brocoli Documentation

Release 1.0.0

Jean-Philippe Labbé

Oct 05, 2017

CONTENTS:

CHAPTER

ONE

INSTALLATION

To install Brocoli, follow the following steps.

1. download the tar ball source
2. in the brocoli folder, type the following command:

   $ sage -pip install –upgrade –no-index -v .

CHAPTER

TWO

GEOMETRIC REPRESENTATION OF COXETER GROUPS

The Brocoli module provides a base class called `GeometricRepresentationCoxeterGroup`. Base class for the Geometric Representations of Coxeter Groups

A geometric representations of a Coxeter group is a morphism from the Coxeter Group to a reflection group stabilizing a (sometimes non-canonical) bilinear form obtained from the group. This class allows to do computations related to the geometric representation. It is possible to compute roots, weights, images of the elements, it is also possible to visualize several related geometric objects such as the Tits cone, the isotropic cone, the limit roots, and more.

EXAMPLES:

```
sage: from brocoli import *
sage: M1 = CoxeterMatrix([[1,4,4],[4,1,4],[4,4,1]])
sage: GR1 = GeometricRepresentationCoxeterGroup(M1);GR1
Geometric representation of a Coxeter group of rank 3 with Coxeter matrix
[1 4 4]
[4 1 4]
[4 4 1]
```

If the parameter `exact` is set to `False`, then the representation is done in `RDF`:

```
sage: GR1rdf = GeometricRepresentationCoxeterGroup(M1,exact=False)
sage: GR1rdf.base_ring()
Real Double Field
sage: GR1.base_ring()
Universal Cyclotomic Field
```

It is possible to specify a the symbols (or alphabet) for the generators:

```
sage: M2 = CoxeterMatrix([[1,oo,oo,oo],[oo,1,oo,oo],[oo,oo,1,oo],[oo,oo,oo,1]])
sage: GR2 = GeometricRepresentationCoxeterGroup(M2, generators=['a','b','c','d'])
sage: GR2.generators()
('a', 'b', 'c', 'd')
```

The geometric representation uses a certain bilinear form:

```
sage: GR1.bilinear_form()
[                    1 -1/2*E(8) + 1/2*E(8)^3 -1/2*E(8) + 1/2*E(8)^3]
[-1/2*E(8) + 1/2*E(8)^3                     1 -1/2*E(8) + 1/2*E(8)^3]
[-1/2*E(8) + 1/2*E(8)^3 -1/2*E(8) + 1/2*E(8)^3                     1]
sage: GR2.bilinear_form()
[ 1 -1 -1 -1]
[-1  1 -1 -1]
[-1 -1  1 -1]
[-1 -1 -1  1]
```

We can ask for the signature of the bilinear form:

```
sage: GR1.signature()
(2, 1, 0)
sage: GR2.signature()
(3, 1, 0)
```

so both of them are Lorentzian; they act on Lorentz space:

```
sage: GR2.is_lorentzian()
True
```

We can go from generators to their matrices:

```
sage: GR2.generator_to_reflection('a')
[-1  2  2  2]
[ 0  1  0  0]
[ 0  0  1  0]
[ 0  0  0  1]
sage: GR2.simple_reflections()[0]
[-1  2  2  2]
[ 0  1  0  0]
[ 0  0  1  0]
[ 0  0  0  1]
```

One can compute some elements of length two (the first two):

```
sage: GR2.elements(2)[:2]
[
[ 1  0  0  0]  [ 3  6  6 -2]
[ 2 -1  2  2]  [ 0  1  0  0]
[ 0  0  1  0]  [ 0  0  1  0]
[ 6 -2  6  3], [ 2  2  2 -1]
]
```

We can then ask for a reduced expression for an element that was obtained:

```
sage: GR2.matrix_to_word(GR2.elements(2)[0])
word: bd
sage: GR2.matrix_to_word(GR2.elements(2)[1])
word: da
```

We can compute some roots:

```
sage: GR1.roots(1)
{(1, E(8) - E(8)^3, 0), (1, 0, E(8) - E(8)^3), (E(8) - E(8)^3, 1, 0), (E(8) - E(8)^3,␣
↪0, 1), (0, E(8) - E(8)^3, 1), (0, 1, E(8) - E(8)^3)}
sage: GR2.roots(1)[:5]
[(0, 2, 0, 1), (0, 0, 2, 1), (0, 2, 1, 0), (0, 1, 0, 2), (2, 0, 0, 1)]
```

and some weights:

```
sage: GR1.fundamental_weights()
((1 - E(8) + E(8)^3, -1, -1),
 (-1, 1 - E(8) + E(8)^3, -1),
 (-1, -1, 1 - E(8) + E(8)^3))
sage: GR2.fundamental_weights()
((1/4, -1/4, -1/4, -1/4),
```

```
 (-1/4, 1/4, -1/4, -1/4),
 (-1/4, -1/4, 1/4, -1/4),
 (-1/4, -1/4, -1/4, 1/4))
```

Finally, we can visualize the isotropic cone:

```
sage: GR1.visualize_isotropic_cone()
Graphics object consisting of 4 graphics primitives
sage: GR2.visualize_isotropic_cone()
Graphics3d Object
```

But that's a bit boring, so we can add roots:

```
sage: GR1.visualize_roots([0,1,2])
Graphics object consisting of 22 graphics primitives
sage: GR2.visualize_roots([0,1,2])
Graphics3d Object
```

or weights:

```
sage: GR1.visualize_weights([0,1,2])
Graphics object consisting of 19 graphics primitives
sage: GR2.visualize_weights([0,1,2])
Graphics3d Object
```

or limit roots:

```
sage: GR1.visualize_limit_roots([3,4,5])
Graphics object consisting of 51 graphics primitives
sage: GR2.visualize_limit_roots([2,3,4])
Graphics3d Object
```

we can add the isotropic cone:

```
sage: GR1.visualize_limit_roots([3,4,5],isotropic=True)
Graphics object consisting of 55 graphics primitives
sage: GR2.visualize_limit_roots([2,3,4],isotropic=True)
Graphics3d Object
```

and finally the Tits cone:

```
sage: GR1.visualize_tits_cone(2)
Graphics object consisting of 13 graphics primitives
sage: GR2.visualize_tits_cone(3)
Graphics3d Object
```

REFERENCES:

- Chapter 5 of Reflection Groups and Coxeter Groups, J. Humphreys, (1990)
- Discrete linear groups generated by reflections (Russian translated to English), E.B. Vinberg, (1971)
- C. Hohlweg, J.-P. Labbé, and V. Ripoll, Asymptotical behaviour of roots of infinite Coxeter groups, Canad. J. Math., **66**, (2014), no. 2, 323–353.
- M. Dyer, C. Hohlweg, V. Ripoll, Imaginary cones and limit roots of infinite Coxeter groups, Math. Z. **284** (2016), no. 3-4, 715–780.
- C. Hohlweg, J.-P. Préaux, V. Ripoll, On the Limit Set of Root Systems of Coxeter Groups acting on Lorentzian spaces, arxiv:1305.0052 (July 2013), 24 p.

- H. Chen and J.-P. Labbé, Lorentzian Coxeter systems and Boyd–Maxwell ball packings, Geom. Dedic., **174**, (2015), no. 1, 43–73.
- H. Chen and J.-P. Labbé, Limit directions for Lorentzian Coxeter systems, Groups Geom. Dyn. **11** (2017), no. 2, 469–498.
- C. Hohlweg and J.-P. Labbé, On inversion sets and the weak order in Coxeter groups, European J. Combin. **55** (2016), 1–19.

AUTHORS:

- Jean-Philippe Labbé (2011-): Initial version
- Vivien Ripoll (2011-2013): Added more options and functionalities

class brocoli.geom_repr_class.**GeometricRepresentationCoxeterGroup**(*coxeter_matrix*, *generators=None*, *exact=True*)

Base class for geometric representations of Coxeter groups.

INPUT:

- `coxeter_matrix` – a CoxeterMatrix object form Sage
- `generators` – an alphabet for the generators (default: `None`)
- `exact` – boolean keyword argument (default: `True`); whether to do the computation in an exact ring.

EXAMPLES:

To create a geometric representation, we start with Coxeter matrices:

```
sage: from brocoli import *
sage: QF.<a> = QuadraticField(2)
sage: M1 = CoxeterMatrix([[1,4,4],[4,1,4],[4,4,1]])
sage: M2 = CoxeterMatrix([[1,4,oo],[4,1,4],[oo,4,1]])
sage: M3 = CoxeterMatrix([[1,4,-2],[4,1,4],[-2,4,1]])
sage: M4 = CoxeterMatrix([[1,4,-1.5],[4,1,4],[-1.5,4,1]])
sage: M5 = CoxeterMatrix([[1,4,-3/2],[4,1,4],[-3/2,4,1]])
sage: M6 = CoxeterMatrix([[1,4,-3/2],[4,1,-a],[-3/2,-a,1]])
sage: M7 = CoxeterMatrix([[1,4,-1.5],[4,1,-a],[-1.5,-a,1]])
```

depending on the Coxeter matrix, different base rings are chosen for the representation. When some labels are integers greater or equal to 4, then it uses the Universal Cyclotomic Field. If some other not rational labels are given, then it tries to find the appropriate ring:

```
sage: for m in [M1,M2,M3,M4,M5,M6,M7]:
....:     print(GeometricRepresentationCoxeterGroup(m).base_ring())
....:
Universal Cyclotomic Field
Universal Cyclotomic Field
Universal Cyclotomic Field
Real Double Field
Universal Cyclotomic Field
Algebraic Field
Real Double Field
```

There is a slightly different behavior if there are no labels that require the Univeral Cyclotomic Field:

```
sage: M1p = CoxeterMatrix([[1,3,3],[3,1,3],[3,3,1]])
sage: M2p = CoxeterMatrix([[1,3,oo],[3,1,3],[oo,3,1]])
sage: M3p = CoxeterMatrix([[1,3,-2],[3,1,3],[-2,3,1]])
sage: M4p = CoxeterMatrix([[1,3,-1.5],[3,1,3],[-1.5,3,1]])
sage: M5p = CoxeterMatrix([[1,3,-3/2],[3,1,3],[-3/2,3,1]])
sage: M6p = CoxeterMatrix([[1,3,-3/2],[3,1,-a],[-3/2,-a,1]])
sage: M7p = CoxeterMatrix([[1,3,-1.5],[3,1,-a],[-1.5,-a,1]])
sage: for m in [M1p,M2p,M3p,M4p,M5p,M6p,M7p]:
....:     print(GeometricRepresentationCoxeterGroup(m).base_ring())
....:
Rational Field
Rational Field
Rational Field
Real Double Field
Rational Field
Number Field in a with defining polynomial x^2 - 2
Real Double Field
```

If the parameter `exact` is set to `False`, then computations are done in `RDF`:

```
sage: GR1rdf = GeometricRepresentationCoxeterGroup(M1,exact=False);GR1rdf.base_
↪ring()
Real Double Field
```

It is possible to specify a the symbols (or alphabet) for the generators:

```
sage: GR1 = GeometricRepresentationCoxeterGroup(M1, generators=['a','b','c'])
sage: GR1.generators()
('a', 'b', 'c')
```

TESTS:

```
sage: S = [GeometricRepresentationCoxeterGroup(CM) for CM in CoxeterMatrix.
↪samples()]
```

base_ring()

Return the base ring of the representation

OUTPUT:

A ring

EXAMPLES:

```
sage: from brocoli import *
sage: QF.<a> = QuadraticField(2)
sage: M1 = CoxeterMatrix([[1,oo,oo],[oo,1,oo],[oo,oo,1]])
sage: M2 = CoxeterMatrix([[1,-3/2,-3/2],[-3/2,1,-3/2],[-3/2,-3/2,1]])
sage: M3 = CoxeterMatrix([[1,-1.5,-1.5],[-1.5,1,-1.5],[-1.5,-1.5,1]])
sage: M4 = CoxeterMatrix([[1,3,4],[3,1,5],[4,5,1]])
sage: M5 = CoxeterMatrix([[1,3,-3/2],[3,1,-a],[-3/2,-a,1]])
sage: GR1 = GeometricRepresentationCoxeterGroup(M1);GR1.base_ring()
Rational Field
sage: GR2 = GeometricRepresentationCoxeterGroup(M2);GR2.base_ring()
Rational Field
sage: GR3 = GeometricRepresentationCoxeterGroup(M3);GR3.base_ring()
Real Double Field
sage: GR4 = GeometricRepresentationCoxeterGroup(M4);GR4.base_ring()
Universal Cyclotomic Field
```

```
sage: GR5 = GeometricRepresentationCoxeterGroup(M5);GR5.base_ring()
Number Field in a with defining polynomial x^2 - 2
```

bilinear_form

File: brocoli/geom_repr_class.py (starting at line 755)

Return the associated bilinear form of the representation

INPUT:

- `exact` – a boolean (default = `True`); whether to give the bilinear in an exact base ring or not.

OUTPUT:

A matrix giving the bilinear form

EXAMPLES:

```
sage: from brocoli import *
sage: M1 = CoxeterMatrix([[1,4,4],[4,1,4],[4,4,1]])
sage: GR1 = GeometricRepresentationCoxeterGroup(M1)
sage: GR1.bilinear_form()
[                     1 -1/2*E(8) + 1/2*E(8)^3 -1/2*E(8) + 1/2*E(8)^3]
[-1/2*E(8) + 1/2*E(8)^3                      1 -1/2*E(8) + 1/2*E(8)^3]
[-1/2*E(8) + 1/2*E(8)^3 -1/2*E(8) + 1/2*E(8)^3                     1]
sage: GR1.bilinear_form(exact=False)
[                1.0 -0.7071067811865475 -0.7071067811865475]
[-0.7071067811865475                 1.0 -0.7071067811865475]
[-0.7071067811865475 -0.7071067811865475                 1.0]

sage: M2 = CoxeterMatrix([[1,4,oo],[4,1,4],[oo,4,1]])
sage: GR2 = GeometricRepresentationCoxeterGroup(M2)
sage: GR2.bilinear_form()
[                     1 -1/2*E(8) + 1/2*E(8)^3                    -1]
[-1/2*E(8) + 1/2*E(8)^3                      1 -1/2*E(8) + 1/2*E(8)^3]
[                    -1 -1/2*E(8) + 1/2*E(8)^3                     1]
sage: GR2.bilinear_form(exact=False)
[                1.0 -0.7071067811865475                -1.0]
[-0.7071067811865475                 1.0 -0.7071067811865475]
[               -1.0 -0.7071067811865475                 1.0]

sage: M3 = CoxeterMatrix([[1,4,-1.5],[4,1,4],[-1.5,4,1]])
sage: GR3 = GeometricRepresentationCoxeterGroup(M3)
sage: GR3.bilinear_form()
[                1.0 -0.7071067811865476                -1.5]
[-0.7071067811865476                 1.0 -0.7071067811865476]
[               -1.5 -0.7071067811865476                 1.0]

sage: QF.<a> = QuadraticField(3/2)
sage: Sq3 = CoxeterMatrix([[1,-a,2,2],[-a,1,3,2],[2,3,1,-a],[2,2,-a,1]])
sage: GR_Sq3 = GeometricRepresentationCoxeterGroup(Sq3)
sage: GR_Sq3.bilinear_form()
[   1   -a    0    0]
[  -a    1 -1/2    0]
[   0 -1/2    1   -a]
[   0    0   -a    1]
```

coxeter_matrix()

Return the associated Coxeter matrix

OUTPUT:

A Coxeter Matrix object

EXAMPLES:

```
sage: from brocoli import *
sage: M1 = CoxeterMatrix([[1,oo,oo,oo],[oo,1,oo,oo],[oo,oo,1,oo],[oo,oo,oo,
↪1]])
sage: M2 = CoxeterMatrix([[1,3,3,3],[3,1,3,3],[3,3,1,3],[3,3,3,1]])
sage: M3 = CoxeterMatrix([[1,4,-2],[4,1,4],[-2,4,1]])
sage: GR1 = GeometricRepresentationCoxeterGroup(M1)
sage: GR2 = GeometricRepresentationCoxeterGroup(M2)
sage: GR3 = GeometricRepresentationCoxeterGroup(M3)
sage: GR1.coxeter_matrix()
[ 1 -1 -1 -1]
[-1  1 -1 -1]
[-1 -1  1 -1]
[-1 -1 -1  1]
sage: GR2.coxeter_matrix()
[1 3 3 3]
[3 1 3 3]
[3 3 1 3]
[3 3 3 1]
sage: GR3.coxeter_matrix()
[ 1  4 -2]
[ 4  1  4]
[-2  4  1]
```

edge_labels()

Return the edge labels of the associated the Coxeter graph

OUTPUT:

a tuple containing the labels

EXAMPLES:

```
sage: from brocoli import *
sage: M1 = CoxeterMatrix([[1,oo,oo,oo],[oo,1,oo,oo],[oo,oo,1,oo],[oo,oo,oo,
↪1]])
sage: M2 = CoxeterMatrix([[1,3,3,3],[3,1,3,-1.5],[3,3,1,-3/2],[3,-1.5,-3/2,
↪1]])
sage: M3 = CoxeterMatrix([[1,4,-2],[4,1,3],[-2,3,1]])
sage: GR1 = GeometricRepresentationCoxeterGroup(M1)
sage: GR2 = GeometricRepresentationCoxeterGroup(M2)
sage: GR3 = GeometricRepresentationCoxeterGroup(M3)
sage: GR1.edge_labels()
(-1,)
sage: GR2.edge_labels()
(-1.50000000000000, 3.00000000000000)
sage: GR3.edge_labels()
(-2, 3, 4)
```

elements

File: brocoli/geom_repr_class.py (starting at line 1345)

Return the elements of the represented Coxeter group with length given by `length`.

INPUT:

- `length` – a non-negative integer

OUTPUT:

A set of matrices

EXAMPLES:

Finite examples:

```
sage: from brocoli import *
sage: Di5 = CoxeterMatrix([[1,5],[5,1]])
sage: GR_Di5 = GeometricRepresentationCoxeterGroup(Di5)
sage: [len(GR_Di5.elements(i)) for i in range(7)]
[1, 2, 2, 2, 2, 1, 0]
sage: A3 = CoxeterMatrix([[1,3,2],[3,1,3],[2,3,1]])
sage: GR_A3 = GeometricRepresentationCoxeterGroup(A3)
sage: [len(GR_A3.elements(i)) for i in range(8)]
[1, 3, 5, 6, 5, 3, 1, 0]
```

Infinite examples:

```
sage: M1 = CoxeterMatrix([[1,4,4],[4,1,4],[4,4,1]])
sage: GR1 = GeometricRepresentationCoxeterGroup(M1)
sage: [len(GR1.elements(i)) for i in range(6)]
[1, 3, 6, 12, 21, 36]
sage: M2 = CoxeterMatrix([[1,4,-3/2],[4,1,4],[-3/2,4,1]])
sage: GR2 = GeometricRepresentationCoxeterGroup(M2)
sage: [len(GR2.elements(i)) for i in range(6)]
[1, 3, 6, 12, 22, 40]
```

TESTS:

```
sage: GR1.elements(-1)
Traceback (most recent call last):
...
ValueError: length has to be a positive integer
```

elliptic_elements(*length*)

For Lorentzian Coxeter groups, return the elliptic elements of the representation of length `length`

INPUT:

- `length` – a non-negative integer

OUTPUT:

A set of elements

EXAMPLES:

```
sage: from brocoli import *
sage: M1 = CoxeterMatrix([[1,oo,oo,oo],[oo,1,oo,oo],[oo,oo,1,oo],[oo,oo,oo,
↪1]])
sage: M2 = CoxeterMatrix([[1,3,3,3],[3,1,3,3],[3,3,1,3],[3,3,3,1]])
sage: M3 = CoxeterMatrix([[1,oo,oo,oo],[oo,1,3,3],[oo,3,1,3],[oo,3,3,1]])
sage: M4 = CoxeterMatrix([[1,-2,-2,-2],[-2,1,-2,-2],[-2,-2,1,-2],[-2,-2,-2,
↪1]])

sage: GR1 = GeometricRepresentationCoxeterGroup(M1)
sage: GR2 = GeometricRepresentationCoxeterGroup(M2)
sage: GR3 = GeometricRepresentationCoxeterGroup(M3)
```

```
sage: GR4 = GeometricRepresentationCoxeterGroup(M4)

sage: GR1.elliptic_elements(0)
{[1 0 0 0]
[0 1 0 0]
[0 0 1 0]
[0 0 0 1]}
sage: len(GR1.elliptic_elements(3))
12
sage: len(GR1.elliptic_elements(4))
0

sage: len(GR2.elliptic_elements(3))
6
sage: len(GR2.elliptic_elements(4))
24

sage: len(GR3.elliptic_elements(3))
9
sage: len(GR3.elliptic_elements(4))
12

sage: GR4.elliptic_elements(4)[-1]
Traceback (most recent call last):
...
IndexError: list index out of range
```

TESTS:

```
sage: for gr in [GR1,GR2,GR3,GR4]:
....:     for ell in [0,4]:
....:         elliptic = gr.elliptic_elements(ell)
....:         print(ell, len(elliptic))
....:         if len(elliptic)>0:
....:             print(elliptic[-1])
....:
0 1
[1 0 0 0]
[0 1 0 0]
[0 0 1 0]
[0 0 0 1]
4 0
0 1
[1 0 0 0]
[0 1 0 0]
[0 0 1 0]
[0 0 0 1]
4 24
[ 0 -1  2  2]
[ 0 -2  6  3]
[ 0  0  1  0]
[ 1 -2  4  2]
0 1
[1 0 0 0]
[0 1 0 0]
[0 0 1 0]
[0 0 0 1]
4 12
```

```
[ 1  0  0  0]
[12 -2  0  3]
[ 4 -1  0  2]
[ 8 -2  1  2]
0 1
[1 0 0 0]
[0 1 0 0]
[0 0 1 0]
[0 0 0 1]
4 0
```

fundamental_space_weights

File: brocoli/geom_repr_class.py (starting at line 1065)

Return the fundamental weights which are positive with respect to the bilinear form.

INPUT:

- `exact` – a boolean (default = `True`); whether to give the bilinear in an exact base ring or not.

OUTPUT:

a list of vectors

EXAMPLES:

```
sage: from brocoli import *
sage: M1 = CoxeterMatrix([[1,oo,oo,oo],[oo,1,oo,oo],[oo,oo,1,oo],[oo,oo,oo,
↪1]])
sage: M2 = CoxeterMatrix([[1,3,3,3],[3,1,3,3],[3,3,1,3],[3,3,3,1]])
sage: M3 = CoxeterMatrix([[1,oo,oo,oo],[oo,1,3,3],[oo,3,1,3],[oo,3,3,1]])
sage: GR1 = GeometricRepresentationCoxeterGroup(M1)
sage: GR2 = GeometricRepresentationCoxeterGroup(M2)
sage: GR3 = GeometricRepresentationCoxeterGroup(M3)
sage: GR1.fundamental_space_weights()
[(1/4, -1/4, -1/4, -1/4),
 (-1/4, 1/4, -1/4, -1/4),
 (-1/4, -1/4, 1/4, -1/4),
 (-1/4, -1/4, -1/4, 1/4)]
sage: GR2.fundamental_space_weights()
[]
sage: GR3.fundamental_space_weights()
[(-1/3, 1/3, -1/3, -1/3), (-1/3, -1/3, 1/3, -1/3), (-1/3, -1/3, -1/3, 1/3)]
```

fundamental_weights

File: brocoli/geom_repr_class.py (starting at line 995)

Return the fundamental weights of the representation

INPUT:

- `exact` – a boolean (default = `True`); whether to give the bilinear in an exact base ring or not.

OUTPUT:

A tuple of vectors

EXAMPLES:

```
sage: from brocoli import *
sage: M = CoxeterMatrix([[1,oo,oo,oo],[oo,1,oo,oo],[oo,oo,1,oo],[oo,oo,oo,1]])
sage: GR = GeometricRepresentationCoxeterGroup(M)
```

```
sage: GR.fundamental_weights()
((1/4, -1/4, -1/4, -1/4),
 (-1/4, 1/4, -1/4, -1/4),
 (-1/4, -1/4, 1/4, -1/4),
 (-1/4, -1/4, -1/4, 1/4))
sage: GR.fundamental_weights(False)
((0.25, -0.25, -0.25, -0.25),
 (-0.25, 0.25, -0.25, -0.25),
 (-0.25, -0.25, 0.25, -0.25),
 (-0.25, -0.25, -0.25, 0.25))
```

TESTS:

```
sage: N = CoxeterMatrix([[1,oo,oo,oo],[oo,1,oo,oo],[oo,oo,1,oo],[oo,oo,oo,1]])
sage: GRN = GeometricRepresentationCoxeterGroup(N)
sage: GRN._computed_weights_exact
{}
sage: GRN.fundamental_weights()
((1/4, -1/4, -1/4, -1/4),
 (-1/4, 1/4, -1/4, -1/4),
 (-1/4, -1/4, 1/4, -1/4),
 (-1/4, -1/4, -1/4, 1/4))
sage: GRN._computed_weights_exact
{(-1/4, 1/4, -1/4, -1/4), (-1/4, -1/4, 1/4, -1/4), (1/4, -1/4, -1/4, -1/4), (-
↪1/4, -1/4, -1/4, 1/4)}
sage: GRN._computed_weights_rdf
{}
sage: GRN.fundamental_weights(False)
((0.25, -0.25, -0.25, -0.25),
 (-0.25, 0.25, -0.25, -0.25),
 (-0.25, -0.25, 0.25, -0.25),
 (-0.25, -0.25, -0.25, 0.25))
sage: GRN._computed_weights_rdf
{(0.25, -0.25, -0.25, -0.25), (-0.25, 0.25, -0.25, -0.25), (-0.25, -0.25, 0.
↪25, -0.25), (-0.25, -0.25, -0.25, 0.25)}
```

generator_to_reflection (*generator*)

Return the matrix associated to `generator`

INPUT:

- `generator` – letter or number; a generator of the Coxeter group

OUTPUT:

a matrix representing the generator

EXAMPLES:

```
sage: from brocoli import *
sage: M1 = CoxeterMatrix([[1,4,4],[4,1,4],[4,4,1]])
sage: GR1 = GeometricRepresentationCoxeterGroup(M1)
sage: GR1.generator_to_reflection(1)
[           -1 E(8) - E(8)^3 E(8) - E(8)^3]
[            0             1             0]
[            0             0             1]
sage: GR1.generator_to_reflection(2)
[            1             0             0]
[E(8) - E(8)^3            -1 E(8) - E(8)^3]
[            0             0             1]
```

```
sage: GR1.generator_to_reflection(3)
[            1             0             0]
[            0             1             0]
[E(8) - E(8)^3 E(8) - E(8)^3            -1]
```

Also works if we change the set of generators:

```
sage: M2 = CoxeterMatrix([[1,oo,oo],[oo,1,oo],[oo,oo,1]])
sage: GR2 = GeometricRepresentationCoxeterGroup(M2,generators=['a','b','c'])
sage: GR2.generator_to_reflection(1)
Traceback (most recent call last):
...
KeyError: 1
sage: GR2.generator_to_reflection('a')
[-1  2  2]
[ 0  1  0]
[ 0  0  1]
sage: GR2.generator_to_reflection('b')
[ 1  0  0]
[ 2 -1  2]
[ 0  0  1]
sage: GR2.generator_to_reflection('c')
[ 1  0  0]
[ 0  1  0]
[ 2  2 -1]
```

generators()

Return the generators of the associated Coxeter group as letters of an alphabet

OUTPUT:

A tuple containing the (letter)-generators of the Coxeter group

EXAMPLES:

```
sage: from brocoli import *
sage: M = CoxeterMatrix([[1,oo,oo,oo],[oo,1,oo,oo],[oo,oo,1,oo],[oo,oo,oo,1]])
sage: GR1 = GeometricRepresentationCoxeterGroup(M)
sage: GR1.generators()
(1, 2, 3, 4)
sage: GR2 = GeometricRepresentationCoxeterGroup(M,generators=['a','b','c','d
↪'])
sage: GR2.generators()
('a', 'b', 'c', 'd')
sage: GR3 = GeometricRepresentationCoxeterGroup(M,generators=['a','b','c'])
Traceback (most recent call last):
...
ValueError: The number of generators is not equal to the rank
```

hyperbolic_elements(*length*)

For Lorentzian Coxeter groups, return the hyperbolic elements of the representation of length `length`

INPUT:

- `length` – a non-negative integer

OUTPUT:

A set of elements

EXAMPLES:

```
sage: from brocoli import *
sage: M1 = CoxeterMatrix([[1,oo,oo,oo],[oo,1,oo,oo],[oo,oo,1,oo],[oo,oo,oo,
↪1]])
sage: M2 = CoxeterMatrix([[1,3,3,3],[3,1,3,3],[3,3,1,3],[3,3,3,1]])
sage: M3 = CoxeterMatrix([[1,oo,oo,oo],[oo,1,3,3],[oo,3,1,3],[oo,3,3,1]])
sage: M4 = CoxeterMatrix([[1,-2,-2,-2],[-2,1,-2,-2],[-2,-2,1,-2],[-2,-2,-2,
↪1]])

sage: GR1 = GeometricRepresentationCoxeterGroup(M1)
sage: GR2 = GeometricRepresentationCoxeterGroup(M2)
sage: GR3 = GeometricRepresentationCoxeterGroup(M3)
sage: GR4 = GeometricRepresentationCoxeterGroup(M4)

sage: GR1.hyperbolic_elements(2)[0]
Traceback (most recent call last):
...
IndexError: list index out of range
sage: len(GR1.hyperbolic_elements(3))
24
sage: len(GR1.hyperbolic_elements(4))
72

sage: len(GR2.hyperbolic_elements(3))
0
sage: len(GR2.hyperbolic_elements(4))
24

sage: len(GR3.hyperbolic_elements(3))
18
sage: len(GR3.hyperbolic_elements(4))
54

sage: GR4.hyperbolic_elements(0)[-1]
Traceback (most recent call last):
...
IndexError: list index out of range
```

hyperbolic_limit_roots

File: brocoli/geom_repr_class.py (starting at line 2339)

For Lorentzian Coxeter groups, return the limit roots coming from the eigenvectors of hyperbolic elements of length `length`

INPUT:

- `length` – a non-negative integer

OUTPUT:

A set of vectors

EXAMPLES:

```
sage: from brocoli import *
sage: M = CoxeterMatrix([[1,3,3,3],[3,1,3,3],[3,3,1,3],[3,3,3,1]])
sage: GR = GeometricRepresentationCoxeterGroup(M)
sage: [GR.hyperbolic_limit_roots(i) for i in range(4)]
[{}, {}, {}, {}]
sage: len(GR.hyperbolic_limit_roots(4)) # long time
24
```

```
sage: M1 = CoxeterMatrix([[1,-2,-2,-2],[-2,1,-2,-2],[-2,-2,1,-2],[-2,-2,-2,
↪1]])
sage: GR1 = GeometricRepresentationCoxeterGroup(M1)
sage: [len(GR1.hyperbolic_limit_roots(i)) for i in range(4)]
[0, 0, 12, 24]

sage: M2 = CoxeterMatrix([[1,4,4],[4,1,4],[4,4,1]])
sage: GR2 = GeometricRepresentationCoxeterGroup(M2)
sage: for lm in GR2.limit_roots(3):
....:     print(lm)
....:
(1, 3.546455444684995?, 1.883203505913526?)
(1, 1.883203505913526?, 3.546455444684995?)
(1, 0.2819716800611949?, 0.5310100564595692?)
(1, 1.883203505913526?, 0.5310100564595692?)
(1, 0.5310100564595692?, 1.883203505913526?)
(1, 0.5310100564595692?, 0.2819716800611949?)
```

identity_element

File: brocoli/geom_repr_class.py (starting at line 835)

Return the matrix corresponding to the identity element, i.e. the identity matrix of the representation space.

OUTPUT:

an identity matrix

EXAMPLES:

```
sage: from brocoli import *
sage: M1 = CoxeterMatrix([[1,4,4],[4,1,4],[4,4,1]])
sage: GR1 = GeometricRepresentationCoxeterGroup(M1)
sage: GR1.identity_element()
[1 0 0]
[0 1 0]
[0 0 1]
sage: GR1.identity_element().base_ring()
Universal Cyclotomic Field
sage: M2 = CoxeterMatrix([[1,4,4,-2],[4,1,4,-1],[4,4,1,-1.5],[-2,-1,-1.5,1]])
sage: GR2 = GeometricRepresentationCoxeterGroup(M2)
sage: GR2.identity_element()
[1.0 0.0 0.0 0.0]
[0.0 1.0 0.0 0.0]
[0.0 0.0 1.0 0.0]
[0.0 0.0 0.0 1.0]
sage: GR2.identity_element().base_ring()
Real Double Field
```

is_affine()

Check if `self` is a geometric representation of an affine Coxeter group

EXAMPLES:

```
sage: from brocoli import *
sage: C4t = CoxeterMatrix([[1,4,2,2],[4,1,3,2],[2,3,1,4],[2,2,4,1]])
sage: GR_C4t = GeometricRepresentationCoxeterGroup(C4t)
sage: GR_C4t.is_affine()
True
```

```
sage: Di_Af = CoxeterMatrix([[1,oo],[oo,1]])
sage: GR_DiAf = GeometricRepresentationCoxeterGroup(Di_Af)
sage: GR_DiAf.is_affine()
True

sage: B2 = CoxeterMatrix([[1,4],[4,1]])
sage: GR_B2 = GeometricRepresentationCoxeterGroup(B2)
sage: GR_B2.is_affine()
False
```

TESTS:

```
sage: from brocoli import *
sage: Affine = [GeometricRepresentationCoxeterGroup(CM) for CM in␣
↪CoxeterMatrix.samples(affine=True)]
sage: all([a.is_affine() for a in Affine])
True
sage: Finite = [GeometricRepresentationCoxeterGroup(CM) for CM in␣
↪CoxeterMatrix.samples(finite=True)]
sage: any([f.is_affine() for f in Finite])
False
```

is_degenerate_lorentzian()

Check if `self` is a geometric representation of a Lorentzian Coxeter group with a singular bilinear form

EXAMPLES:

```
sage: from brocoli import *
sage: QF.<a> = QuadraticField(3/2)
sage: Cyl = CoxeterMatrix([[1,-a,2,2],[-a,1,3,2],[2,3,1,-a],[2,2,-a,1]])
sage: GR_Cyl = GeometricRepresentationCoxeterGroup(Cyl)
sage: GR_Cyl.is_degenerate_lorentzian()
True
```

is_finite()

Check if `self` is a geometric representation of a finite Coxeter group

EXAMPLES:

```
sage: from brocoli import *
sage: B2 = CoxeterMatrix([[1,4],[4,1]])
sage: A3 = CoxeterMatrix([[1,3,2],[3,1,3],[2,3,1]])
sage: GR_B2 = GeometricRepresentationCoxeterGroup(B2)
sage: GR_A3 = GeometricRepresentationCoxeterGroup(A3)
sage: GR_B2.is_finite()
True
sage: GR_A3.is_finite()
True

sage: Di_Af = CoxeterMatrix([[1,oo],[oo,1]])
sage: GR_DiAf = GeometricRepresentationCoxeterGroup(Di_Af)
sage: GR_DiAf.is_finite()
False
```

TESTS:

```
sage: Finite = [GeometricRepresentationCoxeterGroup(CM) for CM in␣
↪CoxeterMatrix.samples(finite=True)]
```

```
sage: all([f.is_finite() for f in Finite])
True
sage: Affine = [GeometricRepresentationCoxeterGroup(CM) for CM in␣
↪CoxeterMatrix.samples(affine=True)]
sage: any([a.is_finite() for a in Affine])
False
```

is_lorentzian()

Check if `self` is a geometric representation of a Lorentzian Coxeter group

EXAMPLES:

```
sage: from brocoli import *
sage: L1 = CoxeterMatrix([[1,oo,2,2],[oo,1,3,2],[2,3,1,oo],[2,2,oo,1]])
sage: L2 = CoxeterMatrix([[1,-3/2,2,2],[-3/2,1,3,2],[2,3,1,-3/2],[2,2,-3/2,
↪1]])
sage: GR_L1 = GeometricRepresentationCoxeterGroup(L1)
sage: GR_L2 = GeometricRepresentationCoxeterGroup(L2)
sage: GR_L1.is_lorentzian()
True
sage: GR_L2.is_lorentzian()
False
```

If the bilinear form has only one negative eigenvalue, it is not necessarily lorentzian:

```
sage: QF.<a> = QuadraticField(3/2)
sage: Cyl = CoxeterMatrix([[1,-a,2,2],[-a,1,3,2],[2,3,1,-a],[2,2,-a,1]])
sage: GR_Cyl = GeometricRepresentationCoxeterGroup(Cyl)
sage: GR_Cyl.is_lorentzian()
False
sage: GR_Cyl.signature()
(2, 1, 1)
```

See also:

- *is_degenerate_lorentzian()*

TESTS:

```
sage: Finite = [GeometricRepresentationCoxeterGroup(CM) for CM in␣
↪CoxeterMatrix.samples(finite=True)]
sage: all([not a.is_lorentzian() for a in Finite])
True
sage: Affine = [GeometricRepresentationCoxeterGroup(CM) for CM in␣
↪CoxeterMatrix.samples(affine=True)]
sage: all([not a.is_lorentzian() for a in Affine])
True
```

limit_roots

File: brocoli/geom_repr_class.py (starting at line 2404)

For Lorentzian Coxeter groups, return the limit roots coming from the eigenvectors of infinite order elements of length `length`

INPUT:

- `length` – a non-negative integer

OUTPUT:

A set of vectors

EXAMPLES:

```
sage: from brocoli import *
sage: M1 = CoxeterMatrix([[1,oo,oo,oo],[oo,1,oo,oo],[oo,oo,1,oo],[oo,oo,oo,
↪1]])
sage: M2 = CoxeterMatrix([[1,3,3,3],[3,1,3,3],[3,3,1,3],[3,3,3,1]])
sage: M3 = CoxeterMatrix([[1,oo,oo,oo],[oo,1,3,3],[oo,3,1,3],[oo,3,3,1]])
sage: M4 = CoxeterMatrix([[1,5,oo,oo],[5,1,3,3],[oo,3,1,3],[oo,3,3,1]])
sage: GR1 = GeometricRepresentationCoxeterGroup(M1)
sage: GR2 = GeometricRepresentationCoxeterGroup(M2)
sage: GR3 = GeometricRepresentationCoxeterGroup(M3)
sage: GR4 = GeometricRepresentationCoxeterGroup(M4)
sage: [len(GR1.limit_roots(i)) for i in range(4)]
[0, 0, 6, 24]
sage: [len(GR2.limit_roots(i)) for i in range(4)]
[0, 0, 0, 4]
sage: [len(GR3.limit_roots(i)) for i in range(4)]
[0, 0, 3, 19]
sage: [len(GR4.limit_roots(i)) for i in range(4)] # long time (8 seconds)
[0, 0, 2, 19]
```

TESTS:

```
sage: from brocoli import *
sage: M = CoxeterMatrix([[1,4,oo,oo],[4,1,oo,oo],[oo,oo,1,-2],[oo,oo,-2,1]])
sage: GR = GeometricRepresentationCoxeterGroup(M)
sage: l_roots = GR.limit_roots(4) # long time (60 seconds)
There was an approximation problem with the element 1241
There was an approximation problem with the element 1421
There was an approximation problem with the element 1321
There was an approximation problem with the element 1231
```

matrix_to_word(*matrix*)

Return a reduced word associated to `matrix`

EXAMPLES:

```
sage: from brocoli import *
sage: M = CoxeterMatrix([[1,oo,oo,oo],[oo,1,oo,oo],[oo,oo,1,oo],[oo,oo,oo,1]])
sage: GR = GeometricRepresentationCoxeterGroup(M)
```

The matrices have to be created beforehand inside the representation:

```
sage: my_ident = identity_matrix(GR.base_ring(),4)
sage: GR.matrix_to_word(my_ident)
Traceback (most recent call last):
...
ValueError: the matrix was not computed before.
sage: e = GR.identity_element()
sage: GR.matrix_to_word(e)
word:
sage: elmts = GR.elements(2)
sage: for elmt in elmts:
....:     print(GR.matrix_to_word(elmt))
....:
24
41
```

```
14
43
13
23
34
42
21
31
12
32
```

parabolic_elements (*length*)

For Lorentzian Coxeter groups, return the parabolic elements of the representation of length `length`

INPUT:

- `length` – a non-negative integer

OUTPUT:

A set of elements

EXAMPLES:

```
sage: from brocoli import *
sage: M1 = CoxeterMatrix([[1,oo,oo,oo],[oo,1,oo,oo],[oo,oo,1,oo],[oo,oo,oo,
↪1]])
sage: M2 = CoxeterMatrix([[1,3,3,3],[3,1,3,3],[3,3,1,3],[3,3,3,1]])
sage: M3 = CoxeterMatrix([[1,oo,oo,oo],[oo,1,3,3],[oo,3,1,3],[oo,3,3,1]])
sage: M4 = CoxeterMatrix([[1,-2,-2,-2],[-2,1,-2,-2],[-2,-2,1,-2],[-2,-2,-2,
↪1]])

sage: GR1 = GeometricRepresentationCoxeterGroup(M1)
sage: GR2 = GeometricRepresentationCoxeterGroup(M2)
sage: GR3 = GeometricRepresentationCoxeterGroup(M3)
sage: GR4 = GeometricRepresentationCoxeterGroup(M4)

sage: GR1.parabolic_elements(0)
{}
sage: len(GR1.parabolic_elements(3))
0
sage: len(GR1.parabolic_elements(4))
36

sage: len(GR2.parabolic_elements(3))
24
sage: len(GR2.parabolic_elements(4))
24

sage: len(GR3.parabolic_elements(3))
6
sage: len(GR3.parabolic_elements(4))
24

sage: GR4.parabolic_elements(4)[-1]
Traceback (most recent call last):
...
IndexError: list index out of range
```

parabolic_limit_roots

File: brocoli/geom_repr_class.py (starting at line 2254)

For Lorentzian Coxeter groups, return the limit roots coming from the eigenvectors of parabolic elements of length `length`

INPUT:

- `length` – a non-negative integer

OUTPUT:

A set of vectors

EXAMPLES:

```
sage: from brocoli import *
sage: M = CoxeterMatrix([[1,oo,oo,oo],[oo,1,3,3],[oo,3,1,3],[oo,3,3,1]])
sage: GR = GeometricRepresentationCoxeterGroup(M)
sage: GR.parabolic_limit_roots(0)
{}
sage: GR.parabolic_limit_roots(1)
{}
sage: GR.parabolic_limit_roots(2)
{(1.0, 1.0, 0.0, -0.0), (1.0, 0.0, -0.0, 1.0), (1.0, 0.0, 1.0, -0.0)}
sage: GR.parabolic_limit_roots(3)
{(0.0, 1.0, 1.0, 1.0)}
sage: len(GR.parabolic_limit_roots(4))
10
```

TODO:

Make this method more robust and not return vectors of symbolic ring

rank()

Return the rank of the associated Coxeter group

OUTPUT:

An integer

EXAMPLES:

```
sage: from brocoli import *
sage: B2 = CoxeterMatrix([[1,4],[4,1]])
sage: GR_B2 = GeometricRepresentationCoxeterGroup(B2)
sage: GR_B2.rank()
2
sage: A3 = CoxeterMatrix([[1,3,2],[3,1,3],[2,3,1]])
sage: GR_A3 = GeometricRepresentationCoxeterGroup(A3)
sage: A3.rank()
3
sage: M = CoxeterMatrix([[1,oo,oo,oo],[oo,1,oo,oo],[oo,oo,1,oo],[oo,oo,oo,1]])
sage: GR = GeometricRepresentationCoxeterGroup(M)
sage: GR.rank()
4
```

roots

File: brocoli/geom_repr_class.py (starting at line 1698)

Return the roots of the representation of depth `depth`

INPUT:

- `depth` – a non-negative integer

OUTPUT:

A set of roots of depth `depth`

EXAMPLES:

```
sage: from brocoli import *
sage: M = CoxeterMatrix([[1,4,oo,oo],[4,1,oo,oo],[oo,oo,1,-2],[oo,oo,-2,1]])
sage: GR = GeometricRepresentationCoxeterGroup(M)
sage: GR.roots(0)
{(0, 0, 0, 1), (1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0)}
sage: len(GR.roots(1))
12
sage: len(GR.roots(2))
34
sage: A3 = CoxeterMatrix([[1,3,2],[3,1,3],[2,3,1]])
sage: GR_A3 = GeometricRepresentationCoxeterGroup(A3)
sage: [len(GR_A3.roots(i)) for i in range(4)]
[3, 2, 1, 0]

sage: QF.<a> = QuadraticField(3/2)
sage: M2 = CoxeterMatrix([[1,-a,5],[-a,1,-3/2],[5,-3/2,1]])
sage: GR2 = GeometricRepresentationCoxeterGroup(M2)
sage: [len(GR2.roots(i)) for i in range(5)]
[3, 4, 7, 11, 18]
```

`signature`

File: brocoli/geom_repr_class.py (starting at line 1105)

Return the signature of the associated quadratic space, i.e., the vector space equipped with the bilinear form

OUTPUT:

A 3-tuple containing the number of positive, negative and zero eigenvalues of the bilinear form

EXAMPLES:

A finite type:

```
sage: from brocoli import *
sage: B2 = CoxeterMatrix([[1,4],[4,1]])
sage: GR_B2 = GeometricRepresentationCoxeterGroup(B2)
sage: GR_B2.signature()
(2, 0, 0)
```

An affine type has zero as an eigenvalue of multiplicity 1:

```
sage: Di_Af = CoxeterMatrix([[1,oo],[oo,1]])
sage: GR_DiAf = GeometricRepresentationCoxeterGroup(Di_Af)
sage: GR_DiAf.signature()
(1, 0, 1)
```

A Lorentzian type in rank 2:

```
sage: Di_Hy = CoxeterMatrix([[1,-2],[-2,1]])
sage: GR_DiHy = GeometricRepresentationCoxeterGroup(Di_Hy)
sage: GR_DiHy.signature()
(1, 1, 0)
```

A Lorentzian type in rank 4:

```
sage: M1 = CoxeterMatrix([[1,oo,oo,oo],[oo,1,oo,oo],[oo,oo,1,oo],[oo,oo,oo,
↪1]])
sage: GR1 = GeometricRepresentationCoxeterGroup(M1)
sage: GR1.signature()
(3, 1, 0)
```

The affine type $\tilde{C}_4$:

```
sage: C4t = CoxeterMatrix([[1,4,2,2],[4,1,3,2],[2,3,1,4],[2,2,4,1]])
sage: GR_C4t = GeometricRepresentationCoxeterGroup(C4t)
sage: GR_C4t.signature()
(3, 0, 1)
```

A special Lorentzian type:

```
sage: L1 = CoxeterMatrix([[1,oo,2,2],[oo,1,3,2],[2,3,1,oo],[2,2,oo,1]])
sage: GR_L1 = GeometricRepresentationCoxeterGroup(L1)
sage: GR_L1.signature()
(3, 1, 0)
```

If we change the entry for the infinite labels to $-\sqrt{3}/2$, we get:

```
sage: QF.<a> = QuadraticField(3/2)
sage: Cyl = CoxeterMatrix([[1,-a,2,2],[-a,1,3,2],[2,3,1,-a],[2,2,-a,1]])
sage: GR_Cyl = GeometricRepresentationCoxeterGroup(Cyl)
sage: GR_Cyl.signature()
(2, 1, 1)
```

Putting a small label gives:

```
sage: L2 = CoxeterMatrix([[1,-3/2,2,2],[-3/2,1,3,2],[2,3,1,-3/2],[2,2,-3/2,
↪1]])
sage: GR_L2 = GeometricRepresentationCoxeterGroup(L2)
sage: GR_L2.signature()
(2, 2, 0)
```

Using an inexact base ring raises an error:

```
sage: L3 = CoxeterMatrix([[1,-1.5,2,2],[-1.5,1,3,2],[2,3,1,-1.5],[2,2,-1.5,
↪1]])
sage: GR_L3 = GeometricRepresentationCoxeterGroup(L3)
sage: GR_L3.signature()
Traceback (most recent call last):
...
ValueError: the base ring is not exact
```

TESTS:

```
sage: Finite = [GeometricRepresentationCoxeterGroup(CM) for CM in␣
↪CoxeterMatrix.samples(finite=True)]
sage: all(gm.signature()[1:] == (0,0) for gm in Finite)
True
sage: Affine = [GeometricRepresentationCoxeterGroup(CM) for CM in␣
↪CoxeterMatrix.samples(affine=True)]
sage: all(gm.signature()[1:] == (0,1) for gm in Affine)
True
```

simple_reflections

File: brocoli/geom_repr_class.py (starting at line 902)

Return the reflections associated to the simple roots

OUTPUT:

a tuple containing the simple reflections as matrices

EXAMPLES:

```
sage: from brocoli import *
sage: M1 = CoxeterMatrix([[1,4,4],[4,1,4],[4,4,1]])
sage: GR1 = GeometricRepresentationCoxeterGroup(M1)
sage: GR1.simple_reflections()[1]
[             1              0              0]
[E(8) - E(8)^3             -1 E(8) - E(8)^3]
[             0              0              1]
sage: M2 = CoxeterMatrix([[1,4,4,-2],[4,1,4,-1],[4,4,1,-1.5],[-2,-1,-1.5,1]])
sage: GR2 = GeometricRepresentationCoxeterGroup(M2)
sage: GR2.simple_reflections()[3]
[ 1.0  0.0  0.0  0.0]
[ 0.0  1.0  0.0  0.0]
[ 0.0  0.0  1.0  0.0]
[ 4.0  2.0  3.0 -1.0]
```

simple_roots

File: brocoli/geom_repr_class.py (starting at line 872)

Return the simple roots

OUTPUT:

A tuple containing the simple roots as vectors

EXAMPLES:

```
sage: from brocoli import *
sage: M1 = CoxeterMatrix([[1,4,4],[4,1,4],[4,4,1]])
sage: GR1 = GeometricRepresentationCoxeterGroup(M1)
sage: GR1.simple_roots()
((1, 0, 0), (0, 1, 0), (0, 0, 1))
sage: M2 = CoxeterMatrix([[1,4,4,-2],[4,1,4,-1],[4,4,1,-1.5],[-2,-1,-1.5,1]])
sage: GR2 = GeometricRepresentationCoxeterGroup(M2)
sage: GR2.simple_roots()
((1.0, 0.0, 0.0, 0.0),
 (0.0, 1.0, 0.0, 0.0),
 (0.0, 0.0, 1.0, 0.0),
 (0.0, 0.0, 0.0, 1.0))
```

space_weights

File: brocoli/geom_repr_class.py (starting at line 1905)

Return the weights of the representation with positive bilinear form evaluation

INPUT:

- `depth` – a non-negative integer
- `exact` – a boolean (default = `True`); whether to give the bilinear in an exact base ring or not.

OUTPUT:

A set of weights

EXAMPLES:

```
sage: from brocoli import *
sage: M1 = CoxeterMatrix([[1,oo,oo,oo],[oo,1,oo,oo],[oo,oo,1,oo],[oo,oo,oo,
↪1]])
sage: M2 = CoxeterMatrix([[1,3,3,3],[3,1,3,3],[3,3,1,3],[3,3,3,1]])
sage: M3 = CoxeterMatrix([[1,oo,oo,oo],[oo,1,3,3],[oo,3,1,3],[oo,3,3,1]])
sage: GR1 = GeometricRepresentationCoxeterGroup(M1)
sage: GR2 = GeometricRepresentationCoxeterGroup(M2)
sage: GR3 = GeometricRepresentationCoxeterGroup(M3)
sage: GR1.space_weights(0)
{(-1/4, 1/4, -1/4, -1/4), (-1/4, -1/4, 1/4, -1/4), (1/4, -1/4, -1/4, -1/4), (-
↪1/4, -1/4, -1/4, 1/4)}
sage: GR1.space_weights(1)
{(-1/4, -7/4, -1/4, -1/4), (-1/4, -1/4, -7/4, -1/4), (-7/4, -1/4, -1/4, -1/4),
↪ (-1/4, -1/4, -1/4, -7/4)}
sage: [len(GR1.weights(i)) for i in range(5)]
[4, 4, 12, 36, 108]

sage: GR2.space_weights(0)
{}

sage: GR3.space_weights(0)
{(-1/3, 1/3, -1/3, -1/3), (-1/3, -1/3, -1/3, 1/3), (-1/3, -1/3, 1/3, -1/3)}
sage: [len(GR3.weights(i)) for i in range(5)]
[4, 4, 12, 30, 84]

sage: [len(GR3.weights(i,False)) for i in range(5)] # not tested
[4, 8, 30, 91, 273]
```

tikz_picture_rank3(*range_roots*, *range_limitroots*, *range_orbit*, *range_weights*, *range_limitdir*, *range_hyperplane*, *limit_type=2*, *isotropic=True*)

This function returns a LaTeX expression containing a tikzpicture of a rank 3 root system.

INPUT:

- `range_roots` – list of non-negative integers; the depths of roots to print
- `range_limitroots` – list of non-negative integers; give the limit roots associated to elements with lengths in $range_limitroots$
- `range_orbit` – list of non-negative integers; lengths by which we act on roots and limit roots to obtain more (be careful not to put to many)
- `range_weights` – list of non-negative integers; depth of weights to print
- `range_limitdir` – list of non-negative integers; lengths of elements of which we print the possible associated limit direction (eigenspace of dimension 1)
- `range_hyperplane` – list of non-negative integers; depths of roots for which we print the corresponding hyperplane inside the isotropic cone
- `limit_type` – 0, 1, 2 (default: `2`); `0` : Parabolic, `1` : Hyperbolic, or `2` : All
- `isotropic` – Boolean (default: `True`); whether to draw the isotropic cone

OUTPUT:

A string containing a tikzpicture.

EXAMPLES:

```
sage: from brocoli import *
sage: M = CoxeterMatrix([[1,-10/9,oo],[-10/9,1,4],[oo,4,1]])
sage: GR = GeometricRepresentationCoxeterGroup(M)
sage: tikz_pic = GR.tikz_picture_rank3(range(3),[2,3],[0,1],range(3),[2,3],
↪range(2)) # long time (8 seconds)
sage: print("\n".join(tikz_pic.splitlines()[:10])) # long time
\begin{tikzpicture}
    [scale=1,
     isotropcone/.style={red,line join=round,thick},
     root/.style={blue},
     simpleroot/.style={black},
     dihedral_roots/.style={blue},
     limit/.style={fill=red,draw=black,diamond},
     limdir/.style={fill=orange,draw=black,diamond},
     weight/.style={fill=green,draw=black,diamond},
     rotate=0]
sage: print("\n".join(tikz_pic.splitlines()[69:74])) # long time
\node[limit,inner sep=1pt] at (2.79384317461,0.308476464594) {};
\node[limit,inner sep=1pt] at (1.6411385843,2.19814939939) {};
\node[limit,inner sep=1pt] at (1.11895251609,0.356038263561) {};
\node[limit,inner sep=1pt] at (2.35114381286,2.06826137522) {};
\node[limit,inner sep=1pt] at (0.900982021125,0.792578618065) {};
```

visualize_isotropic_cone

File: brocoli/geom_repr_class.py (starting at line 3087)

Return a graphical representation of the isotropic cone

INPUT:

- `color` – a color (default= `(1,1,0)`)
- `color_simplex` – a color to give the limit roots (default: `(0,1,0)`, green)
- `simplex` – a boolean (default: `True`); whether to plot the affine simplex

OUTPUT:

A Graphic Object

EXAMPLES:

```
sage: from brocoli import *
sage: A3 = CoxeterMatrix([[1,3,2],[3,1,3],[2,3,1]])
sage: GRA3 = GeometricRepresentationCoxeterGroup(A3)
sage: GRA3.visualize_isotropic_cone()
Traceback (most recent call last):
...
ValueError: The bilinear form is positive definite

sage: C4t = CoxeterMatrix([[1,4,2,2],[4,1,3,2],[2,3,1,4],[2,2,4,1]])
sage: GRC4t = GeometricRepresentationCoxeterGroup(C4t)
sage: GRC4t.visualize_isotropic_cone()
Graphics3d Object

sage: M = CoxeterMatrix([[1,oo,oo,oo],[oo,1,3,3],[oo,3,1,3],[oo,3,3,1]])
sage: GR = GeometricRepresentationCoxeterGroup(M)
sage: GR.visualize_isotropic_cone()
Graphics3d Object
```

visualize_limit_roots(*list_lengths, list_orbits=[0], limit_type=2, size=4, color=(1, 0, 0), color_simplex=(0, 1, 0), isotropic=False, iso_color=(1, 1, 0)*)

Return a graphical representation of limit roots

INPUT:

- `list_lengths` – list of non-negative integers; they give the length of the infinite order elements to be used to obtain limit roots
- `list_orbits` – a list of non-negative integers (default: `[0]`); they give the length of the elements that will act on the limit roots to obtain even more of them
- `limit_type` – an integer (default: `2`);
  - 2: Compute all limit roots
  - 1: Compute hyperbolic limit roots
  - anything else: Compute parabolic limit roots
- `size` – a non-negative integer (default: `4`); give the size of the point to draw
- `color` – a color to give the limit roots (default: `(1,0,0)`, red);
- `color_simplex` – a color to give the limit roots (default: `(0,1,0)`, green);
- `isotropic` – a boolean (default: `False`); whether to plot the isotropic cone in the infinite case
- `iso_color` – a color to give the isotropic cone (default: `(1, 1, 0)`, yellow)

OUTPUT:

A Graphic Object

EXAMPLES:

```
sage: from brocoli import *
sage: DiAf = CoxeterMatrix([[1,oo],[oo,1]])
sage: GRDiAf = GeometricRepresentationCoxeterGroup(DiAf)
sage: img = GRDiAf.visualize_limit_roots([2]);img
Graphics object consisting of 2 graphics primitives

sage: DiHy = CoxeterMatrix([[1,-2],[-2,1]])
sage: GRDiHy = GeometricRepresentationCoxeterGroup(DiHy)
sage: img = GRDiHy.visualize_limit_roots([2]);img
Graphics object consisting of 3 graphics primitives

sage: M1 = CoxeterMatrix([[1,4,4],[4,1,4],[4,4,1]])
sage: GR1 = GeometricRepresentationCoxeterGroup(M1)
sage: img = GR1.visualize_limit_roots([0,1,2]);img
Graphics object consisting of 3 graphics primitives
sage: img = GR1.visualize_limit_roots([3,4]);img # long time
Graphics object consisting of 21 graphics primitives
sage: img = GR1.visualize_limit_roots([3,4],[0,1,2]);img # long time
Graphics object consisting of 183 graphics primitives

sage: M2 = CoxeterMatrix([[1,oo,oo,oo],[oo,1,oo,oo],[oo,oo,1,oo],[oo,oo,oo,
↪1]])
sage: GR2 = GeometricRepresentationCoxeterGroup(M2)
sage: img = GR2.visualize_limit_roots([2,3]);img # long time
Graphics3d Object
sage: img = GR2.visualize_limit_roots([2,3],limit_type=1);img # long time
Graphics3d Object
```

```
sage: img = GR2.visualize_limit_roots([2,3],limit_type=0);img
Graphics3d Object
```

visualize_roots(*list_depths*, *size=4*, *color=(1, 0, 0)*, *color_simplex=(0, 1, 0)*, *isotropic=True*, *iso_color=(1, 1, 0)*)

Return a graphical representation of roots of depths in `list_depths`

INPUT:

- `list_depths` – a list of non-negative integers; give the depths of the roots to plot
- `size` – a non-negative integer (default: `4`); give the size of the point to draw
- `color` – a color to give the roots (default: `(1,0,0)`, red)
- `color_simplex` – a color to give the simplex (default: `(0,1,0)`, green)
- `isotropic` – a boolean (default: `True`); whether to plot the isotropic cone in the infinite case
- `iso_color` – a color to give the isotropic cone (default: `(1, 1, 0)`, yellow)

OUTPUT:

A Graphic Object

EXAMPLES:

Rank 2 examples:

```
sage: from brocoli import *
sage: A2 = CoxeterMatrix([[1,3],[3,1]])
sage: GRA2 = GeometricRepresentationCoxeterGroup(A2)
sage: img = GRA2.visualize_roots(range(2));img
Graphics object consisting of 4 graphics primitives

sage: B2 = CoxeterMatrix([[1,4],[4,1]])
sage: GRB2 = GeometricRepresentationCoxeterGroup(B2)
sage: img = GRB2.visualize_roots(range(2));img
Graphics object consisting of 5 graphics primitives

sage: DiAf = CoxeterMatrix([[1,oo],[oo,1]])
sage: GRDiAf = GeometricRepresentationCoxeterGroup(DiAf)
sage: img = GRDiAf.visualize_roots(range(5));img
Graphics object consisting of 12 graphics primitives

sage: DiHy = CoxeterMatrix([[1,-2],[-2,1]])
sage: GRDiHy = GeometricRepresentationCoxeterGroup(DiHy)
sage: img = GRDiHy.visualize_roots(range(2));img
Graphics object consisting of 8 graphics primitives
```

Rank 3 examples:

```
sage: A3 = CoxeterMatrix([[1,3,2],[3,1,3],[2,3,1]])
sage: GRA3 = GeometricRepresentationCoxeterGroup(A3)
sage: img = GRA3.visualize_roots(range(3));img
Graphics object consisting of 9 graphics primitives

sage: B3 = CoxeterMatrix([[1,4,2],[4,1,3],[2,3,1]])
sage: GRB3 = GeometricRepresentationCoxeterGroup(B3)
sage: img = GRB3.visualize_roots(range(4));img
Graphics object consisting of 12 graphics primitives
```

```
sage: H3 = CoxeterMatrix([[1,5,2],[5,1,3],[2,3,1]])
sage: GRH3 = GeometricRepresentationCoxeterGroup(H3)
sage: img = GRH3.visualize_roots(range(7));img
Graphics object consisting of 18 graphics primitives

sage: J3 = CoxeterMatrix([[1,6,2],[6,1,3],[2,3,1]])
sage: GRJ3 = GeometricRepresentationCoxeterGroup(J3)
sage: img = GRJ3.visualize_roots(range(10));img
Graphics object consisting of 40 graphics primitives

sage: K3 = CoxeterMatrix([[1,7,2],[7,1,3],[2,3,1]])
sage: GRK3 = GeometricRepresentationCoxeterGroup(K3)
sage: img = GRK3.visualize_roots(range(10));img
Graphics object consisting of 55 graphics primitives
```

Rank 4 examples:

```
sage: A4 = CoxeterMatrix([[1,3,2,2],[3,1,3,2],[2,3,1,3],[2,2,3,1]])
sage: GRA4 = GeometricRepresentationCoxeterGroup(A4)
sage: img = GRA4.visualize_roots(range(4));img
Graphics3d Object

sage: C4t = CoxeterMatrix([[1,4,2,2],[4,1,3,2],[2,3,1,4],[2,2,4,1]])
sage: GRC4t = GeometricRepresentationCoxeterGroup(C4t)
sage: img = GRC4t.visualize_roots(range(6));img # long time
Graphics3d Object

sage: M = CoxeterMatrix([[1,oo,oo,oo],[oo,1,3,3],[oo,3,1,3],[oo,3,3,1]])
sage: GR = GeometricRepresentationCoxeterGroup(M)
sage: img = GR.visualize_roots(range(4));img # long time
Graphics3d Object
```

visualize_tits_cone

File: brocoli/geom_repr_class.py (starting at line 3021)

Return a graphical representation of the Tits cone up to weights of depth `depth`.

INPUT:

- `depth` – a non-negative integer
- `frame_color` – a color (default= `(1,0,0)` red); color of the edges
- `face_color` – a color (default= `(0, 0, 1)`); color of the faces
- `color_simplex` – a color to give the limit roots (default: `(0,1,0)`, green)
- `isotropic` – a boolean (default: `False`); whether to plot the isotropic cone
- `iso_color` – a color to give the isotropic cone (default: `(1, 1, 0)`, yellow)

OUTPUT:

A Graphic Object

EXAMPLES:

```
sage: from brocoli import *
sage: M1 = CoxeterMatrix([[1,4,4],[4,1,4],[4,4,1]])
sage: GR1 = GeometricRepresentationCoxeterGroup(M1)
sage: GR1.visualize_tits_cone(2)
Graphics object consisting of 13 graphics primitives
```

```
sage: M2 = CoxeterMatrix([[1,-2,-2],[-2,1,-2],[-2,-2,1]])
sage: GR2 = GeometricRepresentationCoxeterGroup(M2)
sage: GR2.visualize_tits_cone(2)
Graphics object consisting of 16 graphics primitives

sage: M3 = CoxeterMatrix([[1,oo,oo,oo],[oo,1,3,3],[oo,3,1,3],[oo,3,3,1]])
sage: GR3 = GeometricRepresentationCoxeterGroup(M3)
sage: GR3.visualize_tits_cone(3) # long time (5 seconds)
Graphics3d Object
```

visualize_weights(*list_depths*, *space=False*, *size=4*, *color=(0, 0, 1)*, *color_simplex=(0, 1, 0)*, *isotropic=True*, *iso_color=(1, 1, 0)*)

Return a graphical representation of weights. Some weights may not have an affine representation and will not be drawn.

INPUT:

- `list_depths` – a list of non-negative integers; give the depths of the weights to plot
- `space` – a boolean (default: `False`); whether to plot only space weights
- `size` – a non-negative integer (default: `4`); give the size of the point to draw
- `color` – a color to give the weights (default: `(1,0,0)`, blue)
- `color_simplex` – a color to give the limit roots (default: `(0,1,0)`, green)
- `isotropic` – a boolean (default: `True`); whether to plot the isotropic cone in the infinite case
- `iso_color` – a color to give the isotropic cone (default: `(1, 1, 0)`, yellow)

OUTPUT:

A Graphic Object

EXAMPLES:

Rank 2 examples:

```
sage: from brocoli import *
sage: A2 = CoxeterMatrix([[1,3],[3,1]])
sage: GRA2 = GeometricRepresentationCoxeterGroup(A2)
sage: GRA2.visualize_weights(range(3))
Graphics object consisting of 7 graphics primitives

sage: DiHy = CoxeterMatrix([[1,-2],[-2,1]])
sage: GRDiHy = GeometricRepresentationCoxeterGroup(DiHy)
sage: GRDiHy.visualize_weights(range(3))
Graphics object consisting of 10 graphics primitives
```

Rank 3 examples:

```
sage: A3 = CoxeterMatrix([[1,3,2],[3,1,3],[2,3,1]])
sage: GRA3 = GeometricRepresentationCoxeterGroup(A3)
sage: GRA3.visualize_weights(range(5))
Graphics object consisting of 15 graphics primitives

sage: M = CoxeterMatrix([[1,-10/9,oo],[-10/9,1,4],[oo,4,1]])
sage: GR = GeometricRepresentationCoxeterGroup(M)
sage: GR.visualize_weights(range(4))
Graphics object consisting of 31 graphics primitives
```

Rank 4 examples:

```
sage: A4 = CoxeterMatrix([[1,3,2,2],[3,1,3,2],[2,3,1,3],[2,2,3,1]])
sage: GRA4 = GeometricRepresentationCoxeterGroup(A4)
sage: GRA4.visualize_weights(range(7))  # Not all weights are drawn
Graphics3d Object

sage: C4t = CoxeterMatrix([[1,4,2,2],[4,1,3,2],[2,3,1,4],[2,2,4,1]])
sage: GRC4t = GeometricRepresentationCoxeterGroup(C4t)
sage: GRC4t.visualize_weights(range(5))
Traceback (most recent call last):
...
ValueError: The weights can not be visualized affinely; the bilinear form is␣
↪degenerate

sage: QF.<a> = QuadraticField(3/2)
sage: Cyl = CoxeterMatrix([[1,-a,2,2],[-a,1,3,2],[2,3,1,-a],[2,2,-a,1]])
sage: GRCyl = GeometricRepresentationCoxeterGroup(Cyl)
sage: GRCyl.visualize_weights(range(5))
Traceback (most recent call last):
...
ValueError: The weights can not be visualized affinely; the bilinear form is␣
↪degenerate

sage: M = CoxeterMatrix([[1,oo,oo,oo],[oo,1,3,3],[oo,3,1,3],[oo,3,3,1]])
sage: GR = GeometricRepresentationCoxeterGroup(M)
sage: GR.visualize_weights(range(3)) # long time
Graphics3d Object
sage: GR.visualize_weights(range(3),True)  # This one has less weights
Graphics3d Object
```

WARNING:

In certain cases, some weights have a sum of coordinates equal to zero, they are ignored by the plotting function.

weights

File: brocoli/geom_repr_class.py (starting at line 1836)

Return the weights of the representation of depth `depth`

INPUT:

- `depth` – a non-negative integer
- `exact` – a boolean (default = `True`); whether to give the bilinear in an exact base ring or not.

OUTPUT:

A set of weights of depth `depth`

EXAMPLES:

```
sage: from brocoli import *
sage: M = CoxeterMatrix([[1,4,oo,oo],[4,1,oo,oo],[oo,oo,1,-2],[oo,oo,-2,1]])
sage: GR = GeometricRepresentationCoxeterGroup(M)
sage: [len(GR.weights(i)) for i in range(3)]
[4, 4, 12]
```

Putting `exact=False` will give more weights as computations are not exact. This may be useful if we just want to draw weights:

```
sage: [len(GR.weights(i,False)) for i in range(3)] # not tested
[4, 16, 48]
```

A finite example:

```
sage: A3 = CoxeterMatrix([[1,3,2],[3,1,3],[2,3,1]])
sage: GR_A3 = GeometricRepresentationCoxeterGroup(A3)
sage: [len(GR_A3.weights(i)) for i in range(6)]
[3, 3, 4, 3, 1, 0]
```

brocoli.geom_repr_class.**affinely_project_vector**(*vect*, *projection_space*, *affine_basis*)

This function returns the affine normalization of the vector `vect` in the affine hyperplane orthogonal to the vector (1,...,1) according to the affine basis `affine_basis` of the projection space `projection_space`.

First, the vector is normalize to be a vector with sum of the coordinate equals to 1. If the sum of the coordinates is equal to 0, it returns an error. Then, the coefficients are used to give a linear combination of the `affine_basis` in the affine space $projection_space$.

For example, a 3-dimensional vector with have 3 coefficients which will be used to obtain a linear combination of 3 vectors, say in a 2-dimensional space.

INPUT:

- `vect` – vector; the vector that is to project
- `projection_space` – vector space; the affine space where to project
- `affine_basis` – list of vectors; the image of the basis of the vector space of `vect` under a projection to the affine space `projection_space`

OUTPUT:

A vector of the affine space `projection_space`

EXAMPLES:

```
sage: from brocoli import *
sage: PS = VectorSpace(RDF, 2)
sage: ab = regular_simplex_vertices(2)
sage: v = vector([1,2,3])
sage: affinely_project_vector(v,PS,ab)
(1.1666666666666665, 0.8660254037844386)
sage: UCF.<E> = UniversalCyclotomicField()
sage: u = vector(UCF, [2 + E(8) - E(8)^3, 1, E(8) - E(8)^3])
sage: affinely_project_vector(u,PS,ab)
(0.585786437626905, 0.4202659980740251)

sage: v = vector([1,-4,3])
sage: affinely_project_vector(v,PS,ab)
Traceback (most recent call last):
...
ValueError: The vector (1, -4, 3) does not have an affine image in the affine␣
↪basis
```

brocoli.geom_repr_class.**plot_simplex**(*size*, *color=(0, 1, 0)*)

This function returns a graphical element representing the regular simplex in dimension 2 or 3

INPUT:

- `size` – integer; the number of vertices of the simplex
- `color` – a rgb vector; the color of the simplex

OUTPUT:

A graphical object

EXAMPLES:

```
sage: from brocoli import *
sage: plot_simplex(2)
Graphics object consisting of 1 graphics primitive
sage: plot_simplex(3)
Graphics object consisting of 3 graphics primitives
sage: plot_simplex(5)
Traceback (most recent call last):
...
ValueError: The size of the simplex should be 2, 3 or 4
```

brocoli.geom_repr_class.**reflection_image**(*reflection_vector*, *element*, *bilin_form_matrix*)

Returns the image of the vector `element` by the reflection through the vector $reflection_vector$ with respect to the bilinear form described by the matrix `bilin_form_matrix`.

INPUT:

- `reflection_vector` – vector; a non-isotropic (pseudonorm 1)-vector to reflect with
- `element` – vector; a vector that will be reflected
- `bilin_form_matrix` – matrix; the matrix giving the bilinear form

OUTPUT:

The reflected vector

EXAMPLES:

```
sage: from brocoli import *
sage: bf = matrix([[1,-2],[-2,1]])
sage: reflection_image(vector([1,0]),vector([10,1]),bf)
(-6, 1)

sage: bf = matrix([[1,-1],[-1,1]])
sage: reflection_image(vector([1,0]),vector([1,1]),bf)
(1, 1)
```

brocoli.geom_repr_class.**regular_simplex_vertices**(*dim*)

This function returns the floating point approximation of vertices of a regular simplex of dimension `dim` having `dim+1` vertices

INPUT:

- `dim` : integer; the dimension of the desired simplex

OUTPUT:

A list of vectors, the vertices of a regular simplex

EXAMPLES:

```
sage: from brocoli import *
sage: regular_simplex_vertices(1)
[(0.0, 0.0), (1.0, 0.0)]
sage: regular_simplex_vertices(2)
[(0.0, 0.0), (2.0, 0.0), (1.0, 1.7320508075688772)]
sage: regular_simplex_vertices(3)
```

```
[(0.0, 0.0, 0.0),
 (2.0, 0.0, 0.0),
 (1.0, 1.7320508075688772, 0.0),
 (1.0, 0.5773502691896257, 1.6329931618554518)]
sage: regular_simplex_vertices(4)
Traceback (most recent call last):
...
NotImplementedError: dimension >=4
```

CHAPTER

THREE

BROCOLI - LIMIT ROOTS OF COXETER GROUPS (ETERNAL BETA VERSION)

This is a package to experiment with limit roots and related objects of geometric representations of Coxeter groups.

With this package you can compute the following:

- Roots
- Weights
- Elements of the group (as matrices)
- Bilinear form and its signature

For Lorentzian Coxeter groups:

- Elliptic, Parabolic and Hyperbolic elements
- limit roots coming from parabolic and hyperbolic elements

You can also visualize:

- Affine positive cone
- Isotropic cone
- roots
- weights
- limit roots
- Tits cone

For rank 3 Coxeter groups, you can obtain tikzpicture of the above structures.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

b

INDEX

S

T

V

W