Toric polymake tutorial

This tutorial will go over the basics of polymake and then focus on the `application "fulton"` for toric varieties.

Basics

First let us have a look at how to work with matrices, vectors, etc.

In [1]:
```
$A = new Matrix<Rational>([[1,2,3],[4,5,6]]);
$v = new Vector<Rational>([1,2,3]);
print $A,"\n";
print $A * $v,"\n\n";
print $A * transpose($A),"\n";
$S = new Set<Integer>([1,2,3]);
$S2 = new Set<Integer>([2,3,4]);
print $S * $S2,"\n";
print $S + $S2,"\n";
print $S - $S2,"\n";
```

Out[1]:
```
1 2 3
4 5 6

14 32

14 32
32 77

{2 3}
{1 2 3 4}
{1}
```

Now we use polymakes `cube` command to make a polytope that we will use for the first part of the demo. To see what this command does, use the built-in help. In the terminal, you can use the `F1` key instead.

In [2]:
```
$c = cube(2,1,0);
help "cube";
```

Out[2]:
```
functions/Producing regular polytopes and their g
eneralizations/cube:
cube<Scalar>(d; x_up, x_low, Options) -> Polytope
<Scalar>

 Produce a d-dimensional cube.
 Regular polytope corresponding to the Coxeter gr
oup of type B<sub>d-1</sub> = C<sub>d-1</sub>.

 The bounding hyperplanes are x<sub>i</sub> <= x_
up and x<sub>i</sub> >= x_low.

Type Parameters:
  Scalar Coordinate type of the resulting polytop
e.  Unless specified explicitly, deduced from the
type of bound values, defaults to Rational.

Arguments:
  Int d the dimension
  Scalar x_up upper bound in each dimension
```

If we have a variable we can ask for its datatype in the following way:

In [3]:
```
print $c->type->full_name;
```

Out[3]:
```
Polytope<Rational>
```

One of the main principles of polymake is lazy evaluation. This means that properties are not computed when they are not needed and furthermore they are cached after computation. To get a list of properties that have already been computed use the `->properties()` command. To only get a list of the property names, use `->list_properties()`.

In [4]:
```
$c->properties();
print "---------------\n". join("\n", $c->list_prop
```

Out[4]:
```
name: c
type: Polytope<Rational>
description: cube of dimension 2


CONE_AMBIENT_DIM
3

CONE_DIM
3

FACETS
(3) (1 1)
1 -1 0
(3) (2 1)
1 0 -1


AFFINE_HULL


VERTICES_IN_FACETS
{0 2}
{1 3}
{0 1}
{2 3}


BOUNDED
1
---------------
CONE_AMBIENT_DIM
CONE_DIM
FACETS
AFFINE_HULL
VERTICES_IN_FACETS
BOUNDED
```

Using the `->` arrow operator, we can trigger computation of properties. For example we can use it to compute the vertices and facets of our polytope.

In [5]:
```
$V = $c->VERTICES;
print $V;
```

Out[5]:
```
1 0 0
1 1 0
1 0 1
1 1 1
```

In [6]:
```
$F = $c->FACETS;
print $F;
```

Out[6]:
```
(3) (1 1)
1 -1 0
(3) (2 1)
1 0 -1
```

The matrix `$F` is a sparse matrix. To fill in zeros, we can use `dense`.

In [7]:
```
print $F->type->full_name, "\n";
print dense($F);
```

Out[7]:
```
SparseMatrix<Rational, NonSymmetric>
0 1 0
1 -1 0
0 0 1
1 0 -1
```

In [8]:
```
$facet = $F->row(0);
print $facet * transpose($V);
```

Out[8]:
```
0 1 0 1
```

In [9]:
```
print $F * transpose($V);
```

Out[9]:
```
0 1 0 1
1 0 1 0
0 0 1 1
1 1 0 0
```

In [10]:
```
$c->VISUAL;
```

Out[10]:
```
polymake: used package threejs
   Three.js is a lightweight cross-browser JavaSc
ript library/API used to create and display anima
ted 3D computer graphics on a Web browser.
   See http://github.com/mrdoob (http://github.co
m/mrdoob) for the source code.
```

In [11]:
```
$c->HASSE_DIAGRAM->VISUAL;
```

Out[11]:
```
polymake: used package tikz
  TikZ, a graphical toolkit for LaTeX.
  http://pgf.sourceforge.net. (http://pgf.sourcef
orge.net.)
```

In [12]:
```
print "Reflexive? ", $c->REFLEXIVE, "\n";
print "Smooth? ", $c->SMOOTH, "\n";
print "Normal? ", $c->NORMAL, "\n";
```

Out[12]:
```
polymake: used package lrs
  Implementation of the reverse search algorithm
of Avis and Fukuda.
  Copyright by David Avis.
  http://cgm.cs.mcgill.ca/~avis/C/lrs.html (http:
//cgm.cs.mcgill.ca/~avis/C/lrs.html)

Reflexive?
Smooth? 1
polymake: used package libnormaliz
  Normaliz is a tool for computations in affine m
onoids, vector configurations, lattice polytopes,
and rational cones.
  Copyright by Winfried Bruns, Bogdan Ichim, Chri
stof Soeger.
  http://www.math.uos.de/normaliz/ (http://www.ma
th.uos.de/normaliz/)

Normal? 1
```

In [13]:
```
help "SMOOTH";
```

Out[13]:
```
objects/Polytope/properties/Lattice points in con
es/SMOOTH:
property SMOOTH : Bool
 Only defined for Polytope::Lattice.
 The polytope is smooth if the associated project
ive variety is smooth; the determinant of the edg
e directions is +/-1 at every vertex.
```

In [14]:
```
help "DIM";
```

Out[14]:
```
There are 4 help topics matching 'DIM':
-------------------
1: objects/Cone/methods/Geometry/DIM:
DIM() -> Int

 returns the geometric dimension of the cone (inc
luding the lineality space)
 for the dimension of the pointed part ask for CO
MBINATORIAL_DIM

Returns Int

-------------------
2: objects/Polytope/methods/Geometry/DIM:
DIM() -> Int

 returns the dimension of the polytope

Returns Int

-------------------
3: objects/Symmetry/QuotientSpace/properties/Comb
inatorics/DIM:
property DIM : Int
 The dimension of the quotient space, defined to
be the dimension of the polytope.

-------------------
4: objects/PointConfiguration/methods/Geometry/DI
M:
DIM() -> Int

 Affine dimension of the point configuration.
 Similar to Polytope::DIM.

Returns Int
```

The `save` and `load` commands allow one to write and read xml files.

In [15]:
```
save($c, "square");
```

In [16]:
```
$cc = load("square.poly");
print join("\n", $cc->list_properties()), "\n";
```

Out[16]:
```
CONE_AMBIENT_DIM
CONE_DIM
FACETS
AFFINE_HULL
VERTICES_IN_FACETS
BOUNDED
POINTED
FEASIBLE
N_FACETS
FULL_DIM
N_VERTICES
VERTICES
LINEALITY_SPACE
LINEALITY_DIM
COMBINATORIAL_DIM
DUAL_GRAPH
GRAPH
HASSE_DIAGRAM
MONOID_GRADING
CENTERED
FAR_FACE
LATTICE
GORENSTEIN
GORENSTEIN_INDEX
GORENSTEIN_VECTOR
REFLEXIVE
SIMPLICIAL
SIMPLE
SMOOTH
HILBERT_BASIS_GENERATORS
N_HILBERT_BASIS
LATTICE_POINTS_GENERATORS
N_LATTICE_POINTS
NORMAL
```

For many computations there are different algorithms available. Some come from external software, some are built in. The prefer_now command will set the preferred algorithm for the current line. For a permanent preferrence use prefer .

In [17]:
```
help "prefer_now";
$r=rand_sphere(3,1000,seed=>1); prefer_now("beneath
$r=rand_sphere(3,1000,seed=>1); prefer_now("cdd");
$r=rand_sphere(3,1000,seed=>1); prefer_now("lrs");
```

Out[17]:
```
functions/Basic/prefer_now:
prefer_now(label_expression)

 The same as prefer, but does not store the chang
es persistently.  The lifespan of the
 new preference order lasts up to the end of the
current user cycle
 (that is, end of user script or interpretation o
f the current input line in interactive mode).

Arguments:
  String label_expression
1996
1996
1996
```

In [18]:
```
help "beneath_beyond";
```

Out[18]:
```
preferences/Convex hull computation/beneath_beyon
d:
 Use the sequential (beneath-beyond) convex hull
algorithm. It performs well at lower dimensions
 and produces a triangulation of the polytope as
a byproduct.
```

In [19]:
```
help "lrs";
```

Out[19]:
```
preferences/Convex hull computation/lrs:
 Use the reverse search method, as implemented in
wiki:external_software#lrslib.
```

Computation are done via rules that compute a set of output properties from a set of input properties. To arrive at a desired property, rules are chained together until one gets a path from the given properties. To see which rules will be applied use the `get_schedule($)` command.

In [20]:
```
$c = cube(3);
print join("\n", $c->list_properties()),"\n\n";
$s = $c->get_schedule("HASSE_DIAGRAM");
print $s, "\n\n";
print join("\n", $s->list), "\n";
```

Out[20]:
```
CONE_AMBIENT_DIM
CONE_DIM
FACETS
AFFINE_HULL
VERTICES_IN_FACETS
BOUNDED

Polymake::Core::Scheduler::RuleChain=ARRAY(0x80bd
7f0)

precondition : BOUNDED ( POINTED : )
POINTED :
precondition : POINTED ( LINEALITY_DIM, LINEALITY
_SPACE : CONE_AMBIENT_DIM )
LINEALITY_DIM, LINEALITY_SPACE : CONE_AMBIENT_DIM
COMBINATORIAL_DIM : CONE_DIM, LINEALITY_DIM
HASSE_DIAGRAM.ADJACENCY, HASSE_DIAGRAM.DECORATION
, HASSE_DIAGRAM.INVERSE_RANK_MAP, HASSE_DIAGRAM.T
OP_NODE, HASSE_DIAGRAM.BOTTOM_NODE : RAYS_IN_FACE
TS, COMBINATORIAL_DIM
```

Apply the schedule using `->apply($)` .

In [21]:
```
$s->apply($c);
print join("\n", $c->list_properties()), "\n";
```

Out[21]:
```
CONE_AMBIENT_DIM
CONE_DIM
FACETS
AFFINE_HULL
VERTICES_IN_FACETS
BOUNDED
FEASIBLE
POINTED
N_FACETS
N_VERTICES
FULL_DIM
LINEALITY_DIM
LINEALITY_SPACE
COMBINATORIAL_DIM
HASSE_DIAGRAM
```

Toric geometry

Toric geometry is handled by polymakes application "fulton" .

In [22]:
```
application "fulton";
```

In [23]:
```
$c = new Cone(INPUT_RAYS=>[[-1,1],[1,1]]);
print $c->HILBERT_BASIS;
```

Out[23]:
```
-1 1
0 1
1 1
```

In [24]:
```
$toric = $c->TORIC_IDEAL;
print $toric->BINOMIAL_GENERATORS;
print "\n";
print join("\n", @{$toric->GENERATORS});
```

In [25]:
```
$tv = new NormalToricVariety($c);
print "Is smooth? ", $tv->SMOOTH, "\n";
print "Is affine? ", $tv->AFFINE, "\n";
print "Is orbifold? ", $tv->ORBIFOLD, "\n";
print "Is complete? ", $tv->COMPLETE, "\n";
$tv->properties();
```

Out[25]:
```
polymake: used package cdd
  cddlib
  Implementation of the double description method
of Motzkin et al.
  Copyright by Komei Fukuda.
  http://www-oldurls.inf.ethz.ch/personal/fukudak
/cdd_home/ (http://www-oldurls.inf.ethz.ch/person
al/fukudak/cdd_home/)

Is smooth?
Is affine? 1
Is orbifold? 1
Is complete?
type: NormalToricVariety

LINEALITY_SPACE


RAYS
-1 1
1 1


MAXIMAL_CONES
{0 1}


INTERSECTION_COMPLEX
type: SimplicialComplex

LINEALITY_DIM
0

N_MAXIMAL_CONES
1

MAXIMAL_CONES_COMBINATORIAL_DIMS
1
```

Affine toric varieties

Affine toric varieties arise from cones. To dualize a (pointed) cone just construct a new cone switching facets and rays.

In [26]:
```
$sigma = new Cone(INPUT_RAYS=>[[1,0,0],[0,1,0],[1,0
$sigmad = new Cone(INPUT_RAYS=>$sigma->FACETS);
print $sigmad->HILBERT_BASIS, "\n";
print $sigmad->TORIC_IDEAL->BINOMIAL_GENERATORS, "\
print $sigmad->TORIC_IDEAL->GENERATORS, "\n";
```

The exponent vectors of the binomials generate the lattice of relations of the Hilbert basis of the dual cone. In particular we have

In [27]:
```
$hb = $sigmad->HILBERT_BASIS;
$binomials = $sigmad->TORIC_IDEAL->BINOMIAL_GENERAT
print $binomials * $hb;
```

Cyclic quotient singularities

Two-dimensional pointed cones in two-dimensional space can always be generated by two rays in the standard form `[[1,0],[-q,n]]` with `0<q<n` and `q` and `n` coprime.

In [28]:
```
$N7Q3 = new CyclicQuotient(N=>7, Q=>3);
print primitive($N7Q3->RAYS);
```

Out[28]:
```
1 0
-3 7
```

CQS are deeply connected with the associated continued fractions.

In [29]:
```
$cf = new Vector<Rational>($N7Q3->CONTINUED_FRACTIO
print $cf, "\n";
print $cf->[0]-1/($cf->[1]-1/$cf->[2]), "\n";
```

Out[29]:
```
3 2 2
7/3
```

In [30]:
```
$dcf = new Vector<Rational>($N7Q3->DUAL_CONTINUED_F
print $dcf, "\n";
print $dcf->[0] -1/ $dcf->[1] , "\n";
```

Out[30]:
```
2 4
7/4
```

If we start with the Hilbert basis of the dual cone, sorted by first coordinate:

In [31]:
```
print $N7Q3->WEIGHT_CONE->HILBERT_BASIS, "\n";
$sorted = new Matrix(sort(@{$N7Q3->WEIGHT_CONE->HIL
print $sorted;
```

Out[31]:
```
0 1
1 1
2 1
7 3

0 1
1 1
2 1
7 3
```

Then the dual continued fraction expansion of `n/(n-q)` gives us relations among these elements:

In [32]:
```
print $sorted->[0] + $sorted->[2] == $dcf->[0] * $s
print $sorted->[1] + $sorted->[3] == $dcf->[1] * $s
```

Out[32]:
```
1
1
```

Q: Derive and prove the general relation formula.

Projective varieties

Projective toric varieties arise from polytopes. Normal projective toric varieties arise from fans. To construct a projective toric variety, simply take the normal fan of a polytope and give it to the `NormalToricVariety` constructor.

In [33]:
```
$PP2 = new NormalToricVariety(normal_fan(simplex(2)
print $PP2->RAYS;
print $PP2->MAXIMAL_CONES;
```

Out[33]:
```
0 1
1 0
-1 -1
{0 1}
{0 2}
{1 2}
```

In [34]:
```
$PP2->VISUAL;
```

The Hasse diagram looks as follows:

In [35]:
```
$PP2->HASSE_DIAGRAM->VISUAL;
```

Now consider the following two cones:

In [36]:
```
$c1 = new Cone(simplex(2));
print $c1->HILBERT_BASIS, "\n";
$c2 = new Cone(simplex(2,2));
print $c2->HILBERT_BASIS, "\n";
```

Out[36]:
```
1 0 0
1 0 1
1 1 0

1 0 0
1 0 1
1 0 2
1 1 0
1 1 1
1 2 0
```

Lets look at the associated toric ideals:

In [37]:
```
print "C1: ", join("\n", @{$c1->TORIC_IDEAL->GENERA
print "C2: ", join("\n", @{$c2->TORIC_IDEAL->GENERA
```

The first ideal is actually `0`. Both ideals are homogeneous and hence, they define projective varieties.

Q: What are these projective varieties?

A non-projective toric variety

Not every fan is the normal fan of a polytope. Here we give an example.

In [38]:
```
$f = new PolyhedralFan(INPUT_RAYS=>
[[1,0,0],[0,1,0],[-1,-1,-1],
[0,0,1],[2,1,1],[1,2,1],[1,1,2]],
INPUT_CONES=>[[0,1,2],[0,2,3],
[1,2,3],[4,5,6],[0,1,4],[1,3,5],
[0,3,6],[1,4,5],[3,5,6],[0,4,6]]
);
$tv = new NormalToricVariety($f);
print "Projective? ", $tv->PROJECTIVE, "\n";
print "Smooth? ", $tv->SMOOTH, "\n";
print "Complete? ", $tv->COMPLETE, "\n";
```

Out[38]:
```
Projective?
Smooth?
Complete? 1
```

In [39]:
```
$tv->VISUAL;
```

Hirzebruch surfaces

Hirzebruch surfaces come from two-dimensional complete fans with exactly four rays. Smoothness makes it possible to bring these fans into a standard form such that we arrive at a one-parameter family.

In [40]:
```
$h1 = hirzebruch_surface(1);
print $h1->RAYS;
```

Out[40]:
```
1 0
0 1
-1 1
0 -1
```

In [41]:
```
$h1->VISUAL;
```

In [42]:
```
$h2 = hirzebruch_surface(2);
print $h2->RAYS;
```

Out[42]:
```
1 0
0 1
-1 2
0 -1
```

In [43]:
```
$h2->VISUAL;
```

Q: Construct the/an associated polytope.

Q: Prove the standard form for complete smooth fans in two dimensions with exactly four rays.

polymake has a method to reconstruct a polytope from a regular fan / projective toric variety.

In [44]:
```
$A = generating_polyhedron_facets($h2);
print $A;
```

Out[44]:
```
2 1 0
1 0 1
1 -1 2
1 0 -1
```

This polytope has the given fan as a normal fan.

In [45]:
```
$P = new Polytope(INEQUALITIES=>$A);
$FF = normal_fan($P);
print $FF->RAYS;
print $FF->MAXIMAL_CONES;
```

Out[45]:
```
1 0
0 1
-1 2
0 -1
{2 3}
{0 3}
{1 2}
{0 1}
```

In [46]:
```
$FF->VISUAL;
```

Simple, not smooth

In [47]:
```
$p = new Polytope(POINTS=>[
[1, 0, 0, 0],
[1, 1, 1, 0],
[1, 1, 0, 1],
[1, 0, 1, 1]]);
print "Simple? ", $p->SIMPLE, "\n";
print "Smooth? ", $p->SMOOTH, "\n";
```

Out[47]:
```
Simple? 1
Smooth?
```

Non-normal toric varieties

polymake only handles normal toric varieties. Nevertheless we can use it to get at the toric ideal of a non-normal toric variety by giving the semigroup generators directly as a mock Hilbert basis. For example for the Neil parabola use:

In [48]:
```
$c = new Cone(HILBERT_BASIS_GENERATORS=>[[[2],[3]],
print $c->TORIC_IDEAL->BINOMIAL_GENERATORS;
print $c->TORIC_IDEAL->GENERATORS;
```

If we build a new cone from this semigroup, we see that it was not saturated. The semigroup generated by 2 and 3 does not come from a cone.

In [49]:
```
$cc = new Cone(INPUT_RAYS=>$c->HILBERT_BASIS_GENERA
print $cc->RAYS, "\n";
print $cc->HILBERT_BASIS, "\n";
```

Out[49]:
```
1

1
```

Q: What are necessary conditions for a semigroup to come from a cone?

Smooth vs normal

It is an open question by Oda whether smoothness and normality of polytopes are equivalent.

In [50]:
```
print $p->VERTICES, "\n";
print "Normal? ", $p->NORMAL, "\n";
print "Smooth? ", $p->SMOOTH, "\n";
```

Out[50]:
```
1 0 0 0
1 1 1 0
1 1 0 1
1 0 1 1

Normal?
Smooth?
```

In [51]:
```
help "NORMAL";
help "SMOOTH";
```

Out[51]:
```
objects/Polytope/properties/Lattice points in con
es/NORMAL:
property NORMAL : Bool
 Only defined for Polytope::Lattice.
 The polytope is normal if the cone spanned by P
x {1} is generated in height 1.

Depends on:
  4ti2 or libnormaliz

There are 2 help topics matching 'SMOOTH':
--------------------
1: objects/NormalToricVariety/properties/Algebrai
c Geometry/SMOOTH:
property SMOOTH : Bool
 A toric variety is smooth if the fan is smooth.
 Alias for property fan::PolyhedralFan::SMOOTH_FA
N.

--------------------
2: objects/Polytope/properties/Lattice points in
cones/SMOOTH:
property SMOOTH : Bool
 Only defined for Polytope::Lattice.
 The polytope is smooth if the associated project
ive variety is smooth; the determinant of the edg
e directions is +/-1 at every vertex.
```

In [52]:
```
print $p->HILBERT_BASIS;
```

Out[52]:
```
1 0 0 0
1 0 1 1
1 1 0 1
1 1 1 0
2 1 1 1
```

Accessing Singular

In [53]:
```
$c = new Cone(INPUT_RAYS=>[[-1,1],[0,1],[1,1]]);
$tv = new NormalToricVariety($c);
$toric = $c->TORIC_IDEAL;
```

In [54]:
```
singular_eval("listvar();");
```

Out[54]:
```
polymake: used package singular
  Singular is a computer algebra system for polyn
omial computations, with special emphasis on comm
utative and non-commutative algebra, algebraic ge
ometry, and singularity theory. It is free and op
en-source under the GNU General Public Licence.
  http://www.singular.uni-kl.de/ (http://www.sing
ular.uni-kl.de/)
```

In [55]:
```
$radical = $toric->RADICAL;
print join("\n", @{$radical->GENERATORS});
```

In [56]:
```
singular_eval("listvar();");
```

In [57]:
```
$ideal = $tv->WEIGHT_CONE->TORIC_IDEAL;
print $ideal;
```

In [58]:
```
$cmd = "ring r = 0,(x_0,x_1,x_2),dp;";
print $cmd;
```

Out[58]:
```
ring r = 0,(x_0,x_1,x_2),dp;
```

In [59]:
```
singular_eval($cmd);
singular_eval("r;");
```

Out[59]:
```
// coefficients: QQ
// number of vars : 3
//        block   1 : ordering dp
//                  : names    x_0 x_1 x_2
//        block   2 : ordering C
```

In [60]:
```
singular_eval("int n = nvars(r);");
$n = singular_get_var("n");
print $n."\n";
```

Out[60]:
```
3
```

In [61]:
```
singular_eval("poly p = x_2^2-x_0*x_1");
$p = singular_get_var("p");
print $p."\n";
```

Out[61]:
```
- x_0*x_1 + x_2^2
```

In [62]:
```
load_singular_library("deform.lib");
singular_eval("ideal i = x_0*x_1, x_2;");
singular_eval("def L = versal(i);");
singular_eval("L;");
```

Out[62]:
```
[1]:
   // coefficients: QQ
// number of vars : 4
//        block   1 : ordering ds
//                  : names    A
//        block   2 : ordering dp
//                  : names    x_0 x_1 x_2
//        block   3 : ordering C
[2]:
   // coefficients: QQ
// number of vars : 4
//        block   1 : ordering ds
//                  : names    A
//        block   2 : ordering dp
//                  : names    x_0 x_1 x_2
//        block   3 : ordering C
// quotient ring from ideal ...
[3]:
   // coefficients: QQ
// number of vars : 1
//        block   1 : ordering ds
//                  : names    A
//        block   2 : ordering C
[4]:
   // coefficients: QQ
// number of vars : 4
//        block   1 : ordering ds
//                  : names    A
//        block   2 : ordering dp
//                  : names    x_0 x_1 x_2
//        block   3 : ordering C
```

In [ ]: