Freie Universität Berlin
II. Mathematisches Institut

ConvexPolyhedron - A Macaulay 2 package

René Birkner
Rheinstr. 35
12161 Berlin
`rbirkner@math.fu-berlin.de`
`page.mi.fu-berlin.de/rbirkner/`

Contents

I Types

In this section we state the new types declared in ConvexPolyhedron and give a description.

I.1 Polyhedron

A Polyhedron represents a rational polyhedron. It can be bounded or bounded, need not be full dimensional or may contain a proper affine subspace. It can be empty or zero dimensional. It is saved as a mutable hashtable which contains the vertices, generating rays, and the basis of the lineality space of the Polyhedron as well as the defining affine halfspaces and hyperplanes. The output of a Polyhedron looks like this:

```
o1 = HashTable{ambient dimension => 3           }
               dimension of lineality space => 0
               dimension of polyhedron => 2
               number of facets => 5
               number of rays => 1
               number of vertices => 4

o1 : Polyhedron
```

This table displays a short summary of the properties of the Polyhedron. Note that the number rays and vertices are modulo the lineality space. So for example a line in $\mathbb{Q}^2$ has one vertex and no rays. However, one can not access the above information directly, because this is just a *virtual* hashtable generated for the output. The informations of a Polyhedron are extracted by the functions included in this package:

- `ambdim`
- `halfspaces`
- `hyperplanes`
- `linspace`
- `polydim`
- `rays`
- `vertices`

A Polyhedron can be constructed as the convex hull of a set of points and a set of rays or as the intersection of a set of affine halfspaces and affine hyperplanes. The functions for this in ConvexPolyhedron are:

- `convexHull`
- `intersection`

The following functions and methods for polyhedra are implemented in ConvexPolyhedron:

- `affineImage`
- `bipyramid`
- `contains`
- `crosspolytope`
- `cyclicPolytope`
- `directproduct`
- `equals`
- `faces`
- `fVector`
- `hypercube`
- `minkowskisum`
- `minkSummandCone`
- `newtonPolytope`
- `polar`
- `pyramid`
- `smallestFace`
- `stdSimplex`
- `tailCone`

I.2 Cone

A Cone represents a rational convex polyhedral cone. It need not be full dimensional or may contain a proper linear subspace. It can be empty or zero dimensional. It is saved as a mutable hashtable which contains the generating rays and the basis of the lineality space of the cone as well as the defining halfspaces and hyperplanes. The output of a Cone looks like this:

```
o2 = HashTable{ambient dimension => 4           }
               dimension of lineality space => 1
               dimension of the cone => 3
               number of facets => 2
               number of rays => 2

o2 : Cone
```

This table displays a short summary of the properties of the Cone. Again the number of rays is modulo the lineality space, so in the above example the rays are not the generator of the lineality space and its negative. However, one can not access the above information directly, because this is just a *virtual* hashtable generated for the output. The informations of a Cone are extracted by the functions included in this package:

- `ambdim`
- `halfspaces`
- `hyperplanes`
- `linspace`
- `conedim`
- `rays`

A Cone can be constructed as the positive hull of a set of rays or as the intersection of a set of linear halfspaces and linear hyperplanes. The functions for this in ConvexPolyhedron are:

- `posHull`
- `intersection`

The following functions and methods for cones are implemented in Convex-Polyhedron:

- affineImage
- coneToPolyhedron
- contains
- directproduct
- dualCone
- equals
- faces
- fVector
- hilbertBasis
- smallestFace

II Functions

Since every function in ConvexPolyhedron is implemented as a method which can have different inputs and different output types, there are a couple of functions for which there are multiple synopsis expressions.

II.1 affineImage

Synopsis

- Usage: `affineImage(A,v,P)` or `affineImage(A,P)`
- Inputs:
  - `P`, a Polyhedron
  - `A`, a matrix over ZZ or QQ with the ambient space of `P` as source space
  - `v`, a one column matrix giving a vector in the target space of `A`
- Outputs:
  - The Polyhedron which is the image of `P` under `A` translated by `v`.

Description `affineImage` computes the affine image $\{(A \cdot p) + v \,|\, p \in P\}$ where `v` is set to 0 if omitted.

II.2 ambdim

Synopsis

- Usage: `ambdim P` or `ambdim C`
- Inputs:
  - `P`, a Polyhedron
  - `C`, a Cone
- Outputs:
  - The ambient dimension of type ZZ.

Description `ambdim` returns the dimension of the ambient space either of the Polyhedron `P` or the cone `C`.

II.3 bipyramid

Synopsis

- Usage: `bipyramid P`
- Inputs:
  - `P`, a non-empty Polyhedron
- Outputs:
  - A Polyhedron which is the bipyramid over `P`.

Description The `bipyramid` over a Polyhedron is constructed, by embedding the Polyhedron into $n+1$ space, computing the barycentre of the vertices, which is a point in the relative interior, and taking the convex hull of the embedded Polyhedron and the barycentre $\times\{\pm 1\}$.

II.4 commonFace

Synopsis

- Usage: `commonFace(P,Q)` or `commonFace(C1,C2)`
- Inputs:
  - `P,Q`, two Polyhedra
  - `C1,C2`, two Cones
- Outputs:
  - a boolean

Description `commonFace` tests whether the intersection of both arguments is a face of each argument.

II.5 conedim

Synopsis

- Usage: `conedim C`
- Inputs:

  - `C`, a Cone

- Outputs:
  - The cone dimension of type ZZ.

Description `conedim` returns the dimension of the Cone `C`.

II.6 coneToPolyhedron

Synopsis

- Usage: `coneToPolyhedron C`
- Inputs:
  - `C`, a Cone
- Outputs:
  - The Cone transformed into type Polyhedron.

Description `coneToPolyhedron` returns exactly the same Cone but of type Polyhedron.

II.7 contains

Synopsis

- Usage: `contains(P,Q)` or `contains(P,p)`
- Inputs:
  - `P`, a Polyhedron or a Cone
  - `Q`, a Polyhedron or a Cone
  - `p`, a one column matrix giving a point
- Outputs:
  - boolean

Description `contains` determines if the first argument contains the second argument. Both arguments have to lie in the same ambient space. It tests if the equations of the first argument are satisfied by the generating points/rays of the second argument.

II.8 convexHull

Synopsis

- Usage: `convexHull M` or `convexHull(M,N)` or `convexHull(P,Q)`
- Inputs:
  - `M`, a matrix where the columns are considered as points
  - `N`, a matrix where the columns are considered as rays
  - `P,Q`, two Polyhedra
- Outputs:
  - A Polyhedron.

Description `convexHull` computes the convex hull of the input. In the first two cases it considers the columns of the first matrix as a set of points and the columns of the second (if given) as a set of rays and computes the polyhedron which is the convex hull of the points plus the rays. The two matrices must have the same number rows. If `N` is not given or equal to 0 then the resulting polyhedron is compact and hence a polytope. The points need not be a the vertices of the polyhedron. In the last case it computes the convex hull of the two polyhedra if they lie in the same ambient space.

II.9 crosspolytope

Synopsis

- Usage: `crosspolytope(d,s)` or `crosspolytope(d)`
- Inputs:
  - `d`, a strictly positive integer
  - `s`, a positive integer or a rational number
- Outputs:
  - A Polyhedron, the `d` dimensional crosspolytope with diameter $2 \cdot \mathtt{s}$.

Description The `d` dimensional `crosspolytope` with diameter `s` is the convex hull of $\pm$ `s` the standard basis in $\mathbb{Q}^{\mathtt{d}}$.

II.10 cyclicPolytope

Synopsis

- Usage: `crosspolytope(d,n)`
- Inputs:
  - `d`, a strictly positive integer
  - `n`, a strictly positive integer
- Outputs:
  - A Polyhedron, the `d` dimensional cyclic polytope with `n` vertices.

Description The `d` dimensional `cyclicpolytope` with `n` vertices is the convex hull of `n` points on the moment curve in $\mathbb{Q}^d$. The moment curve is defined by $t \rightarrow (t, t^2, ..., t^d)$ and the function takes the points $\{0, ..., n-1\}$.

II.11 directproduct

Synopsis

- Usage: `directproduct(P,Q)`
- Inputs:
  - `P`, a Polyhedron or a Cone
  - `Q`, a Polyhedron or a Cone
- Outputs:
  - A Polyhedron, which is the direct product of both, but if `P` and `Q` are cones then the direct product is returned as a Cone.

Description The `directproduct` of `P` and `Q` is the polyhedron $\{(p,q) \,|\, p \in P, q \in Q\}$ in the direct product of the ambient spaces of `P` and `Q`.

II.12 dualCone

Synopsis

- Usage: `dualCone C`
- Inputs:
  - `C`, a Cone
- Outputs:
  - A Cone, which is the dual cone of the input cone Cone.

Description The dual Cone of `C`$\subset \mathbb{Q}^n$ is the cone in the dual ambient space $(\mathbb{Q}^n)^*$ given by $\{p \in (\mathbb{Q}^n)^* \,|\, p \cdot c \geq 0 \forall c \in C\}$.

II.13 equals

Synopsis

- Usage: `equals(P,Q)` or `equals(C1,C2)`
- Inputs:
  - `P,Q`, two Polyhedra
  - `C1,C2`, two Cones
- Outputs:
  - a boolean, indicating whether both objects inserted are equal or not.

Description The function `equals` is necessary, because the hashTables by which the two objects are given cannot be compared and also the order of the columns for example in the vertices matrix is not unique. It uses the function `contains` in both directions. Both objects must be contained in the same ambient space, so for example the positive orthant in $\mathbb{Q}^2$ and the cone in $\mathbb{Q}^3$ spanned by e_1 and e_2 are not considered to be equal in ConvexPolyhedron.

II.14 faces

Synopsis

- Usage: `faces(k,P)` or `faces(k,C)`
- Inputs:
  - `k`, an integer
  - `P`, a Polyhedron
  - `C`, a Cone
- Outputs:
  - a list, containing the codim. `k` faces of `P` or `C`

Description The function `faces` computes the faces of codimension `k` if the second argument is a Polyhedron then they are saved as Polyhedra and if the second argument is a Cone then the faces are again cones and thus saved as cones. The function does not consider the empty face of the input as a face so that `k` must be between 0 and the dimension of the Polyhedron/Cone.

II.15 fVector

Synopsis

- Usage: `fVector(P)` or `fVector(C)`
- Inputs:
  - `P`, a Polyhedron
  - `C`, a Cone
- Outputs:
  - a list, containing the number of faces for each dimension

Description The i-th entry of the `fVector` of `P` is the number of codimension $i-1$ faces of `P`, so it starts with a 1 for `P` itself, has dimension(`P`)+1 entries, and the last one is the number of vertices. It is the same for a Cone `C`.

II.16 halfspaces

Synopsis

- Usage: `halfspaces(P)` or `halfspaces(C)`
- Inputs:
  - `P`, a Polyhedron
  - `C`, a Cone
- Outputs:
  - two matrices over $\mathbb{Q}$if the input is a Polyhedron and one matrix over $\mathbb{Q}$ if the input is a Cone

Description This function returns the defining halfspaces. For a polyhedron P the output is (M,v) where the source of M has the dimension of the ambient space of P and v is a one column matrix in the target space of M such that $P = \{p \in H \,|\, M \cdot p \leq v\}$ where H is the intersection of the defining affine hyperplanes. For a cone C the output is M which is the same matrix as before but v is ommited since it is 0, so $C = \{c \in H \,|\, M \cdot c \geq 0\}$ and H is the intersection of the defining hyperplanes.

II.17 hilbertBasis

Synopsis

- Usage: `hilbertBasis(C)`
- Inputs:
  - `C`, a Cone
- Outputs:
  - a list, containing the elements of the Hilbert basis of `C`

Description The Hilbert basis of the cone C is computed by the Project-and-Lift-algorithm by Raymond Hemmecke [Hem02]. It computes a Hilbert basis of the cone modulo the lineality space, so it returns a list of one column matrices which give the Hilbert basis of the Cone if one adds the basis of the lineality space and its negative.

II.18 hypercube

Synopsis

- Usage: `hypercube(d,s)` or `hypercube d`
- Inputs:
  - `d`, a strictly positive integer
  - `s`, a positive integer or rational number
- Outputs:
  - a Polyhedron, the `d` dimensional hypercube with edge length 2·`s`

Description The d dimensional hypercube with edge length $2 \cdot s$ is the convex hull of all points in $\{\pm s\}^d$ in $\mathbb{Q}^d$.

II.19 hyperplanes

Synopsis

- Usage: `hyperplanes(P)` or `hyperplanes(C)`
- Inputs:
  - `P`, a Polyhedron
  - `C`, a Cone
- Outputs:
  - two matrices over $\mathbb{Q}$if the input is a Polyhedron and one matrix over $\mathbb{Q}$ if the input is a Cone

Description This function returns the defining hyperplanes. For a polyhedron P the output is (M,v) where the source of M has the dimension of the ambient space of P and v is a one column matrix in the target space of M such that $P = \{p \in H \,|\, M \cdot p = v\}$ where H is the intersection of the defining affine halfspaces. For a cone C the output is M, which is the same matrix as before, but v is ommited since it is 0, so $C = \{c \in H \,|\, M \cdot c = 0\}$ and H is the intersection of the defining halfspaces.

II.20 intersection

Synopsis

- Usage: `intersection(M,v,N,w)` or `intersection(M,v)` or `intersection(M,N)` or `intersection M` or `intersection(P,Q)`
- Inputs:
  - `M,N`, two matrices over $\mathbb{Z}$ or $\mathbb{Q}$ where the rows are considered as linear equations
  - `v,w`, two one column matrices over $\mathbb{Z}$ or $\mathbb{Q}$
  - `P,Q`, each a Polyhedron or a Cone
- Outputs:
  - A Polyhedron or a Cone.

Description In the first four cases `intersection` considers the the given matrices as defining inequalities and equalities. Thus it computes the polyhedron $P = \{p \,|\, M \cdot p \leq v \text{ and } N \cdot p = w\}$. Therefore M and N must have the same number of columns, which will be the dimension of the ambient space, and M and v as well as N and w must have the same number of rows respectively. If N and w are omitted then the polyhedron is just given by the inequality. If v and w are omitted then they are considered to be 0 so that the `intersection` is a cone and thus the output is also a cone.

In the last case the intersection of both arguments is computed if both arguments lie in the same ambient space. If both arguments are cones then the output is again a cone. Otherwise `intersection` returns a polyhedron.

II.21 isCompact

Synopsis

- Usage: `isCompact P`
- Inputs:
  - `P`, a Polyhedron
- Outputs:
  - a boolean

Description Tests whether `P` is compact, i.e. a polytope.

II.22 isFace

Synopsis

- Usage: `isFace(P,Q)` or `isFace(C1,C2)`
- Inputs:
  - `P,Q`, two Polyhedra
  - `C1,C2`, two Cones
- Outputs:
  - a boolean

Description `isFace` tests whether the first argument is a face of the second argument.

II.23 linspace

Synopsis

- Usage: `linspace P` or `linspace C`
- Inputs:
  - `P`, a Polyhedron
  - `C`, a Cone
- Outputs:
  - a matrix over $\mathbb{Q}$

Description `linspace` returns a basis of the lineality space of the input as the columns of a matrix.

II.24 minkowskisum

Synopsis

- Usage: `minkowskisum(P,Q)`
- Inputs:
  - `P`,`Q`, each a Polyhedron or a Cone
- Outputs:
  - a Polyhedron or a Cone

Description The `minkowskisum` of P and Q is the polyhdedron $P+Q = \{p+q \,|\, p \in P, q \in Q\}$. If P and Q are both cones then their Minkowskisum is their positive hull which is a cone, so the output is a cone. Otherwise the output is a polyhedron. P and Q have to lie in the same ambient space.

II.25 minkSummandCone

Synopsis

- Usage: `minkSummandCone P`
- Inputs:
  - `P`, a Polyhedron
- Outputs:
  - `(C,L,M)`, the first entry `C` is a Cone, the second entry `L` is a list of Polyhedra and the third entry `M` is a matrix

Description For the Minkowski summand cone one takes $\mathbb{Q}^d$ where d is the number of edges of the input polyhedron P. Every Minkowski summand of P has only edges that are edges of P, so it can be constructed by rescaling every edge of P, i.e. is represented by a point in $\mathbb{Q}^d$. But not every point in $\mathbb{Q}^d$ gives a polyhedron via this method. This is the case if on the one hand the point lies in the positive orthant and on the other hand for every compact two dimensional face of P the rescaled edges of this faces give a two dimensional polytope, i.e. the sum of the ordered rescaled edge directions is zero. Therefore every compact two dimensional face of P gives a set of linear equalities on a part of the variables in $\mathbb{Q}^d$. The MinkowskiSummandCone C

is the intersection of the positive orthant with these equations. The corresponding polyhedron to every minimal generator of C is saved in the list L. Finally all possible minimal decompositions of P are saved as columns in the matrix M.

II.26 newtonPolytope

Synopsis

- Usage: `newtonPolytope r`
- Inputs:
  - `r`, a RingElement
- Outputs:
  - a Polyhedron

Description The `newtonPolytope` of r is the covex hull of its exponent vectors.

II.27 polar

Synopsis

- Usage: `polar P`
- Inputs:
  - `P`, a Polyhedron
- Outputs:
  - a Polyhedron, the polar of `P`

Description The polar polyhedron of P is the polyhedron in the dual space given by $\{v \in (\mathbb{Q}^d)^* \,|\, v \cdot p \leq 1 \forall p \in P\}$.

II.28 polydim

Synopsis

- Usage: `polydim P`
- Inputs:
  - `P`, a Polyhedron
- Outputs:
  - an integer, the dimension of the Polyhedron

Description `polydim` returns the dimension of the Polyhedron `P`.

II.29 posHull

Synopsis

- Usage: `posHull(M,N)` or `posHull M` or `posHull(C1,C2)` or `posHull P`
- Inputs:
  - `M`, a matrix where the columns are considered as rays
  - `N`, a matrix where the columns are considered as generators of the lineality space
  - `C1,C2`, two Cones
  - `P`, a Polyhedron
- Outputs:
  - A Cone

Description `posHull` computes the positive hull of the input. In the first two cases it considers the columns of the first matrix as a set of rays and the columns of the second (if given) as generators of the lineality space and computes the cone which is the positive hull of the rays plus the lineality space. The two matrices must have the same number rows. If `N` is not given or equal to 0 then the resulting cone is pointed. The rays need not be a minimal generating set of the cone. If two cones are inserted it computes their positive hull if they lie in the same ambient space. In the last case it computes the cone given by all positive multiples of points of the polyhedron.

II.30 pyramid

Synopsis

- Usage: `pyramid P`
- Inputs:
  - `P`, a Polyhedron
- Outputs:
  - A Polyhedron

Description The function takes the polyhedron P with ambient dimension n and embedds it into $\mathbb{Q}^{n+1}$ on height 0 with respect to the new last variable. The it computes the convex hull of the embedded P and the point $(0, ..., 0, 1)$.

II.31 rays

Synopsis

- Usage: `rays P` or `rays C`
- Inputs:
  - `P`, a Polyhedron
  - `C`, a Cone
- Outputs:
  - a matrix over $\mathbb{Q}$

Description Returns the rays of the input as the columns of a matrix.

II.32 smallestFace

Synopsis

- Usage: `smallestFace(p,P)` or `smallestFace(p,C)`
- Inputs:
  - `p`, a one column matrix over $\mathbb{Z}$ or $\mathbb{Q}$
  - `P`, a Polyhedron

    - `C`, a Cone

- Outputs:
    - a Polyhedron or a Cone

Description p is considered to be point in the ambient space of the second argument, so the number of rows of p must equal the dimension of the ambient space of the second argument. The function computes the smallest face of the second argument which contains p. If the second argument is a polyhedron the output is a polyhedron and if it is a cone the output is a cone.

II.33 stdSimplex

Synopsis

- Usage: `stdSimplex d`
- Inputs:
    - `d`, a positive integer
- Outputs:
    - a Polyhedron, the standard `d` simplex

Description The d dimesional standard simplex is the convex hull of standard basis in $\mathbb{Q}^{d+1}$.

II.34 tailCone

Synopsis

- Usage: `tailCone P`
- Inputs:
    - `P`, a Polyhedron
- Outputs:
    - a Cone, the tail cone of `P`

Description The tail cone of P is the cone generated by the non-compact part, i.e. the rays and the lineality space of P. If P is a polytope then the tail cone is the origin in the ambient space of P.

II.35 vertices

Synopsis

- Usage: `vertices P`
- Inputs:
  - `P`, a Polyhedron
- Outputs:
  - a Matrix

Description Returns the vertices of the polyhedron as the columns of a matrix.

References

[Hem02] Raymond Hemmecke. On the computation of Hilbert bases of cones. In A. M. Cohen, X.-S. Gao, and N. Takayama, editors, *Mathematical Software, ICMS 2002*, pages 307–317. World Scientific, 2002.