1min 13s) for a picture with 10^7 points:

```
sage: from slabbe.mult_cont_frac import ARP
sage: c = {}
sage: c[1] = c[2] = c[3] = [0,0,0]
sage: c[12] = c[13] = c[23] = c[21] = c[31] = c[32] = [255,0,0]
sage: b = [1,2,3,12,13,21,23,31,32]
sage: P = ARP().natural_extension_part_png(10^7, draw='image_right',  # not tested
....:   branch_order=b, color_dict=c, urange=(-.6,.6), vrange=(-.6,.6))  # not tested
```

Half a minute (27s) for a picture zoomed in the orbit of 10^8 points:

```
sage: P = ARP().natural_extension_part_png(10^8, draw='image_right', # not tested
....:   branch_order=b, color_dict=c, urange=(.2,.3), vrange=(.2,.3))   # not tested
```

EXAMPLES:

```
sage: c = {}
sage: c[1] = c[2] = c[3] = [0,0,0]
sage: c[123] = c[132] = c[231] = c[213] = c[312] = c[321] = [255,0,0]
sage: b = [1,2,3,123,132,213,231,312,321]
sage: opt = dict(urange=(-.6,.6), vrange=(-.6,.6), color_dict=c, branch_order=b)
sage: P = ARP().natural_extension_part_png(10^5, draw='domain_left', **opt)
sage: P = ARP().natural_extension_part_png(10^5, draw='domain_right', **opt)
sage: P = ARP().natural_extension_part_png(10^5, draw='image_left', **opt)
sage: P = ARP().natural_extension_part_png(10^5, draw='image_right', **opt)
sage: P.show() # not tested
```

natural_extension_part_tikz(**args*, ***kwds*)

Return a pgfplots or some part of an orbit in the natural extension.

INPUT:

- `n_iterations` - integer, number of iterations
- `part` - integer, taking value 0, 1, 2 or 3
- `norm_xyz` – string (default: `'1'`), either `'sup'` or `'1'`, the norm used for the orbit points
- `norm_uvw` – string (default: `'1'`), either `'sup'` or `'1'`, the norm used for the dual orbit points
- `marksize` – string (default: `'1pt'`), pgfplots mark size value
- `limit_nb_points` – None or integer (default: `None`), limit number of points per patch
- `verbose` – string (default: `False`)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import ARP
sage: s = ARP().natural_extension_part_tikz(1000, part=3)
sage: view(s, tightpage=True)    # not tested
```

also

```
sage: s = ARP().natural_extension_part_tikz(1000, part=3, marksize='.2pt', limit_nb_points=1200, verbose=True)
Taking ... points for key 1
Taking ... points for key 2
Taking ... points for key 3
Taking ... points for key 132
Taking ... points for key 321
Taking ... points for key 231
Taking ... points for key 213
Taking ... points for key 312
Taking ... points for key 123
```

natural_extension_plot(**args*, ***kwds*)

INPUT:

- n_iterations - integer, number of iterations
- norm_xyz – string (default: '1'), either 'sup' or '1', the norm used for the orbit points
- norm_uvw – string (default: '1'), either 'sup' or '1', the norm used for the dual orbit points
- axis_off - boolean (default: False),

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Sorted_ARP
sage: Sorted_ARP().natural_extension_plot(1000)
<matplotlib.figure.Figure object at ...>
```

natural_extension_tikz(**args*, ***kwds*)

INPUT:

- n_iterations – integer, number of iterations
- norm_xyz – string (default: '1'), either 'sup' or '1', the norm used for the orbit points
- norm_uvw – string (default: '1'), either 'sup' or '1', the norm used for the dual orbit points
- marksize – tikz marksize (default:0.2)
- legend_marksize – tikz legend marksize (default:2)
- group_size – string (default:"4 by 1")

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Brun
sage: s = Brun().natural_extension_tikz(1000)
sage: s
\documentclass[tikz]{standalone}
\usepackage{pgfplots}
\usetikzlibrary{pgfplots.groupplots}
\begin{document}
\begin{tikzpicture}[scale=.7]
\begin{groupplot}
[group style={group size=4 by 1},
height=7cm,width=8cm,
xmin=-1.1,xmax=1.1,ymin=-.6,ymax=1.20,
...
... 4184 lines not printed (177644 characters in total) ...
...
\draw[draw=none] (group c2r1.center) --
node {$\to$}     (group c3r1.center);
\draw[draw=none] (group c3r1.center) --
node {$\times$}  (group c4r1.center);
\end{tikzpicture}
\end{document}

sage: from sage.misc.temporary_file import tmp_filename
sage: filename = tmp_filename('temp','.pdf')
sage: _ = s.pdf(filename)
```

s_adic_word(**args*, ***kwds*)

Return the s-adic word obtained from application of the MCF algorithm on the vector v.

INPUT:

- v - initial vector (default: None), if None, then initial point is random
- n_iterations - integer (default: 100), number of iterations
- nth_letter - letter (default: 1)

OUTPUT:

word

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Brun, ARP, Cassaigne
sage: Brun().s_adic_word((.414578,.571324,.65513))
word: 123231231232312312331232312312331232312 3...
sage: Brun().s_adic_word((1,e,pi))
word: 123232312323323123233231232312323232312323...
sage: ARP().s_adic_word((1,e,pi))
word: 123232312323323123233231232312323232312323...
sage: Cassaigne().s_adic_word((1,e,pi))
word: 232321323232313232321323232132323132323 2...
```

On integer entries:

```
sage: w = Brun().s_adic_word((3,4,5))
sage: w
word: 123233123123
sage: w.abelian_vector()
[3, 4, 5]
```

When the gcd is not one:

```
sage: Brun().s_adic_word((2,4,10))
word: 1233323312333233
```

```
sage: from slabbe.mult_cont_frac import FullySubtractive
sage: FullySubtractive().s_adic_word((1,2,5))
Traceback (most recent call last):
...
ValueError: On input=(1, 2, 5), algorithm Fully Subtractive
loops on (1.0, 0.0, 3.0)
```

```
sage: from slabbe.mult_cont_frac import Reverse
sage: algo = Reverse()
sage: algo.s_adic_word((18,1,1))
word: 31111111111211111111
sage: _.abelian_vector()
[18, 1, 1]
```

```
sage: Reverse().s_adic_word((3,1,1))
Traceback (most recent call last):
...
ValueError: On input=(3, 1, 1), algorithm Reverse reaches non
integer entries (0.5, 0.5, 0.5)
```

TESTS:

```
sage: v = ARP().s_adic_word((1,e,pi))
sage: w = Brun().s_adic_word((1,e,pi))
sage: v.longest_common_prefix(w, 'finite').length()
212
```

simplex_orbit_filtered_list(**args*, ***kwds*)

Return a list of the orbit filtered to fit into a rectangle.

INPUT:

- `start` - initial vector (default: `None`), if None, then initial point is random
- `n_iterations` - integer, number of iterations
- `norm_xyz` – string (default: `'sup'`), either `'sup'` or `'1'`, the norm used for the orbit of points (x, y, z) of the algo

- norm_uvw – string (default: 'sup'), either 'sup' or '1' or 'hypersurfac', the norm used for the orbit of dual coordinates (u, v, w).
- xmin - double
- ymin - double
- umin - double
- vmin - double
- xmax - double
- ymax - double
- umax - double
- vmax - double
- ndvis - integer, number of divisions

OUTPUT:

list

BENCHMARK:

```
sage: from slabbe.mult_cont_frac import Brun
sage: %time D = Brun().simplex_orbit_filtered_list(10^6) # not tested
CPU times: user 366 ms, sys: 203 ms, total: 568 ms
Wall time: 570 ms
```

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Brun
sage: start=(.414578,.571324,.65513)
sage: D = Brun().simplex_orbit_filtered_list(start, 3)
sage: D      # random
[(0.3049590483124023,
  -0.36889249928767137,
  -0.21650635094610976,
  -0.125,
  312,
  312),
 (0.08651831333735083,
  -0.31784823591841554,
  -0.34641016151377557,
  -0.2,
  312,
  312),
 (-0.41045591033143647,
  -0.20171750067080554,
  -0.4330127018922195,
  -0.25000000000000006,
  312,
  231)]

sage: Brun().simplex_orbit_filtered_list(n_iterations=3, norm_xyz='1',ndivs=1000)
Traceback (most recent call last):
...
ValueError: when ndivs is specified, you must provide a value
for xmin, xmax, ymin, ymax, umin, umax, vmin and vmax

sage: Brun().simplex_orbit_filtered_list(n_iterations=7,  # random
....:       norm_xyz='1', ndivs=100,
....:       xmin=-.866, xmax=.866, ymin=-.5, ymax=1.,
....:       umin=-.866, umax=.866, vmin=-.5, vmax=1.)
[(30, 47, 50, 50, 132, 213),
 (15, 83, 33, 66, 213, 231),
```

```
 (18, 80, 38, 44, 231, 231),
 (22, 75, 41, 33, 231, 231),
 (30, 68, 43, 26, 231, 231),
 (44, 53, 44, 22, 231, 213),
 (41, 78, 24, 56, 213, 321)]
```

simplex_orbit_iterator(**args*, ***kwds*)

INPUT:

- `start` - initial vector (default: `None`), if None, then initial point is random
- `norm_xyz` – string (default: `'sup'`), either `'sup'` or `'1'`, the norm used for the orbit of points (x, y, z) of the algo
- `norm_uvw` – string (default: `'sup'`), either `'sup'` or `'1'` or `'hypersurfac'`, the norm used for the orbit of dual coordinates (u, v, w).

NOTE:

This iterator is 10x slower because of the yield statement. So avoid using this when writing fast code. Just copy paste the loop or use simplex_orbit_list or simplex_orbit_filtered_list method.

OUTPUT:

iterator

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Brun
sage: it = Brun().simplex_orbit_iterator((.414578,.571324,.65513))
sage: for _ in range(4): next(it)
((0.7256442929056017, 1.0, 0.14668734378391243),
 (0.25, 0.5, 0.25),
 123)
((1.0, 0.37808566783572695, 0.20214772612150184),
 (0.5, 0.3333333333333333, 0.16666666666666666),
 312)
((1.0, 0.6079385025908344, 0.32504111204194974),
 (0.3333333333333333, 0.5555555555555555, 0.1111111111111111),
 321)
((0.6449032192209051, 1.0, 0.534661171576946),
 (0.25, 0.6666666666666666, 0.08333333333333333),
 321)

sage: from slabbe.mult_cont_frac import Brun
sage: it = Brun().simplex_orbit_iterator((.414578,.571324,.65513), norm_xyz='1')
sage: for _ in range(4): next(it)
((0.3875618393056797, 0.5340934161472103, 0.07834474454711005),
 (0.25, 0.5, 0.25),
 123)
 ((0.6328179140018012, 0.23925938363378257, 0.12792270236441622),
 (0.5, 0.3333333333333333, 0.16666666666666666),
 312)
((0.5173360300491189, 0.3145084914443481, 0.16815547850653312),
 (0.3333333333333333, 0.5555555555555555, 0.1111111111111111),
 321)
((0.2958862889959549, 0.45880727553726447, 0.24530643546678058),
 (0.25, 0.6666666666666666, 0.08333333333333333),
 321)
```

simplex_orbit_list(**args*, ***kwds*)

INPUT:

- `start` - initial vector (default: `None`), if None, then initial point is random
- `n_iterations` - integer, number of iterations

- norm_xyz – string (default: 'sup'), either 'sup' or '1', the norm used for the orbit of points (x, y, z) of the algo
- norm_uvw – string (default: 'sup'), either 'sup' or '1' or 'hypersurfac', the norm used for the orbit of dual coordinates (u, v, w).

OUTPUT:

list

BENCHMARK:

It could be 10 times faster because 10^6 iterations can be done in about 60ms on this machine. But for drawing images, it does not matter to be 10 times slower:

```
sage: %time L = Brun().simplex_orbit_list(10^6)   # not tested
CPU times: user 376 ms, sys: 267 ms, total: 643 ms
Wall time: 660 ms
```

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Brun
sage: L = Brun().simplex_orbit_list(n_iterations=10^5)
sage: L[-1]    # random
(0.7307002153148079,
 1.0,
 0.31588474491578816,
 0.29055326655584235,
 0.4690741038784866,
 0.24037262956567113,
 321)
```

substitutions(**args*, ***kwds*)

This method must be implemented in the inherited classes.

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Brun
sage: Brun().substitutions()
{123: WordMorphism: 1->1, 2->23, 3->3,
 132: WordMorphism: 1->1, 2->2, 3->32,
 213: WordMorphism: 1->13, 2->2, 3->3,
 231: WordMorphism: 1->1, 2->2, 3->31,
 312: WordMorphism: 1->12, 2->2, 3->3,
 321: WordMorphism: 1->1, 2->21, 3->3}
```

class slabbe.mult_cont_frac.**Poincare**

Bases: slabbe.mult_cont_frac.MCFAlgorithm

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Poincare
sage: algo = Poincare()
sage: TestSuite(algo).run()
sage: algo._test_dual_substitution_definition()
sage: algo._test_coherence()
sage: algo._test_definition()
```

dual_substitutions(**args*, ***kwds*)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Poincare
sage: Poincare().dual_substitutions()
{123: WordMorphism: 1->1, 2->21, 3->321,
 132: WordMorphism: 1->1, 2->231, 3->31,
 213: WordMorphism: 1->12, 2->2, 3->312,
 231: WordMorphism: 1->132, 2->2, 3->32,
```

```
 312: WordMorphism: 1->13, 2->213, 3->3,
 321: WordMorphism: 1->123, 2->23, 3->3}
```

name(**args*, ***kwds*)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Poincare
sage: Poincare().name()
"Poincar\\'e"
```

substitutions(**args*, ***kwds*)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Poincare
sage: Poincare().substitutions()
{123: WordMorphism: 1->123, 2->23, 3->3,
 132: WordMorphism: 1->132, 2->2, 3->32,
 213: WordMorphism: 1->13, 2->213, 3->3,
 231: WordMorphism: 1->1, 2->231, 3->31,
 312: WordMorphism: 1->12, 2->2, 3->312,
 321: WordMorphism: 1->1, 2->21, 3->321}
```

class slabbe.mult_cont_frac.**Reverse**

Bases: slabbe.mult_cont_frac.MCFAlgorithm

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Reverse
sage: algo = Reverse()
sage: TestSuite(algo).run()
sage: algo._test_dual_substitution_definition()
sage: algo._test_coherence()
sage: algo._test_definition()
```

dual_substitutions(**args*, ***kwds*)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Reverse
sage: Reverse().dual_substitutions()
{1: WordMorphism: 1->123, 2->2, 3->3,
 2: WordMorphism: 1->1, 2->231, 3->3,
 3: WordMorphism: 1->1, 2->2, 3->312,
 4: WordMorphism: 1->23, 2->13, 3->12}
```

substitutions(**args*, ***kwds*)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Reverse
sage: Reverse().substitutions()
{1: WordMorphism: 1->1, 2->21, 3->31,
 2: WordMorphism: 1->12, 2->2, 3->32,
 3: WordMorphism: 1->13, 2->23, 3->3,
 4: WordMorphism: 1->23, 2->31, 3->12}
```

class slabbe.mult_cont_frac.**Selmer**

Bases: slabbe.mult_cont_frac.MCFAlgorithm

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Selmer
sage: algo = Selmer()
sage: TestSuite(algo).run()
sage: algo._test_dual_substitution_definition()
sage: algo._test_coherence()
sage: algo._test_definition()
```

dual_substitutions(**args*, ***kwds*)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Selmer
sage: Selmer().dual_substitutions()
{123: WordMorphism: 1->1, 2->2, 3->31,
 132: WordMorphism: 1->1, 2->21, 3->3,
 213: WordMorphism: 1->1, 2->2, 3->32,
 231: WordMorphism: 1->12, 2->2, 3->3,
 312: WordMorphism: 1->1, 2->23, 3->3,
 321: WordMorphism: 1->13, 2->2, 3->3}
```

substitutions(**args*, ***kwds*)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Selmer
sage: Selmer().substitutions()
{123: WordMorphism: 1->13, 2->2, 3->3,
 132: WordMorphism: 1->12, 2->2, 3->3,
 213: WordMorphism: 1->1, 2->23, 3->3,
 231: WordMorphism: 1->1, 2->21, 3->3,
 312: WordMorphism: 1->1, 2->2, 3->32,
 321: WordMorphism: 1->1, 2->2, 3->31}
```

class slabbe.mult_cont_frac.**Sorted_ARMonteil**

Bases: slabbe.mult_cont_frac.MCFAlgorithm

class slabbe.mult_cont_frac.**Sorted_ARP**

Bases: slabbe.mult_cont_frac.MCFAlgorithm

class slabbe.mult_cont_frac.**Sorted_ARPMulti**

Bases: slabbe.mult_cont_frac.MCFAlgorithm

class slabbe.mult_cont_frac.**Sorted_ARrevert**

Bases: slabbe.mult_cont_frac.MCFAlgorithm

class slabbe.mult_cont_frac.**Sorted_ARrevertMulti**

Bases: slabbe.mult_cont_frac.MCFAlgorithm

class slabbe.mult_cont_frac.**Sorted_ArnouxRauzy**

Bases: slabbe.mult_cont_frac.MCFAlgorithm

class slabbe.mult_cont_frac.**Sorted_ArnouxRauzyMulti**

Bases: slabbe.mult_cont_frac.MCFAlgorithm

class slabbe.mult_cont_frac.**Sorted_Brun**

Bases: slabbe.mult_cont_frac.MCFAlgorithm

class slabbe.mult_cont_frac.**Sorted_BrunMulti**

Bases: slabbe.mult_cont_frac.MCFAlgorithm

class slabbe.mult_cont_frac.**Sorted_Delaunay**

Bases: slabbe.mult_cont_frac.MCFAlgorithm

class slabbe.mult_cont_frac.**Sorted_Meester**

Bases: slabbe.mult_cont_frac.MCFAlgorithm

class slabbe.mult_cont_frac.**Sorted_Poincare**

Bases: slabbe.mult_cont_frac.MCFAlgorithm

class slabbe.mult_cont_frac.**Sorted_Selmer**

Bases: slabbe.mult_cont_frac.MCFAlgorithm

4.3 Lyapunov exponents (comparison)

slabbe.lyapunov.**lyapunov_comparison_table**(*L*, *n_orbits=100*, *n_iterations=10000*)

Return a table of values of Lyapunov exponents for many algorithm.

INPUT:

- L – list of algorithms
- n_orbits – integer
- n_iterations – integer

OUTPUT:

table

EXAMPLES:

```
sage: import slabbe.mult_cont_frac as mcf
sage: from slabbe.lyapunov import lyapunov_comparison_table
sage: algos = [mcf.Brun(), mcf.ARP()]
sage: lyapunov_comparison_table(algos)    # abs tol 0.01
  Algorithm                \#Orbits   $\theta_1$ (std)   $\theta_2$ (std)   $1-\theta_2/\theta_1$ (std)
+-------------------------+----------+------------------+------------------+----------------------------+
  Arnoux-Rauzy-Poincar\'e   100        0.44 (0.012)       -0.172 (0.0060)    1.388 (0.0054)
  Brun                      100        0.30 (0.011)       -0.113 (0.0049)    1.370 (0.0070)
```

slabbe.lyapunov.**lyapunov_sample**(*algo*, *n_orbits*, *n_iterations=1000*, *verbose=False*)

Return lists of values for theta1, theta2 and 1-theta2/theta1 computed on many orbits.

This is computed in parallel.

INPUT:

- n_orbits – integer, number of orbits
- n_iterations – integer, length of each orbit
- verbose – bool (default: False)

OUTPUT:

tuple of three lists

EXAMPLES:

```
sage: from slabbe.lyapunov import lyapunov_sample
sage: from slabbe.mult_cont_frac import Brun
sage: lyapunov_sample(Brun(), 5, 1000000) # abs tol 0.01
[(0.3027620661266397,
  0.3033468535021702,
  0.3044950176856005,
  0.3030531162480779,
  0.30601169862996064),
 (-0.11116236859835525,
  -0.11165563059874498,
  -0.1122595926203868,
  -0.11190323336181864,
  -0.11255687513610782),
 (1.367160820443926,
  1.3680790794750939,
  1.3686746452327765,
  1.3692528714016428,
  1.3678188632657973)]
```

slabbe.lyapunov.**lyapunov_table**(*algo*, *n_orbits*, *n_iterations=1000*)

Return a table of values of Lyapunov exponents for this algorithm.

INPUT:

- `n_orbits` – integer, number of orbits
- `n_iterations` – integer, length of each orbit

OUTPUT:

table of liapounov exponents

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Brun
sage: from slabbe.lyapunov import lyapunov_table
sage: lyapunov_table(Brun(), 10, 1000000) # random
  10 succesfull orbits    min       mean      max       std
+----------------------+---------+---------+---------+---------+
  $\theta_1$              0.303     0.305     0.307     0.0013
  $\theta_2$              -0.1131   -0.1124   -0.1115   0.00051
  $1-\theta_2/\theta_1$   1.3678    1.3687    1.3691    0.00043
```

CHAPTER

FIVE

MISCELLANEOUS

5.1 Tikz Picture

Conversions to pdf, png.

EXAMPLES:

```
sage: from slabbe import TikzPicture
sage: g = graphs.PetersenGraph()
sage: s = latex(g)
sage: t = TikzPicture(s, standalone_configs=["border=3mm"], packages=['tkz-graph'])
sage: _ = t.pdf()    # not tested
```

```
sage: t
\documentclass[tikz]{standalone}
\standaloneconfig{border=3mm}
\usepackage{tkz-graph}
\begin{document}
\begin{tikzpicture}
%
\useasboundingbox (0,0) rectangle (5.0cm,5.0cm);
%
\definecolor{cv0}{rgb}{0.0,0.0,0.0}
...
... 68 lines not printed (3748 characters in total) ...
...
\Edge[lw=0.1cm,style={color=cv6v8,},](v6)(v8)
\Edge[lw=0.1cm,style={color=cv6v9,},](v6)(v9)
\Edge[lw=0.1cm,style={color=cv7v9,},](v7)(v9)
%
\end{tikzpicture}
\end{document}
```

```
sage: V = [[1,0,1],[1,0,0],[1,1,0],[0,0,-1],[0,1,0],[-1,0,0],[0,1,1],[0,0,1],[0,-1,0]]
sage: P = Polyhedron(vertices=V).polar()
sage: s = P.projection().tikz([674,108,-731],112)
sage: t = TikzPicture(s)
sage: _ = t.pdf()    # not tested
```

class slabbe.tikz_picture.**TikzPicture**(*code*, *standalone_configs*=[ ], *packages*=[ ], *tikzlibraries*=[ ], *macros*=[ ])

Bases: sage.structure.sage_object.SageObject

INPUT:

- code – string, tikzpicture code
- standalone_configs – list of strings (default: []), standalone configuration options.

- packages – list of strings or 'sage_preamble' (default: []), latex packages. If 'sage_preamble', it is replaced by the sage latex preamble.
- tikzlibraries – list of strings (default: []), tikz libraries to use.
- macros – list of strings or 'sage_latex_macros' (default: []), stuff you need for the picture. If 'sage_latex_macros', it is replaced by the sage latex macros.

POSSIBLE OPTIONS:

Here are some standalone config you may want to use (see standalone documentation for more options):

```
standalone_configs = ['preview', 'border=3mm', 'varwidth', 'beamer',
'float']
```

Here are some packages you may want to load:

```
packages = ['nicefrac', 'amsmath', 'pifont', 'tikz-3dplot',
'tkz-graph', 'tkz-berge', 'pgfplots']
```

Here are some tikzlibraries you may want to load:

```
tikzlibraries = ['arrows', 'snakes', 'backgrounds', 'patterns',
'matrix', 'shapes', 'fit', 'calc', 'shadows', 'plotmarks',
'positioning', 'pgfplots.groupplots', 'mindmap']
```

Here are some macros you may want to set:

```
macros = [r'\newcommand{\ZZ}{\mathbb{Z}}']
```

EXAMPLES:

```
sage: from slabbe import TikzPicture
sage: g = graphs.PetersenGraph()
sage: s = latex(g)
sage: t = TikzPicture(s, standalone_configs=["border=3mm"], packages=['tkz-graph'])
sage: _ = t.pdf()    # not tested
```

pdf(*filename=None*, *view=True*)

Compiles the latex code with pdflatex and create a pdf file.

INPUT:

- filename – string (default:None), the output filename. If None, it saves the file in a temporary directory.
- view – bool (default:True), whether to open the file in a pdf viewer. This option is ignored if filename is not None.

OUTPUT:

string, path to pdf file

EXAMPLES:

```
sage: from slabbe import TikzPicture
sage: V = [[1,0,1],[1,0,0],[1,1,0],[0,0,-1],[0,1,0],[-1,0,0],[0,1,1],[0,0,1],[0,-1,0]]
sage: P = Polyhedron(vertices=V).polar()
sage: s = P.projection().tikz([674,108,-731],112)
sage: t = TikzPicture(s)
sage: _ = t.pdf()    # not tested

sage: from sage.misc.temporary_file import tmp_filename
sage: filename = tmp_filename('temp','.pdf')
sage: _ = t.pdf(filename)
```

png(*filename=None*, *density=150*, *view=True*)

Compiles the latex code with pdflatex and converts to a png file.

INPUT:

- `filename` – string (default:`None`), the output filename. If `None`, it saves the file in a temporary directory.
- `density` – integer, (default: `150`), horizontal and vertical density of the image
- `view` – bool (default:`True`), whether to open the file in a png viewer. This option is ignored if `filename` is not `None`.

OUTPUT:

string, path to png file

EXAMPLES:

```
sage: from slabbe import TikzPicture
sage: V = [[1,0,1],[1,0,0],[1,1,0],[0,0,-1],[0,1,0],[-1,0,0],[0,1,1],[0,0,1],[0,-1,0]]
sage: P = Polyhedron(vertices=V).polar()
sage: s = P.projection().tikz([674,108,-731],112)
sage: t = TikzPicture(s)
sage: _ = t.png()    # not tested

sage: from sage.misc.temporary_file import tmp_filename
sage: filename = tmp_filename('temp','.png')
sage: _ = t.png(filename)
```

5.2 Ranking Scale for Ultimate Frisbee

EXEMPLES:

```
sage: from slabbe.ranking_scale import *
sage: R = RankingScale_CQU4_2011()
sage: R = RankingScale_USAU_2013()
sage: R = RankingScale_CQU4_2014()
```

AUTHOR :

- Sébastien Labbé, Fall 2011, first version

class slabbe.ranking_scale.**RankingScale**(*scale_names*, *scales*)

Bases: `object`

INPUT:

- `scale_names` – iterable of str
- `scales` – iterable of list

NOTE: the ordering of both input must match

length()

EXEMPLES:

```
sage: from slabbe.ranking_scale import RankingScale_CQU4_2011
sage: R = RankingScale_CQU4_2011()
sage: R.length()
105
```

plot(*pointsize=20*)

pointage_csv(*filename='pointage.csv'*, *dialect='excel'*)

EXEMPLES:

```
sage: from slabbe.ranking_scale import RankingScale_CQU4_2011
sage: R = RankingScale_CQU4_2011()
sage: R.pointage_csv()          # not tested
Creation of file pointage.csv
```

table()

EXAMPLES:

```
sage: from slabbe.ranking_scale import RankingScale_CQU4_2011
sage: R = RankingScale_CQU4_2011()
sage: R.table()
  Position   Grand Chelem   Mars Attaque   La Flotte   Petit Chelem
  1          1000           1000           800         400
  2          938            938            728         355
  3          884            884            668         317
  4          835            835            614         285
  5          791            791            566         256
  6          750            750            522         230
  7          711            711            482         206
  8          675            675            444         184
  9          641            641            409         164
  10         609            609            377         146
...
  95         0              11             0           0
  96         0              10             0           0
  97         0              9              0           0
  98         0              8              0           0
  99         0              7              0           0
  100        0              6              0           0
  101        0              4              0           0
  102        0              3              0           0
  103        0              2              0           0
  104        0              1              0           0
```

slabbe.ranking_scale.**RankingScale_CQU4_2011**()

EXEMPLES:

```
sage: from slabbe.ranking_scale import RankingScale_CQU4_2011
sage: R = RankingScale_CQU4_2011()
sage: R.table()
  Position   Grand Chelem   Mars Attaque   La Flotte   Petit Chelem
  1          1000           1000           800         400
  2          938            938            728         355
  3          884            884            668         317
  4          835            835            614         285
  5          791            791            566         256
  6          750            750            522         230
  7          711            711            482         206
  8          675            675            444         184
  9          641            641            409         164
  10         609            609            377         146
...
  95         0              11             0           0
  96         0              10             0           0
  97         0              9              0           0
  98         0              8              0           0
  99         0              7              0           0
  100        0              6              0           0
  101        0              4              0           0
  102        0              3              0           0
  103        0              2              0           0
  104        0              1              0           0
```

slabbe.ranking_scale.**RankingScale_CQU4_2014**(*R=1*, *base=e*)

EXAMPLES:

```
sage: from slabbe.ranking_scale import RankingScale_CQU4_2014
sage: R = RankingScale_CQU4_2014()
```

slabbe.ranking_scale.**RankingScale_CQU4_2015_v1**()

EXAMPLES:

```
sage: from slabbe.ranking_scale import RankingScale_CQU4_2015_v1
sage: R = RankingScale_CQU4_2015_v1()
```

slabbe.ranking_scale.**RankingScale_CQU4_2015_v2**()

EXAMPLES:

```
sage: from slabbe.ranking_scale import RankingScale_CQU4_2015_v2
sage: R = RankingScale_CQU4_2015_v2()
```

slabbe.ranking_scale.**RankingScale_CQU4_2015_v3**()

EXAMPLES:

```
sage: from slabbe.ranking_scale import RankingScale_CQU4_2015_v3
sage: R = RankingScale_CQU4_2015_v3()
```

slabbe.ranking_scale.**RankingScale_CQU4_2015_v4**()

EXAMPLES:

```
sage: from slabbe.ranking_scale import RankingScale_CQU4_2015_v4
sage: R = RankingScale_CQU4_2015_v4()
```

slabbe.ranking_scale.**RankingScale_CQU4_2015_v5**()

EXAMPLES:

```
sage: from slabbe.ranking_scale import RankingScale_CQU4_2015_v5
sage: R = RankingScale_CQU4_2015_v5()
```

slabbe.ranking_scale.**RankingScale_USAU_2013**()

EXAMPLES:

```
sage: from slabbe.ranking_scale import RankingScale_USAU_2013
sage: R = RankingScale_USAU_2013()
sage: R.table()
  Position   Serie 1500   Serie 1000   Serie 500   Serie 250
  1          1500         1000         500         250
  2          1366         911          455         228
  3          1252         835          417         209
  4          1152         768          384         192
  5          1061         708          354         177
  6          979          653          326         163
  7          903          602          301         151
  8          832          555          278         139
  9          767          511          256         128
  10         706          471          236         118
  11         648          432          217         109
  12         595          397          199         100
  13         544          363          182         91
  14         497          332          166         83
  15         452          302          151         76
  16         410          274          137         69
  17         371          248          124         62
  18         334          223          112         56
  19         299          199          100         50
  20         266          177          89          45
  21         235          157          79          40
  22         206          137          69          35
  23         178          119          60          30
```

```
24          152          102          51          26
25          128          86           43          22
26          106          71           36          18
27          85           57           29          15
28          66           44           22          12
29          47           32           16          9
30          31           21           11          6
31          15           10           6           3
32          1            1            1           1
```

slabbe.ranking_scale.**curve**(*nb_equipes*, *max_points=100*, *K=1*, *R=2*, *base=2*, *verbose=False*)

INPUT:

- nb_equipes – integer
- max_points – integer
- K – the value at p = nb_equipes
- R – real (default: 2), curve parameter
- base – 2
- verbose - bool

EXEMPLES:

```
sage: from slabbe.ranking_scale import curve
sage: curve(20, 100)
-99*(p*(log(40) + 1) - p*log(p) - 20*log(40) + 20*log(20) -
20)/(19*log(40) - 20*log(20) + 19) + 1
sage: curve(64, 100)
-99*(p*(log(128) + 1) - p*log(p) - 64*log(128) + 64*log(64) -
64)/(63*log(128) - 64*log(64) + 63) + 1

sage: curve(64, 100)(p=64)
1
sage: curve(64, 100)(p=1)
100
sage: curve(64, 100)(p=2)
198*(31*log(128) - 32*log(64) + log(2) + 31)/(63*log(128) - 64*log(64) + 63) + 1
sage: n(curve(64, 100)(p=2))
95.6871477097753

sage: curve(64, 100, verbose=True)
fn = -(p*(log(128) + 1) - p*log(p) - 64*log(128) + 64*log(64) - 64)/log(2)
aire = 147.889787576005
fn normalise = -99*(p*(log(128) + 1) - p*log(p) - 64*log(128) +
64*log(64) - 64)/(63*log(128) - 64*log(64) + 63) + 1
-99*(p*(log(128) + 1) - p*log(p) - 64*log(128) + 64*log(64) -
64)/(63*log(128) - 64*log(64) + 63) + 1
```

The base argument seems to be useless (why?):

```
sage: curve(100,100,base=3)
-99*(p*(log(200) + 1) - p*log(p) - 100*log(200) + 100*log(100) -
100)/(99*log(200) - 100*log(100) + 99) + 1
sage: curve(100,100,base=2)
-99*(p*(log(200) + 1) - p*log(p) - 100*log(200) + 100*log(100) -
100)/(99*log(200) - 100*log(100) + 99) + 1
```

slabbe.ranking_scale.**discrete_curve**(*nb_equipes*, *max_points=100*, *K=1*, *R=2*, *base=2*, *verbose=False*)

INPUT:

- nb_equipes – integer

- max_points – integer
- K – the value at p = nb_equipes
- R – real (default: 2), curve parameter
- base – 2
- verbose - bool

EXEMPLES:

```
sage: from slabbe.ranking_scale import discrete_curve
sage: A = discrete_curve(64, 100,verbose=True)
First difference sequence is
[1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 2, 1,
2, 2, 2, 2, 2, 2, 2, 3, 2, 2, 3, 3, 2, 3, 3, 3, 4, 4, 4]

sage: A = discrete_curve(64, 100)
sage: B = discrete_curve(32, 50)
sage: C = discrete_curve(16, 25)

sage: A
[100, 96, 92, 88, 85, 82, 79, 77, 74, 71, 69, 67, 64, 62, 60, 58,
56, 54, 52, 50, 49, 47, 45, 44, 42, 41, 39, 38, 36, 35, 33, 32, 31,
29, 28, 27, 26, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12,
11, 10, 9, 8, 8, 7, 6, 5, 5, 4, 3, 2, 2, 1]
sage: B
[50, 46, 43, 40, 37, 35, 33, 31, 29, 27, 25, 23, 21, 20, 18, 17,
16, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 2, 1]
sage: C
[25, 22, 19, 17, 15, 13, 11, 9, 8, 7, 6, 5, 4, 3, 2, 1]

sage: A = discrete_curve(64+2, 100) #Movember, Bye Bye, Cdf, MA
sage: B = discrete_curve(32+2, 70)  # la flotte
sage: C = discrete_curve(16+2, 40) # october fest, funenuf, la viree
```

slabbe.ranking_scale.**discrete_curve_2**(*nb_equipes*, *max_points=100*)

slabbe.ranking_scale.**table_to_csv**(*self*, *filename*, *dialect='excel'*)

5.3 Fruit

This file is an example of Sage conventions for syntax and documentation. It also serves as an example to explain class inheritance: methods defined for fruits are automatically defined for bananas and strawberries.

AUTHORS:

- Sébastien Labbé, 2010-2013

EXAMPLES:

```
sage: from slabbe import Fruit
sage: f = Fruit(5)
sage: f
A fruit of 5 kilos.
sage: f.weight()
5
sage: f.is_a_fruit()
True
```

Because of class inheritance which says that a banana is a fruit and a strawberry is a fruit, the methods written for fruits are defined for bananas and strawberries automatically:

```
sage: from slabbe import Strawberry
sage: s = Strawberry(32)
sage: s
A strawberry of 32 kilos.
sage: s.weight()
32
sage: s.is_a_fruit()
True


sage: from slabbe import Banana
sage: b = Banana(13)
sage: b
A banana of 13 kilos.
sage: b.weight()
13
sage: b.is_a_fruit()
True


sage: s = Strawberry(32)
sage: t = Strawberry(7)
sage: s
A strawberry of 32 kilos.
sage: t
A strawberry of 7 kilos.
sage: s + t
A strawberry of 1073 kilos.
```

class slabbe.fruit.**Banana**(*weight=1*)

Bases: slabbe.fruit.Fruit

Creates a banana.

INPUT:

- `weight` - number, in kilos

OUTPUT:

Banana

EXAMPLES:

```
sage: from slabbe import Banana
sage: b = Banana(9)
sage: b
A banana of 9 kilos.
```

TESTS:

Testing that pickle works:

```
sage: loads(dumps(b))
A banana of 9 kilos.
sage: b == loads(dumps(b))
True
```

Running the test suite:

```
sage: TestSuite(b).run()
```

class slabbe.fruit.**Fruit**(*weight=1*)

Bases: sage.structure.sage_object.SageObject

Creates a fruit.

INPUT:

- weight - number, in kilos

OUTPUT:

Fruit

EXAMPLES:

```
sage: from slabbe import Fruit
sage: f = Fruit(5)
sage: f
A fruit of 5 kilos.
```

is_a_fruit()

Returns True if it is a fruit.

OUTPUT:

Boolean

EXAMPLES:

```
sage: from slabbe import Fruit
sage: f = Fruit(3)
sage: f.is_a_fruit()
True
```

weight()

Return the weight.

OUTPUT:

Number

EXAMPLES:

```
sage: from slabbe import Fruit
sage: f = Fruit(3)
sage: f.weight()
3
```

class slabbe.fruit.**Strawberry**(*weight=1*)

Bases: slabbe.fruit.Fruit

Creates a strawberry.

INPUT:

- weight - number, in kilos

OUTPUT:

Strawberry

EXAMPLES:

```
sage: from slabbe import Strawberry
sage: s = Strawberry(34)
sage: s
A strawberry of 34 kilos.
```

TESTS:

Testing that pickle works:

```
sage: loads(dumps(s))
A strawberry of 34 kilos.
sage: s == loads(dumps(s))
True
```

Running the test suite:

```
sage: TestSuite(s).run()
```

CHAPTER

SIX

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

[LR2014] Labbé, Sébastien, and Christophe Reutenauer. A d-dimensional Extension of Christoffel Words. arXiv:1404.4021 (April 15, 2014).

[WVL1984] Wijshoff, H. A. G, et J. Van Leeuwen. Arbitrary versus periodic storage schemes and tessellations of the plane using one type of polyomino. INFORM. AND CONTROL 62 (1984): 1-25.

[BN1991] Beauquier, D., and M. Nivat. On translating one polyomino to tile the plane. Discrete & Computational Geometry 6 (1991): 575-592. doi:10.1007/BF02574705

[BFP2009] S. Brlek, J.-M Fédou, X. Provençal, On the Tiling by Translation Problem, Discrete Applied Mathematics 157 Issue 3 (2009) 464-475. doi:10.1016/j.dam.2008.05.026

[BBL2012] A. Blondin Massé, S. Brlek, S. Labbé, A parallelogram tile fills the plane by translation in at most two distinct ways, Discrete Applied Mathematics 160 (2012) 1011-1018. doi:10.1016/j.dam.2011.12.023

[BBGL2011] A. Blondin Massé, S. Brlek, A. Garon, S. Labbé, Two infinite families of polyominoes that tile the plane by translation in two distinct ways, Theoret. Comput. Sci. 412 (2011) 4778-4786. doi:10.1016/j.tcs.2010.12.034

[BGL2012] A. Blondin Massé, A. Garon, S. Labbé, Combinatorial properties of double square tiles, Theoretical Computer Science, Available online 2 November 2012. doi:10.1016/j.tcs.2012.10.040

[K65] William Kolakoski, proposal 5304, American Mathematical Monthly 72 (1965), 674; for a partial solution, see "Self Generating Runs," by Necdet Üçoluk, Amer. Math. Mon. 73 (1966), 681-2.

[O39] R. Oldenburger, Exponent trajectories in dynamical systems, Trans. Amer. Math. Soc. 46 (1939), 453–466.

[BL2014] V. Berthé, S. Labbé, Factor Complexity of S-adic sequences generated by the Arnoux-Rauzy-Poincaré Algorithm. arXiv:1404.4189 (April, 2014).

[T1980] R., Tijdeman. The chairman assignment problem. Discrete Mathematics 32, no 3 (1980): 323-30. doi:10.1016/0012-365X(80)90269-1.

[POG] Geoffrey Grimmett, Probability on Graphs, http://www.statslab.cam.ac.uk/~grg/books/pgs.html

S

A

B

C

D

E

F

G

H

I

J

K

L

M

N

P

R

S

T

U

V

W

Z