(Toric) Geometry in Sage

Very long time ago

polytope.py (by William Stein) - basic interface to Polymake.

Polymake: not terribly difficult to build, but took half the time and half the size of the whole Sage to do it - no question about making Polymake a standard package.

Long time ago

lattice_polytope.py (by A.N.) with the original goal to use PALP conveniently and interactively on single polytopes.

PALP is small and easy to build, was included right away.

Used via files/system calls.

```
%time
simplex = LatticePolytope([(1,0), (0,1), (-1,-1)])
simplex
```

Δ^2_0

```
CPU time: 0.04 s,  Wall time: 0.07 s
```

```
%time
simplex.poly_x("p")
```

```
2 4  Points of P
    1    0   -1    0
    0    1   -1    0
```

```
CPU time: 0.01 s,  Wall time: 0.02 s
```

```
%time
simplex.points()
```

$$\left(\begin{array}{rrrr} 1 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 \end{array}\right)$$

```
CPU time: 0.00 s,  Wall time: 0.00 s
```

```
simplex.nef_x("-N -P -p")
```

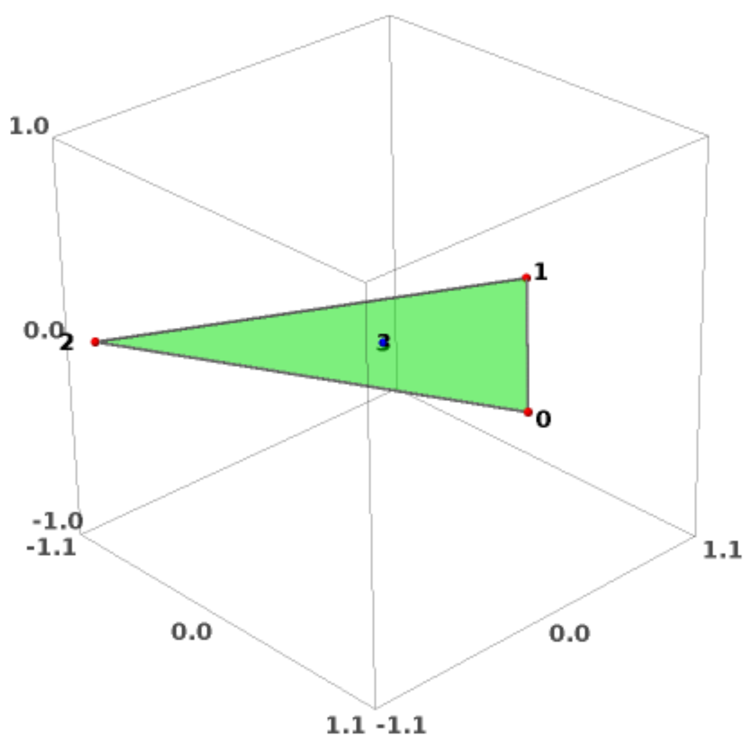
```
M:10 3 N:4 3  codim=2 #part=1
 P:0 V:2       0sec  0cpu
np=0 d:0 p:1    0sec     0cpu
```

```
simplex.nef_partitions()
```

[Nef-partition $\{0, 1\} \sqcup \{2\}$]

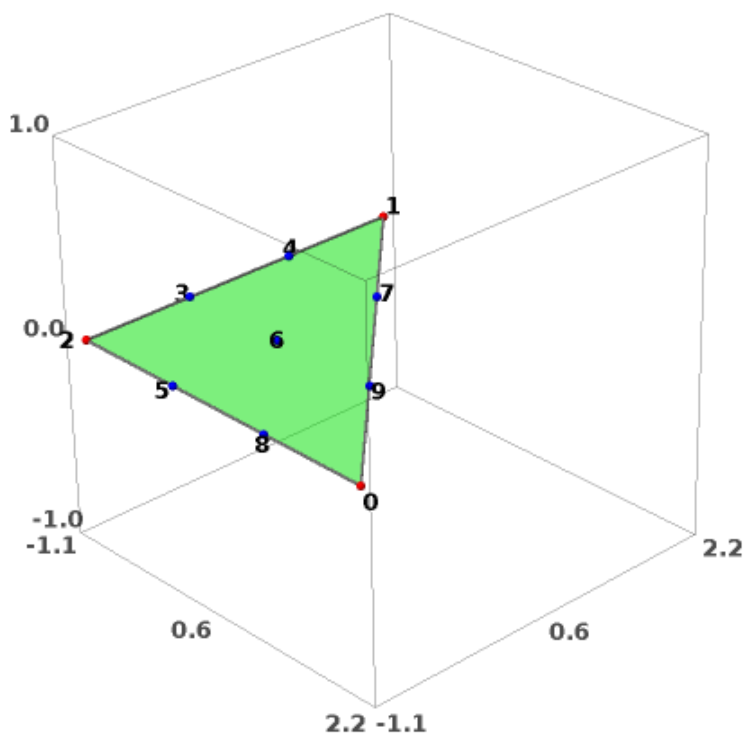
```
simplex.plot3d()
```

Sleeping... Make Interactive




```
simplex.polar().plot3d()
```

Sleeping... Make Interactive



Still long time ago

polyhedra.py (by Marshal Hampton) - file-based (?) interface to cdd

Allows working with generic polyhedra, i.e. non-integer points, unbounded.

Rewritten by Volker Braun. Rewrite is now gone.

A few years ago

Cones for toric geometry - not special cases of polyhedra.

Toric lattices, fans, fan morphisms. Written and cross-reviewed by Volker Braun and A.N.

Most "polyhedral operations" use library interface to PPL (by Volker Braun).

```
N = ToricLattice(2)
N
```

N

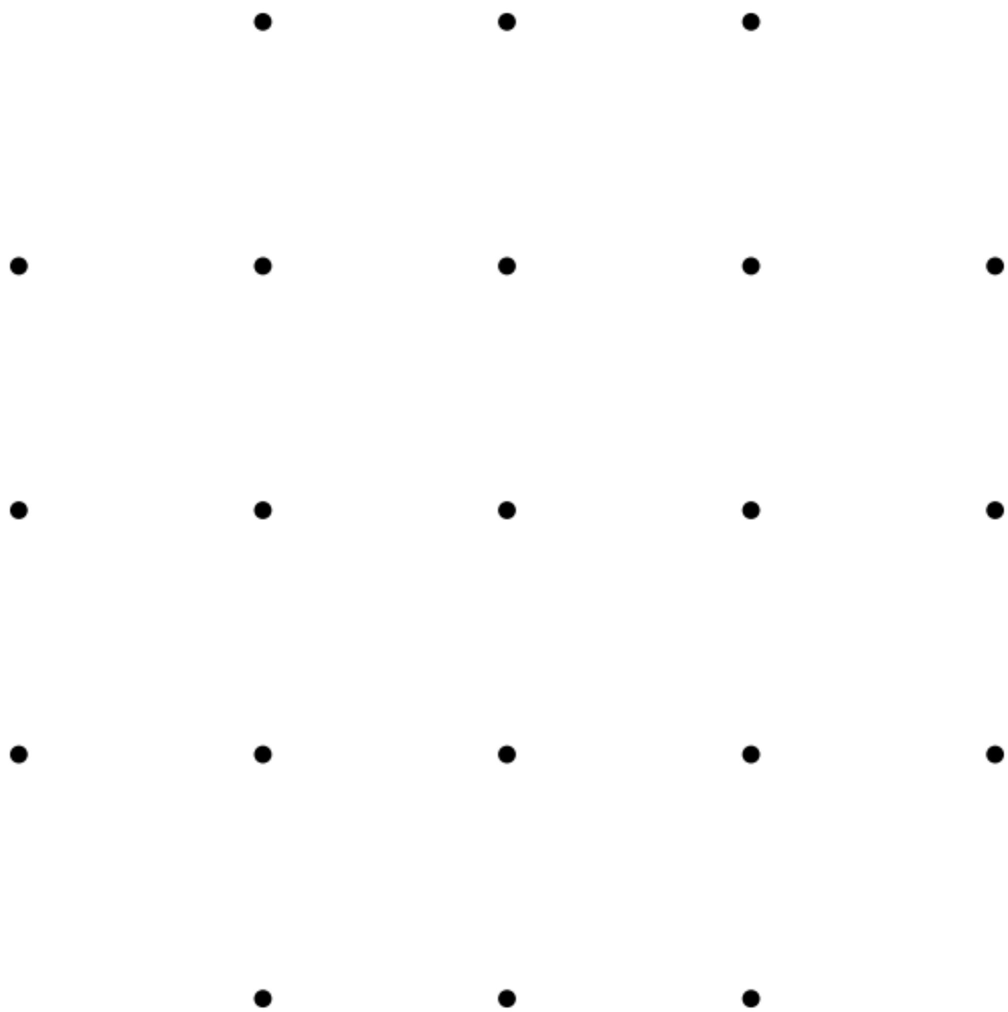
```
M = N.dual()
M
```

M

```
M.dual()
```

N

```
M.plot()
```



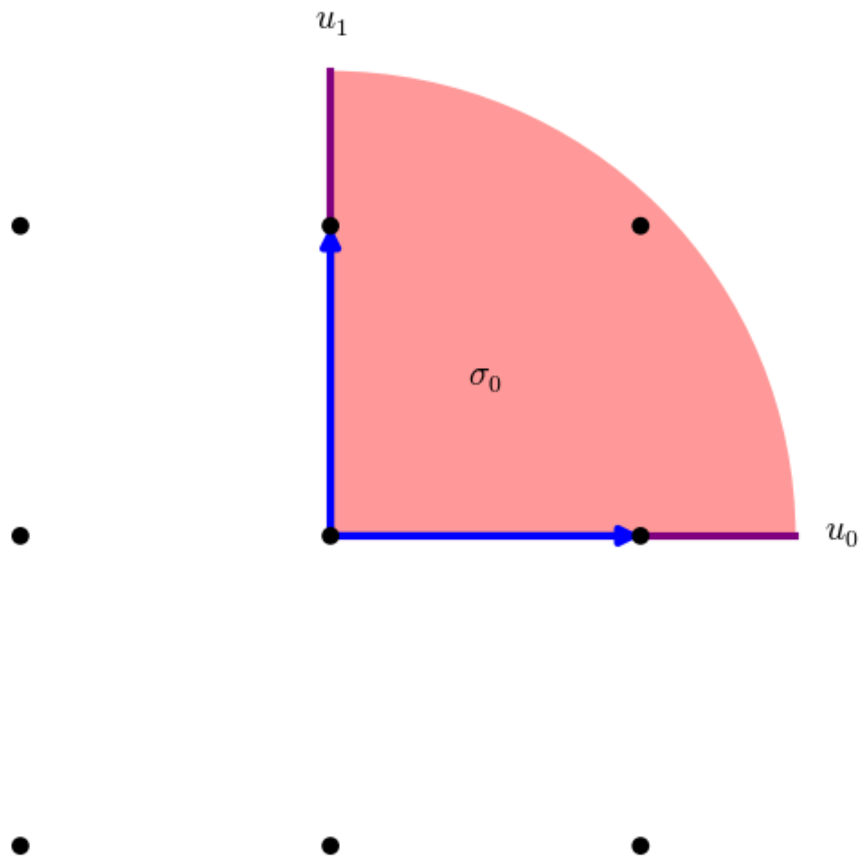
```
sigma0 = Cone([(1/2,0), (0,2)], lattice=N)
sigma0
```

σ^2

```
sigma0.rays()
```

$\left((1,\,0)_N, (0,\,1)_N\right)_N$

```
sigma0.plot()
```



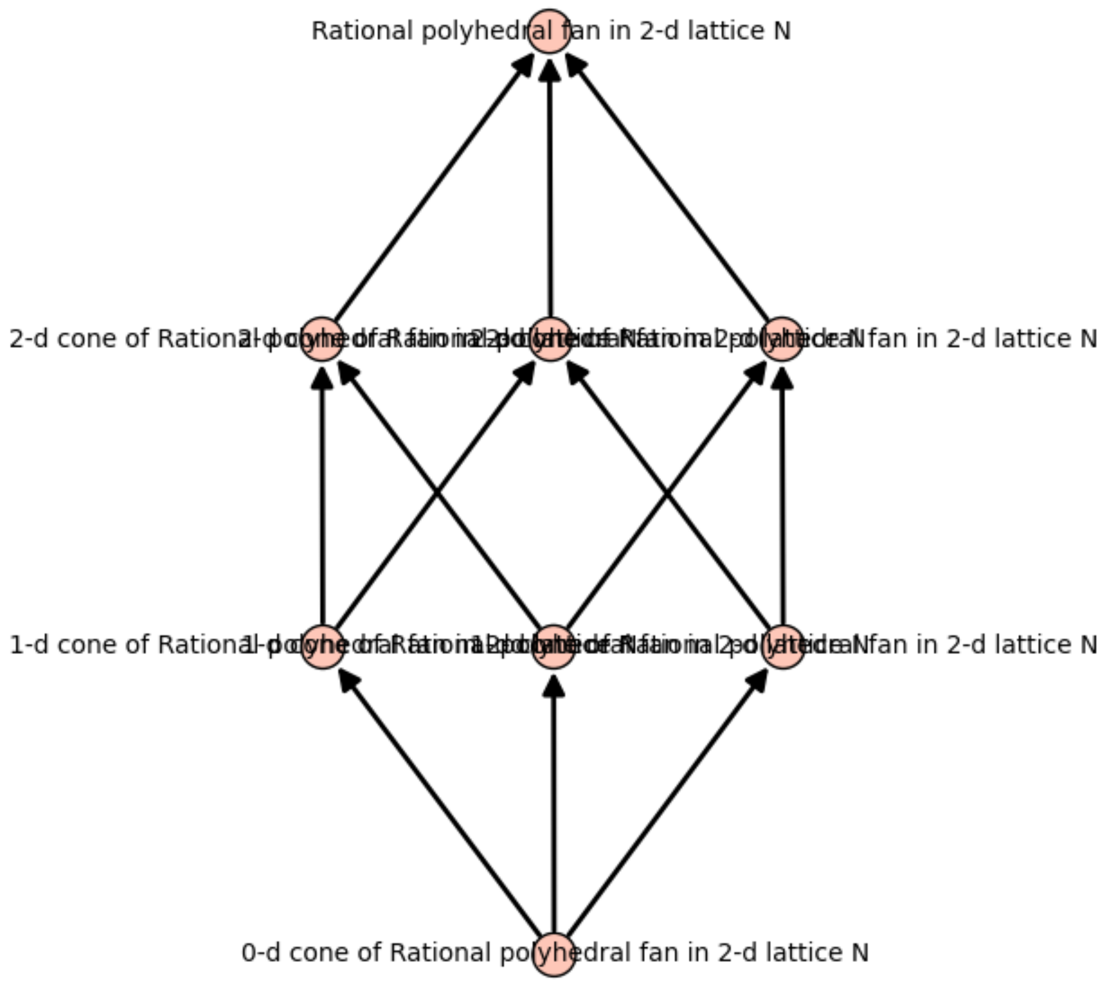

```
sigma1 = Cone([(0,1), (-1, -1)])
sigma2 = Cone([(-1,-1), (1, 0)])
Sigma = Fan([sigma0, sigma1, sigma2])
Sigma
```

Σ^2

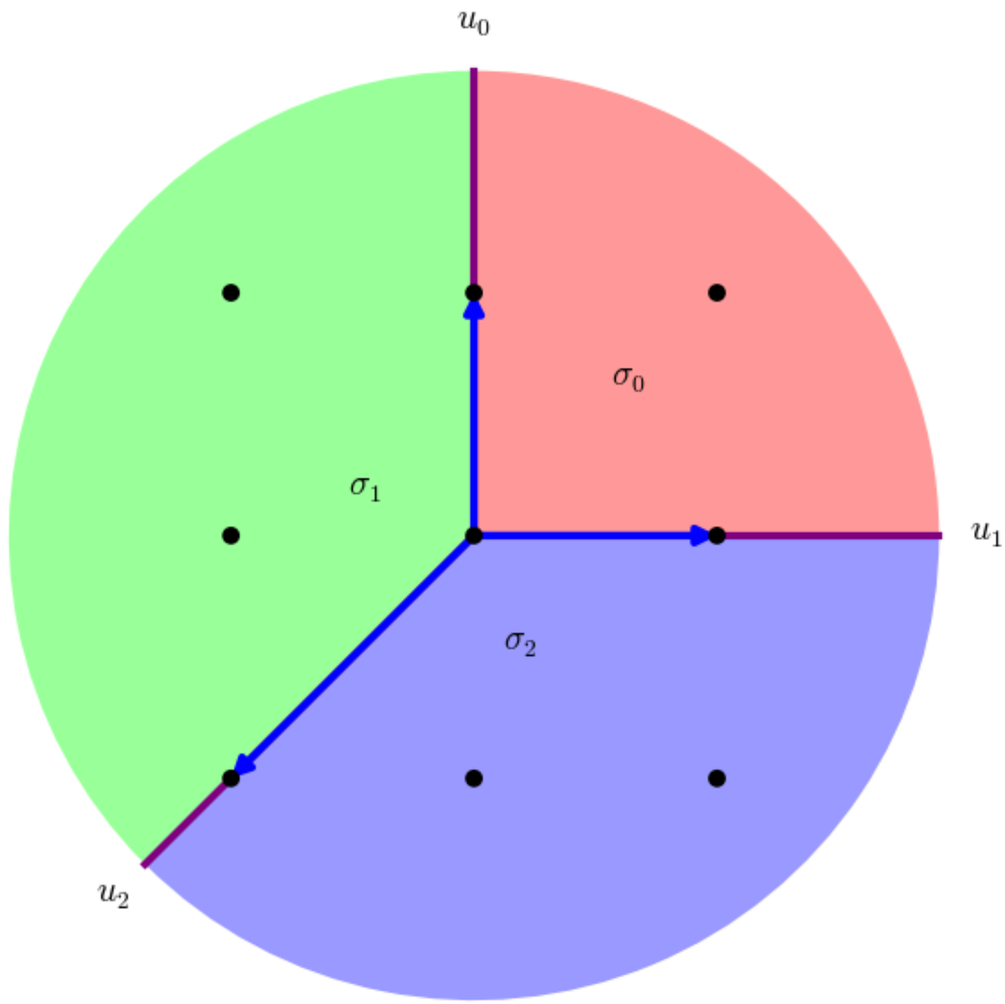
```
Sigma.rays()
```

$\left((0,\, 1)_N, (1,\, 0)_N, (-1,\, -1)_N\right)_N$

```
Sigma.cone_lattice().plot()
```

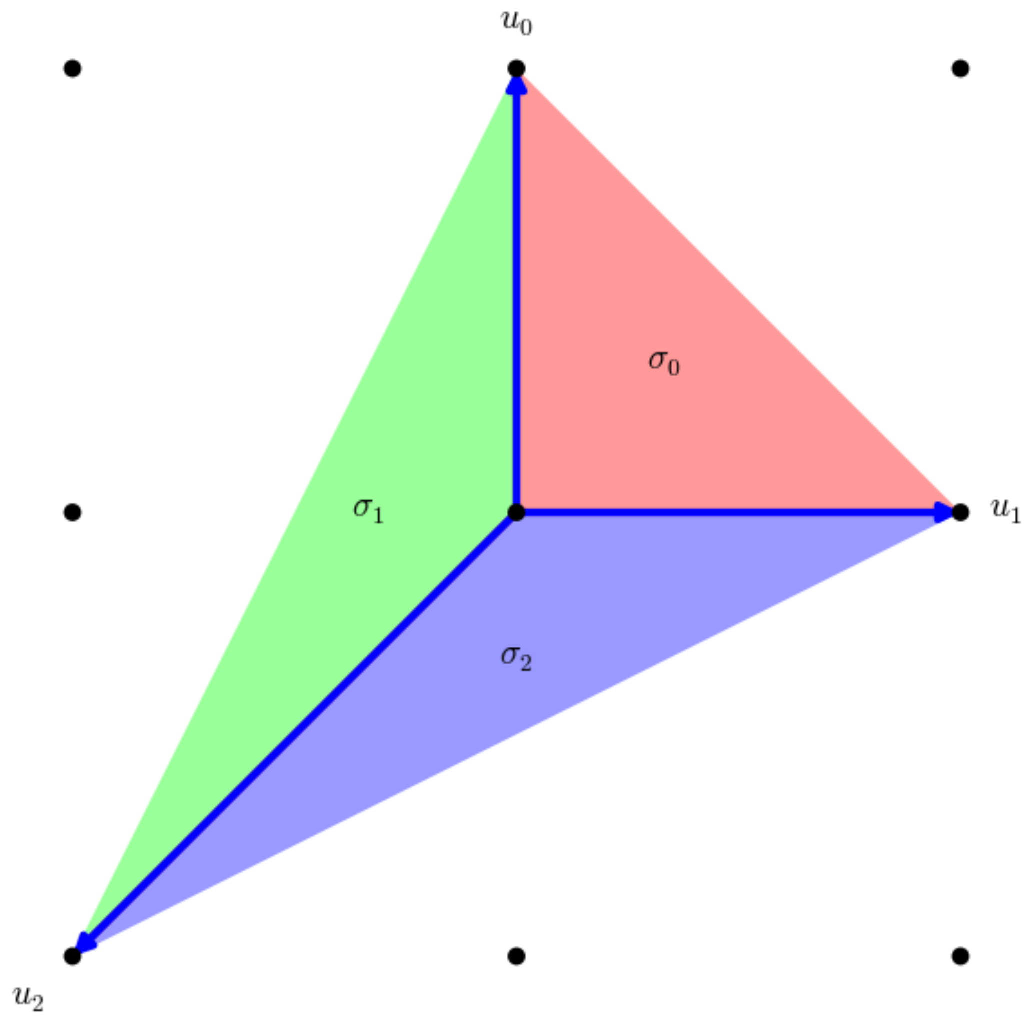

Rational polyhedral fan in 2-d lattice N
0-d cone of Rational polyhedral fan in 2-d lattice N

```
Sigma.plot()
```




```
Sigma.plot(mode="generators")
```

```
/home/novoselt/sage-5.12.beta5/local/lib/python2.7/site-packages/sag\
e/geometry/toric_plotter.py:634: DeprecationWarning: use the option
'base_ring' instead of 'field'
See http://trac.sagemath.org/11634 for details.
  result += Polyhedron(vertices=vertices, field=RDF).render_solid(
```




```
Sigma == FaceFan(simplex)
```

False

```
Sigma.is_equivalent(FaceFan(simplex))
```

True

```
Sigma.is_isomorphic(FaceFan(simplex))
```

True

```
Sigma.isomorphism(FaceFan(simplex))
```

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} : \Sigma^2 \to \Sigma^2$$

```
Sigma.is_equivalent(NormalFan(simplex.polar()))
```

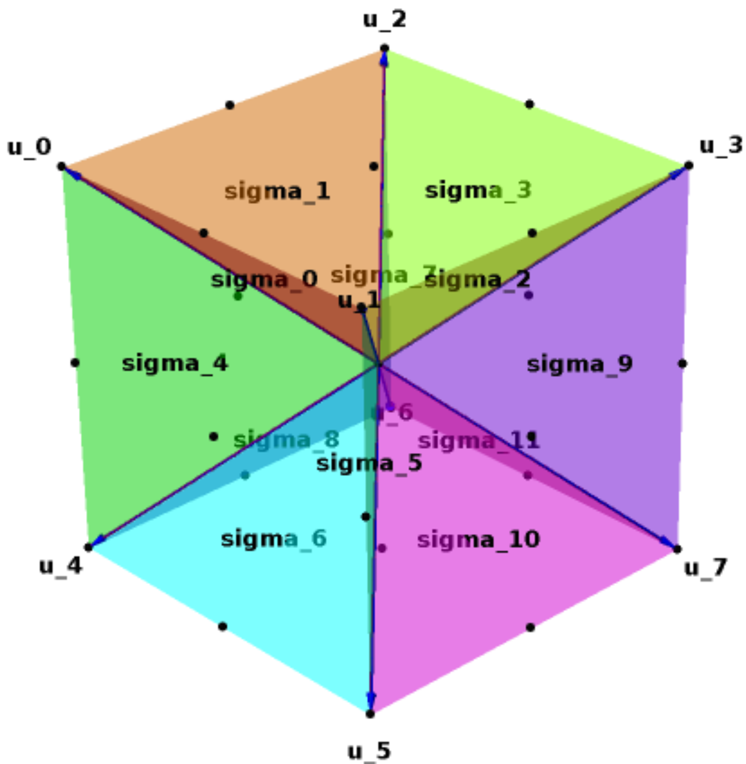
True

```
fm = FanMorphism(2*identity_matrix(2), Fan([sigma0]), Sigma)
fm
```

$$\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} : \Sigma^2 \to \Sigma^2$$

```
NormalFan(lattice_polytope.octahedron(3)).plot(mode="generators")
```

Sleeping... Make Interactive



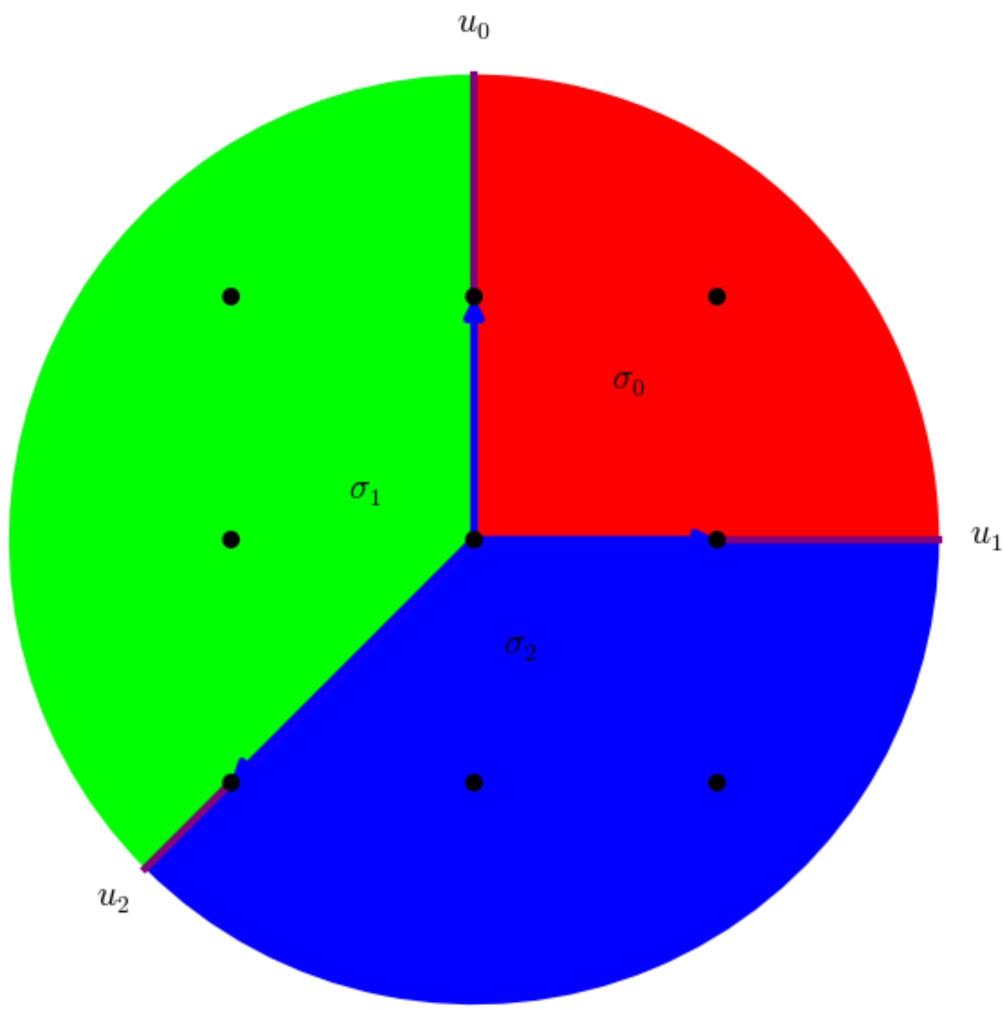

```
for i in sorted(toric_plotter.options().items()):
    print i
```

```
('font_size', 14)
('generator_color', 'blue')
('generator_thickness', None)
('generator_zorder', -3)
('label_color', 'black')
('label_zorder', -1)
('lattice_filter', None)
('mode', 'round')
('point_color', 'black')
('point_size', None)
('point_zorder', -2)
('radius', None)
('ray_color', 'purple')
('ray_label', 'u')
('ray_thickness', 3)
('ray_zorder', -4)
```

```
('show_generators', True)
('show_lattice', None)
('show_rays', True)
('show_walls', True)
('wall_alpha', 0.4)
('wall_color', 'rainbow')
('wall_label', '\\sigma')
('wall_zorder', -5)
('xmax', None)
('xmin', None)
('ymax', None)
('ymin', None)
('zmax', None)
('zmin', None)
```

```
toric_plotter.options(wall_alpha=1)
```

```
Sigma.plot()
```




```
toric_plotter.reset_options()
```

Recent additions by Volker Braun

New rewrite of generic polyhedra allowing different backends: currently PPL and cddlib.

Triangulations (with and without optional package TOPCOM).

Integral points.

ppl_lattice_polytope.py - not as featurefull as LatticePolytope but much faster.

Future (this week?..)

Clean up LatticePolytope code to make backend change possible.

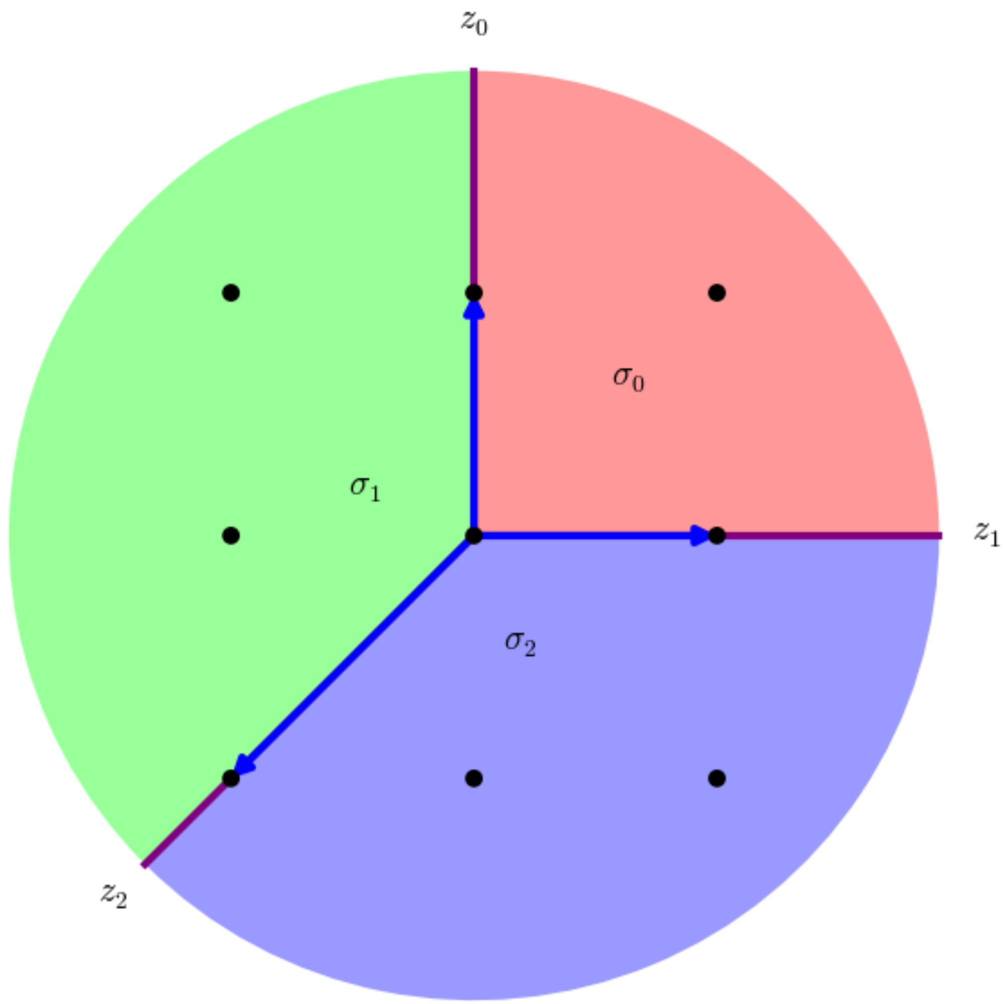
Unify its interface with Cone and Fan, including plotting.

Toric Geometry

```
P2 = ToricVariety(Sigma)
P2
```

$\mathbb{X}_{\Sigma^2}$

```
P2.plot()
```




```
P2.coordinate_ring()
```

$\mathbf{Q}[z_0, z_1, z_2]$

Default ring is $\mathbb{Q}$, assumed ring is $\mathbb{C}$, arbitrary one should work in principle.

```
P2.inject_variables()
```

```
Defining z0, z1, z2
```

```
P2.subscheme(z0 + z1^2)
```

```
Traceback (click to the left of this block for traceback)
...
ValueError: z1^2 + z0 is not homogeneous on 2-d toric variety
covered by 3 affine patches!
```

```
P2.subscheme(z0 + z1)
```

Closed subscheme of $\mathbb{X}_{\Sigma^2}$ defined by $z_0 + z_1$

```
P2.subscheme([z0, z1, z2]).dimension()
```

-1

```
A2 = AffineToricVariety(sigma0)
f = A2.hom(fm, P2)
f
```

```
Scheme morphism:
  From: 2-d affine toric variety
  To:   2-d toric variety covered by 3 affine patches
  Defn: Defined by sending Rational polyhedral fan in 2-d lat
```

```
f.as_polynomial_map()
```

```
Scheme morphism:
  From: 2-d affine toric variety
  To:   2-d toric variety covered by 3 affine patches
  Defn: Defined on coordinates by sending [z0 : z1] to
        [z1^2 : z0^2 : 1]
```

```
for factor in reversed(f.factor()):
    factor.as_polynomial_map()
```

```
Scheme morphism:
  From: 2-d affine toric variety
  To:   2-d affine toric variety
  Defn: Defined on coordinates by sending [z0 : z1] to
        [z0^2 : z1^2]

Scheme morphism:
  From: 2-d affine toric variety
  To:   2-d toric variety covered by 3 affine patches
  Defn: Defined on coordinates by sending [z0 : z1] to
        [z1 : z0 : 1]

Scheme morphism:
  From: 2-d toric variety covered by 3 affine patches
  To:   2-d toric variety covered by 3 affine patches
  Defn: Defined on coordinates by sending [z0 : z1 : z2] to
        [z0 : z1 : z2]
```

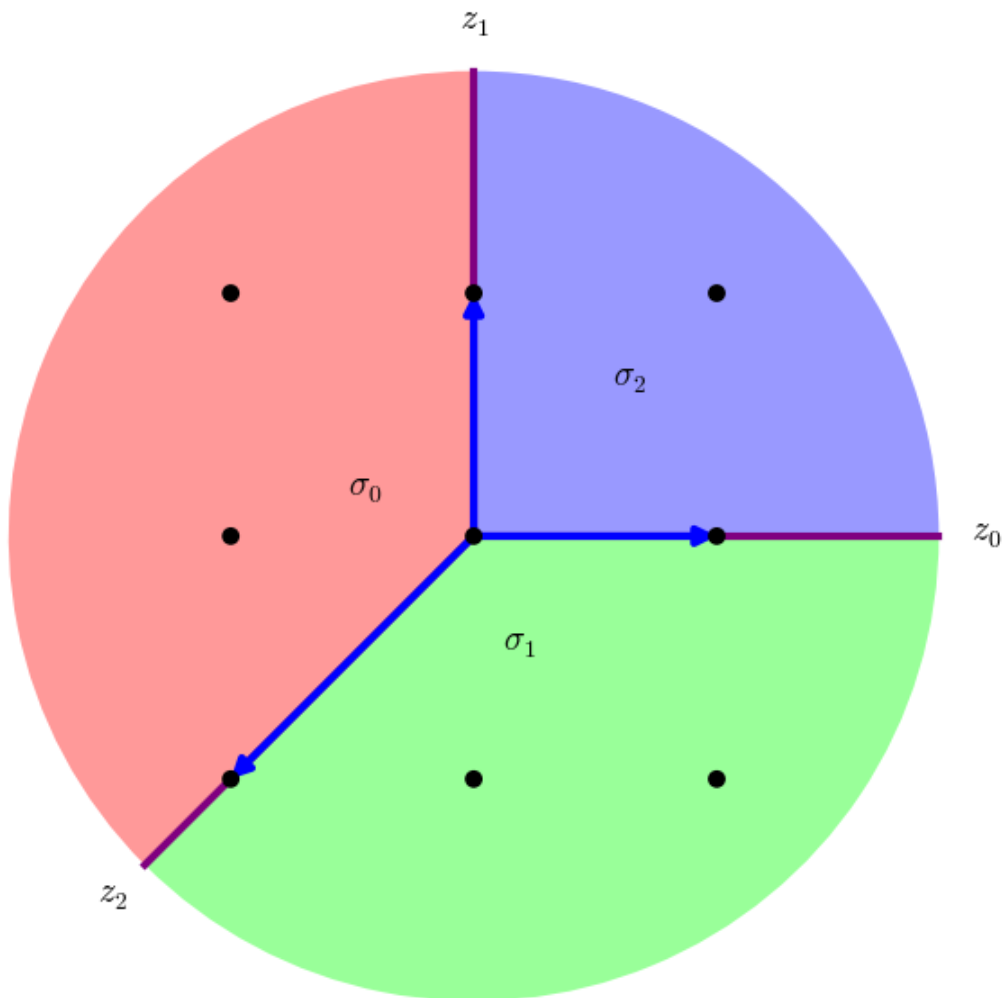
```
print P2
```

```
2-d toric variety covered by 3 affine patches
```

```
P2 = CPRFanoToricVariety(Delta_polar=simplex)
print P2
```

```
2-d CPR-Fano toric variety covered by 3 affine patches
```

```
P2.plot()
```




```
P2.fan().rays().column_matrix()
```

$$\begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \end{pmatrix}$$

```
simplex.vertices()
```

$$\begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \end{pmatrix}$$

```
AH = P2.anticanonical_hypersurface()
AH
```

Closed subscheme of $\mathbb{P}_{\Delta^2}$ defined by $a_0 z_0^3 + a_1 z_1^3 + a_6 z_0 z_1 z_2 + a_2 z_2^3$

```
simplex.polar().point(6)
```

$(0, 0)$

```
AH.ambient_space() is P2
```

False

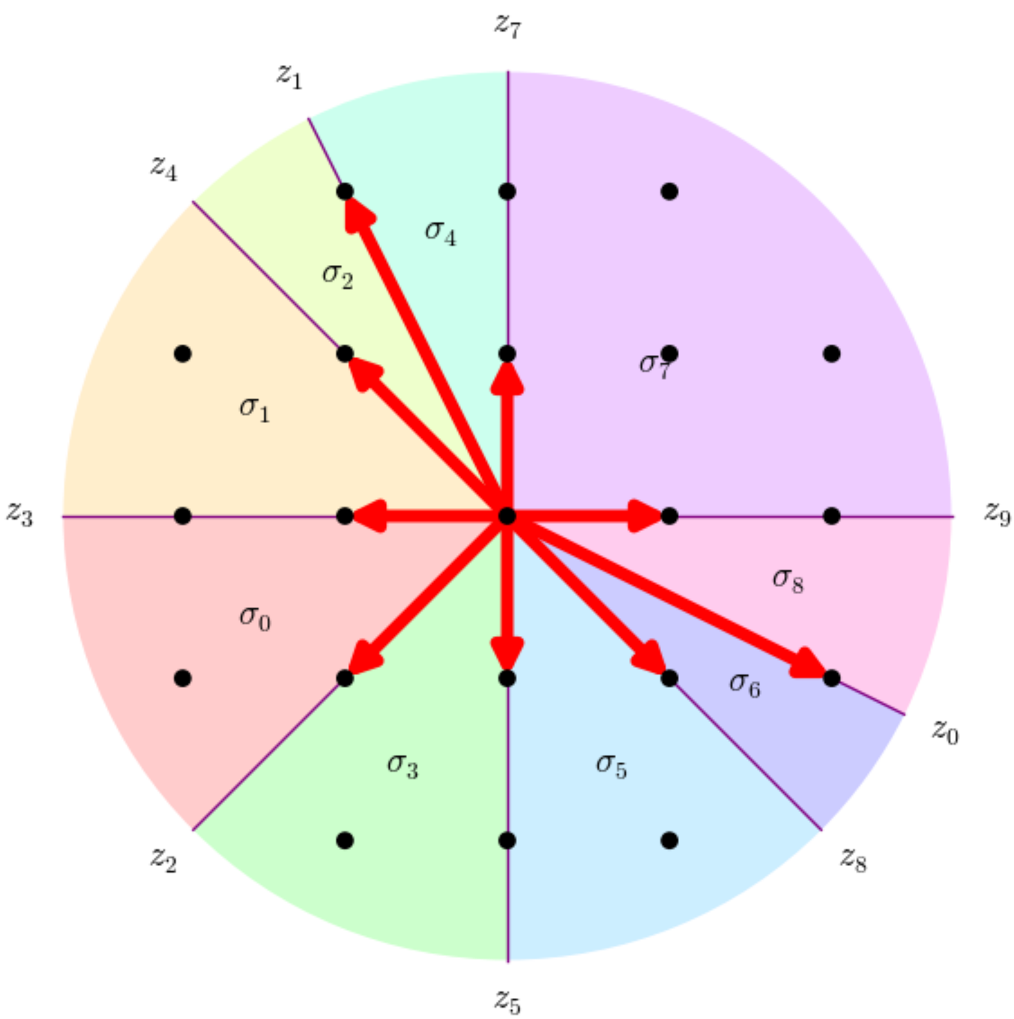
```
AH.ambient_space().coordinate_ring()
```

$\mathrm{Frac}(\mathbf{Q}[a_0, a_1, a_2, a_6])[z_0, z_1, z_2]$

```
P2.anticanonical_hypersurface(monomial_points="all")
```

Closed subscheme of $\mathbb{P}_{\Delta^2}$ defined by $a_0 z_0^3 + a_9 z_0^2 z_1 + a_7 z_0 z_1^2 + a_1 z_1^3 + a_8 z_0^2$

```
P2P = CPRFanoToricVariety(Delta=simplex, coordinate_points="all")
P2P.plot(generator_color="red", wall_alpha=0.2, ray_thickness=1,
generator_thickness=5)
```




```
P2P.anticanonical_hypersurface(monomial_points="all")
```

Closed subscheme of $\mathbb{P}_{\Delta^2_0}$ defined by $a_2 z_2^3 z_3^2 z_4 z_5^2 z_8 + a_1 z_1^3 z_3 z_4^2 z_7^2 z_9 + a_3 z_0 z$

```
D = 1/2*P2P.divisor(prod(P2P.gens()))
D
```

$\frac{1}{2}\,\mathrm{V}(z_0) + \frac{1}{2}\,\mathrm{V}(z_1) + \frac{1}{2}\,\mathrm{V}(z_2) + \frac{1}{2}\,\mathrm{V}(z_3) + \frac{1}{2}\,\mathrm{V}(z_4) + \frac{1}{2}\,\mathrm{V}(z_5) + \frac{1}{2}\,\mathrm{V}(z_7) +$

```
D.is_Cartier()
```

False

```
D.is_QQ_Weil()
```

True

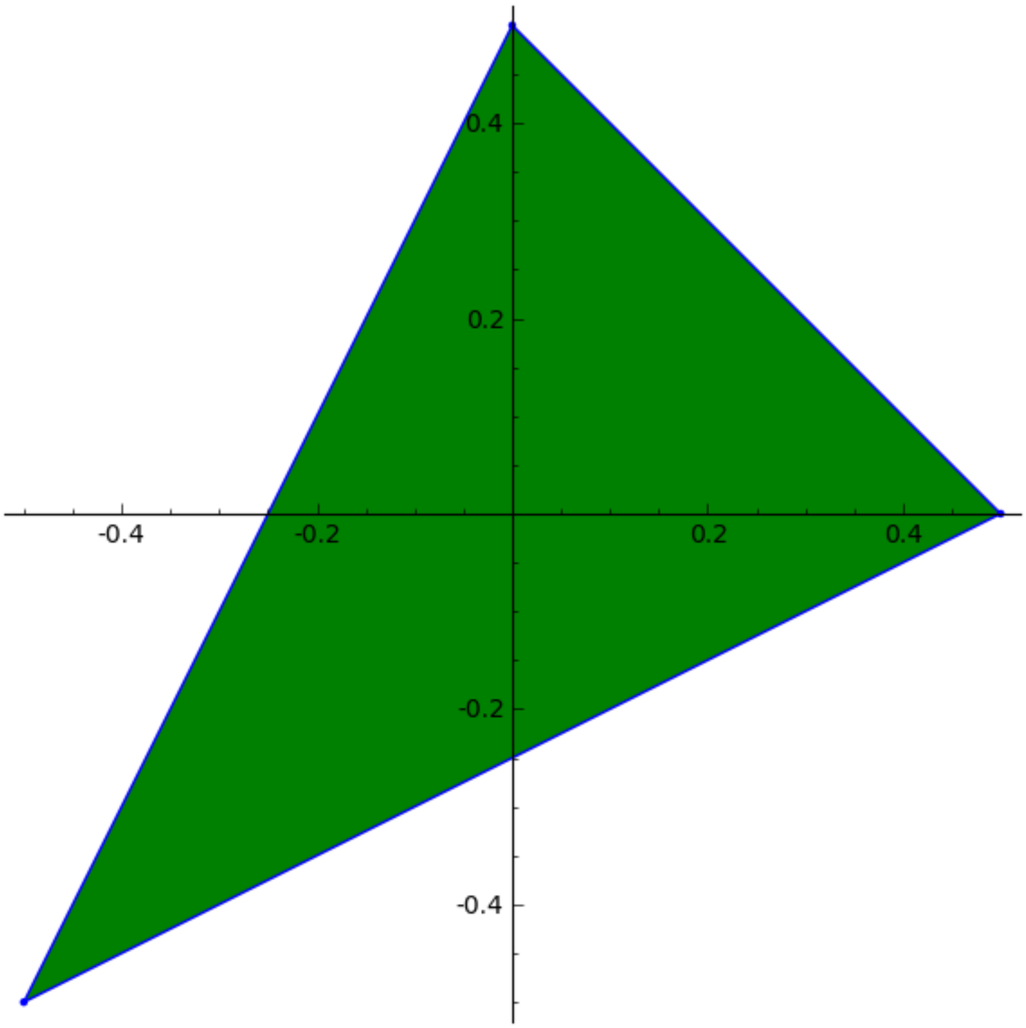
```
D.cohomology_class()
```

$$\left[\frac{3}{2}\, z_{2} + z_{3} + \frac{1}{2}\, z_{4} + z_{5} + \frac{1}{2}\, z_{8}\right]$$

```
P = D.polyhedron()
P
```

```
A 2-dimensional polyhedron in QQ^2 defined as the convex hull
```

```
P.plot()
```




```
P2.Stanley_Reisner_ideal()
```

$(z_{0} z_{1} z_{2})\mathbf{Q}[z_{0}, z_{1}, z_{2}]$

```
R = P2.coordinate_ring()
prod(R.ideal([f for f, m in e.factor()]) for e in
P2.Stanley_Reisner_ideal().gens())
```

$(z_{2}, z_{1}, z_{0})\mathbf{Q}[z_{0}, z_{1}, z_{2}]$

```
P1xP1 = toric_varieties.P1xP1()
```

```
P1xP1.Stanley_Reisner_ideal()
```

$(st, xy)\mathbf{Q}[s, t, x, y]$

```
R = P1xP1.coordinate_ring()
prod(R.ideal([f for f, m in e.factor()]) for e in
P1xP1.Stanley_Reisner_ideal().gens())
```

$(ty, tx, sy, sx)\mathbf{Q}[s, t, x, y]$

```
P1xP1.fan().primitive_collections()
```

$[\texttt{frozenset([0, 1])}, \texttt{frozenset([2, 3])}]$

```
P2P.Stanley_Reisner_ideal()
```

$(z_0 z_2, z_1 z_2, z_2 z_4, z_2 z_7, z_2 z_8, z_2 z_9, z_0 z_3, z_1 z_3, z_3 z_5, z_3 z_7, z_3 z_8, z_3 z_9, z_0 z_4, z_4 z_5$

```
raise RuntimeError("You are out of your RAM!")
R = P2P.coordinate_ring()
prod(R.ideal([f for f, m in e.factor()]) for e in
P2P.Stanley_Reisner_ideal().gens())
```

```
Traceback (click to the left of this block for traceback)
...
RuntimeError: You are out of your RAM!
```