MANUAL OF THE GAP PACKAGE POLYHEDRAL

MATHIEU DUTOUR SIKIRIĆ

Contents

1. Installation

A priori the system works only on unix/linux systems. You need to follow the following steps:

(1) The archive **polyhedral.tar.gz** can be downloaded from http://drobilica.irb.hr/~mathieu/Polyhedral/index.html

(2) Previous to using polyhedral you need to install the GAP computer package (from http://www.gap-system.org/).
(3) The archive **polyhedral.tar.gz** should be untarred in the **pkg** directory of GAP.
(4) Your File **.gap/gap.ini** must contain the following line:
```
SetUserPreference( "PackagesToLoad", [ "polyhedral"]);
```
if you have no other needed packages. If the file is not existent then you need to create it.
(5) Then one needs to run the **configure** perl script in the **polyhedral** directory in order to compile the external programs.

2. Introduction

The package `polyhedral` is designed to be used for doing all kinds of computations related to polytopes and use their symmetry groups in the course of the computation. There are very many different functions but I will try here to explain them as good as I can.

In order to install it, you should

(1) Install `GAP`.
(2) Download `polyhedral.tar.gz` from http://www.liga.ens.fr/~dutour/Polyhedral
(3) Unpack it in the directory `pkg/` of the gap distribution.
(4) Do `./configure` in order to compile the external programs.

3. Combinatorial functions

3.1. **Graph functions.** The package contains a reimplementation of the `nauty` [11] functionalities into GAP. The preceding GRAPE has the defect of being rather slow due to its additional features and does not contained the feature of vertex colored graph. Thus a graph is either described as

- A graph in the GRAPE format
- A graph in the ListAdj Format, so that `ListAdj[iVert]` is the list of vertices adjacent to `iVert`.

(1) The functions
```
CanonicalRepresentativeVertexColoredGraphAdjList:=function(ListAdjacency,
CanonicalRepresentativeVertexColoredGraph:=function(TheGraph, ThePartitio
```
returns the canonical representative of a vertex colored graph by using the nauty program.
(2) The functions
```
EquivalenceVertexColoredGraphAdjList:=function(ListAdjacency1, ListAdjace
EquivalenceVertexColoredGraph:=function(TheGraph1, TheGraph2, ThePartitio
```

tests the equivalence of vertex colored graphs. ThePartition is for example

```
ThePartition:=[[1,2,3,4], [5,6]];
```

and has to be common for both graphs.

(3) The function

```
AutomorphismGroupEdgeColoredGraph:=function(DistMat)
```

returns the automorphism group of a graph with edge weigths. DistMat is the matrix of weight (diagonal weights are not used).

(4) The function

```
CanonicalStringEdgeColoredGraph:=function(DistMat)
```

returns the canonical string of an edge colored graph.

(5) The function

```
IsIsomorphicEdgeColoredGraph:=function(DistMat1, DistMat2)
```

tests if two edge colored graphs are equivalent.

(6) The functions

```
AutomorphismGroupColoredGraph:=function(ScalarMat)
IsIsomorphicColoredGraph:=function(ScalarMat1, ScalarMat2)
```

are the equivalent of the above for vertex colored and edge weighted graphs.

For a graph the returned group is a group expressed as a permutation group on the vertices. For equivalence tests, the program return `false` if the two objects are not isomorphic and a list expressing the isomorphism otherwise.

3.2. **Invariant functions.**

(1) The function

```
__GetMD5sum:=function(FileName)
```

returns the `md5sum` of a file. This is especially useful when one has complex invariants of a mathematical object and wants a smaller one to keep in memory.

(2) The function

```
__GetGraph6Expression:=function(ListAdj)
```

returns the `nauty` Graph6 expression of a graph. It is a compact form useful for memory expression.

(3) The functions

```
SymmetryGroupVertexColoredGraphAdjList:=function(ListAdjacency, ThePartit
SymmetryGroupVertexColoredGraph:=function(TheGraph, ThePartition)
```

return the symmetry group of a vertex colored graph.

3.3. **Combinatorial functions.** Sometimes, we need to enumerate objects and special graph

(1) The function

`GetBipartition:=function(GR)`

returns the bipartition of a graph `GR` if it is bipartite and

(2) The function

`GRAPH_EnumerateCycles:=function(TheGRA, GRP, TheLen)`

enumerates the orbits for the group `GRP` of cycles of `TheGRA` of length `TheLen`.

3.4. **Group functions.** A key point of all the work being done there is to compute stabilizer and equivalence for some various action. We have written some algorithm that realize this for some specific actions.

The functions

```
PermutedStabilizer:=function(TheGRP, eVect)
PermutedRepresentativeAction:=function(TheGRP, eVect1, eVect2)
```

are supposed to behave like

```
Stabilizer:=function(TheGRP, eVect, Permuted)
RepresentativeAction:=function(TheGRP, eVect1, eVect2, Permuted)
```

and hopefully be faster.

Similarly we have for the action `OnSetsSets`

```
OnSetsSetsStabilizer:=function(GRP, eSetSet)
OnSetsSetsRepresentativeAction:=function(GRP, eSetSet1, eSetSet2)
```

for the action `OnTuples`:

```
OnTuplesStabilizer:=function(GRP, eTuple)
OnTuplesRepresentativeAction:=function(SymGrp, Tuple1, Tuple2)
OnTuplesCanonicalization:=function(GroupEXT, ListPts)
```

for the action `OnTuplesSets`:

```
OnTuplesSetsStabilizer:=function(GRP, eTuple)
OnTuplesSetsRepresentativeAction:=function(GroupEXT, FlagEXT1, FlagEXT2)
OnTuplesSetsCanonicalization:=function(GroupEXT, ListSet)
```

4. Linear algebra functionalities

4.1. **Basic functions.**

(1) For a $n \times m$ matrix of rank p we need to remove some rows or columns so that the resulting matrix has rank p. The functions are

`ColumnReduction:=function(EXT)`
`RowReduction:=function(EXT)`

They return a record

`rec(EXT:=EXTred, Select:=eSet)`

with `EXTred` the reduced matrix and `eSet` the set of rows/columns that has to be chosen.

(2) For an integral vector `TheVector` with

`TheVector:=[x1, ..., xN]`

the function

`GcdVector:=function(TheVector)`

returns a record `rec(TheGcd:=TheGcd , ListCoef:=[a1, ..., aN])` such that `TheGcd` is the greatest common divisor and

$$TheGcd = \sum_{i=1}^{N} a_i x_i$$

(3) The functions

`RemoveFraction:=function(TheList)`
`RemoveFractionMatrix:=function(OneMat)`

multiply a vector or a matrix by the smallest integer such that they are integral. If one needs further the coefficient, then use

`RemoveFractionPlusCoef:=function(TheList)`
`RemoveFractionMatrixPlusCoef:=function(OneMat)`

(4) Given an matrix `eBasis` in a space of dimension n, the function

`SubspaceCompletion:=function(eBasis, n)`

returns a completion basis `B` such that `Concatenation(eBasis, B)` is an integral $\mathbb{Z}$-basis of $\mathbb{Z}^n$.

4.2. **Smith Normal Form.** The Smith normal form is the right tool for computing homology groups. What is needed is to compute the smith normal form of the differential. Since those differentials are in most cases sparse matrix and that there exist specific algorithms using sparse matrices, we find it useful to have sparse matrix functionalities.

The format used for sparse matrix is `rec(nbLine:=...., nbCol:=...., ListEntries:=[....])`. The list `ListEntries` is of length `nbLine` and each entry is of the form `rec(ListCol:=....., ListVal:=....)`.

We use the LINBOX library for those computations, which is one of the most advanced in the world.

The following functionalities are provided via LINBOX:

(1) The function

`GetRankLinboxSparse:=function(RecSparseMat, pVal)`

returns the rank of the sparse matrix in input. if `pVal` is 0 then this is the $\mathbb{Z}$ rank and if it is non-zero, then it is the rank over $\mathbb{Z}/p\mathbb{Z}$.

(2) The function

`GetFactorLinboxSparse:=function(RecSparseMat)`

returns the Smith normal form of the sparse matrix in input.

5. Polytopes

5.1. **Description of polytopes and polyhedral cones.** A polyhedral cone spanned by $(v^1, \ldots, v^m)$ in $\mathbb{R}^n$ can be expressed as

$$P = \{x = \lambda_1 v^1 + \cdots + \lambda_m v^m \text{ with } \lambda_i \geq 0\}$$

To any vector $v^i = (x_1, \ldots, x_n)$ we associate the list

`[x_1, x_2, ..., x_n]`

If the polyhedral cone is defined by inequalities of vector w^1, ..., w^p:

$$P = \{x \text{ such that } \langle x, w^i \rangle \geq 0\}$$

with

$$\langle x, y \rangle = x_1 y_1 + \cdots + x_n y_n$$

then to $w^i = (y_1, \ldots, y_n)$ we associate the list

`[y_1, y_2, ..., y_n]`

So a polyhedral cone P can be either described by a list of list generator coordinates or by a list of list of inequalities coordinate. It is important to note nowhere it is said or mentioned that a list of list express a polytope in terms of its generators or in terms of its inequalities. This meaning is something that the user is responsible for.

For polytopes, the situation is slightly more complex. A polytope P can be defined as the convex hull of points $(v^1, \ldots, v^m)$ in $\mathbb{R}^n$, i.e.

$$P = \left\{ \begin{array}{c} x = \lambda_1 v^1 + \cdots + \lambda_m v^m \text{ with } \lambda_i \geq 0 \\ \sum_i \lambda_i = 1 \end{array} \right\}$$

To any convex generator $v^i = (x_1, \ldots, x_n)$ we associate the list

`[1 , x_1 , x_2 , ... , x_n]`

If the polytope P is defined by inequalities of vector w^1, ..., w^p:

$$P = \{x \text{ such that } \langle x, w^i \rangle \geq b_i\}$$

with

$$\langle x, y \rangle = x_1 y_1 + \cdots + x_n y_n$$

then to $w^i = (y_1, \ldots, y_n)$ and b_i we associate the list

`[-b_i , y_1 , y_2 , ... , y_n]`

This is the convention that we recommend and are used by `lrs`, `cdd`, etc. Their use gives consistent result. So, the polyhedral software does not make any distinction between polyhedral cone and polytope, all this is in the eye of the user.

In general, we do not require the polyhedral cone and polytope to be full dimensional, any embedding is ok. But as a consequence, the faces of cone are defined not by defining inequality but by the vertices or facets in which they are contained. Thus, everything about polyhedral cone is expressed in term of sets, and permutation groups and not matrix groups.

A note on the arihmetic. The coordinates of the objects can be integer, rational or belong to $\mathbb{Q}(\sqrt{(}N))$. The reader should read a book on polytope theory (first chapter suffices) in order to get acquainted with the formalism explained here.

5.2. **Linear programming functions.** Linear programming is the problem of mimnizing a linear function over a set P defined by linear inequalities.

The function that realize that is

```
LinearProgramming:=function(InequalitySet, ToBeMinimized)
```

The `cdd` program is directly called. The output of cdd is directly translated and read. For interpretation of the result, we recommend that the reader looks at a book on linear programming. Linear programming is known to be solvable in polynomial time, which explain its attractivity for many algorithms. The program `cdd` that we use uses the simplex algorithm, which is known to be exponential in worst cases. But in practice this is ok for the cases that we consider and the advantage of it is that the solutions that it returns are rational for rational input.

Based on linear programming we can build many useful functions that generally run very fast.

(1) The function

```
PolytopizationGeneralCone:=function(FAC)
```

does the following operation successively:

(a) It does a reduction to a space so that FAC is full dimensional
(b) It applies a linear transformation so that the first coordinate is always > 1.
(c) It multiplies each vector by the inverse of the first coefficient so that the first coefficient is $= 1$ (so the object is now a polytope).

(d) It translate the coordinate so that origin belongs to the interior of the polytope.

(2) The function

`SearchPositiveRelation:=function(ListVect, TheConstraint)`

is for searching positive relations among vector elements. I.e. we have

`ListVect:=[v1, v2, ..., vN]`

and we search for relations

$$\sum_i \alpha_i v_i = 0$$

with the signs of α_i being specified in the following way:

```
TheConstraint:=rec(ListPositive:=[...],
                   ListStrictlyPositive:=[...],
                   ListSetStrictPositive:=[...]);
```

ListPositive is the list of indices of vectors that are ≥ 0, ListStrictlyPositive the list of indices that are > 0, ListSetStrictPositive the list of sets S for which $\sum_{i \in S} \alpha_i > 0$ with S expressed as set of indices.

(3) The function

`SearchPositiveRelationSimple:=function(ListVect)`

is a simpler version of the above where all indices are non-negative and only the sum should be non-negative

(4) The function

`GetViolatedFacet:=function(EXT, eVect)`

takes a polytope `EXT` and a vector `eVect` that is not in the convex hull of `EXT` and returns a facet F of `EXT` that separates `eVect`.

(5) The function

`EliminationByRedundancyEquivariant:=function(EXT, BoundingSet, GRPperm)`

gives the list of indices that corresponds to vertices. That is if `EXT` is the list of generators, `GRPperm` a group of permutation on the generators, `BoundingSet` a list of valid inequalities on `EXT` (usually set to be empty `[]`).

Linear programming is used to detect which generators correspond to vertices and symmetry is used to reduce the number of linear programming tests.

(6) The function

`SkeletonGraph:=function(GroupExt, EXT, BoundingSet)`

takes a list of vertices `EXT`, a permutation group `GroupExt` on `EXT`, a set `BoundingSet` of valid inequalities (usually set to be empty `[]`). The result is the skeleton of the polytope.

(7) The function

`GetInitialRays_LinProg:=function(EXT, nb)`

takes a polytope or polyhedral cone `EXT` and returns nb subsets of `EXT` corresponding to facets of `EXT`

(8) The function

`LinearDeterminedByInequalities:=function(FAC)`

returns a basis of the space spanned by the vectors realizing the inequalities. For example

```
gap> LinearDeterminedByInequalities([[1,1,1],[-1,-1,-1],[1,0,0]]);
[ [ -1, 1, 0 ], [ -1, 0, 1 ] ]
```

It is useful in dealing with polytopes that are not full dimensional.

(9) The function

`GetContainingPrism:=function(EXT, eVect)`

returns a set S defining a facet F of the polytope P such that `eVect` is inside the prism $\mathrm{conv}(F, iso(P))$.

(10) The function

`GetContainingSimplex:=function(EXT, eVect)`

returns a flag $(F_1, \ldots, F_n)$ such that `eVect` is inside the simplex defined by $(Iso(F_1), \ldots, Iso(F_n))$.

5.3. **Dual description functions.** We now consider functionalities related to computing dual description of polyhedral cone, polytopes, etc. That is we have a description by facets or vertices and we want a description by facets or vertices.

Those functions generally return a list of subsets corresponding to orbit representatives of facets. The method generally uses the Recursive Adjacency Decomposition Method for computing. The exact method is fully described in papers and examples. We give mostly here readily usable functions.

(1) The function

`__FindFacetInequality:=function(EXT, ListIncidence)`

for a set of vertices `EXT` and a list of incidence `ListIncidence` returns the inequality corresponding to this set.

(2) The function

`DualDescription:=function(EXT)`

gives the dual description of `EXT` without using any group theoretical features.

(3) The function

`DualDescriptionAdjacencies:=function(EXT)`

gives the list of facets, the ridge graph and the skeleton graph (again without using any group)

(4) The functions

`DualDescriptionLRS:=function(EXT, GroupExt)`
`DualDescriptionCDD:=function(EXT, GroupExt)`
`DualDescriptionPD:=function(EXT, GroupExt)`

computes some orbit representative (with respect to the group `GroupExt`) of the facets of the cone defined by `EXT`. The program used are `lrs`, `cdd` or `pd`. It is difficult to recommend a specific algorithm in general. This is a matter of research.

(5) The function

`DualDescriptionStandard:=function(EXT, PermGRP)`

computes representatives of the orbits of facets of the cone defined by `EXT` with respect to the group `PermGRP`. Some standard heuristics are applied with the `lrs` programs. If the user wants to use different heuristics, programs then he should look at the examples.

(6) The function

`PolytopeVolumeRecursive:=function(EXT, GroupEXT)`

computes the volume of a polytope by using the above recursive adjacency method iteratively.

See [1] for a discussion of the various methods for computing dual descriptions of polytopes using symmetries.

5.4. **Direct applications of dual description computations.** Remind that a linear programming is the problem of minimizing a linear function over a set defined by some linear inequalities. One weakness of the algorithm previous explained is that the point that they give that realize the minimum may not be unique in some cases. This is not a problem in 99% of cases but sometimes, one needs it to find a point that is canonically defined. The solution is to use

`FindGeometricallyUniqueSolutionToLinearProgramming:=function(ListInequalities,`

and it is usually much more expensive that simple linear programming because more calls to linear programming are needed and also dual description.

In many of the papers of [5] it is somehow important to get the representation matrix of the cone, that is the adjacencies between facets

```
RepresentationMatrixAndFacetStandard:=function(EXT, PermGRP)
```

5.5. **Groups and equivalence of polytopes.** For a polyhedral cone P spanned by N vectors $(v_i)_{1\leq i\leq N}$ in $\mathbb{R}^n$ we denote by

(1) *(Linear)* $Lin(P)$ the group of permutations $\sigma \in Sym(N)$ such that there exist $A \in \mathrm{GL}_n(\mathbb{R})$ with $Av_i = v_{\sigma(i)}$.
(2) *(Projective)* $Proj(P)$ the group of permutations $\sigma \in Sym(N)$ such that there exist $A \in \mathrm{GL}_n(\mathbb{R})$ and $\alpha_i > 0$ with $Av_i = \alpha_i v_{\sigma(i)}$.
(3) *(Combinatorial)* $Comb(P)$ the group of permutations $\sigma \in Sym(N)$ such that σ permutes the faces of P (expressed as subsets of $\{1, \ldots, N\}$).

Of course, one can define the corresponding notions of equivalences. See in [6] for more details on those questions.

This gives us the following functions for automorphism groups:

```
LinPolytope_Automorphism:=function(EXT)
ProjPolytope_Automorphism:=function(EXT)
CombPolytope_Automorphism:=function(EXT)
```

and for equivalences

```
LinPolytope_Isomorphism:=function(EXT1, EXT2)
ProjPolytope_Isomorphism:=function(EXT1, EXT2)
CombPolytope_Isomorphism:=function(EXT1, EXT2)
```

In general the method of choice is the linear group and linear equivalence because it is much faster than other methods. If one wants to use projective or combinatorial equivalence then the computational expenses are much higher.

A function related to linear equivalence is

```
LinPolytope_Invariant:=function(EXT)
```

which returns a powerful linear invariant for a given polyhedral cone.

For obtaining integral stabilizers and equivalence, the commands are:

```
LinPolytopeIntegral_Automorphism:=function(EXT)
LinPolytopeIntegral_Isomorphism:=function(EXT1, EXT2)
```

5.6. **Face lattice and flag computation.** Face lattices computations are fundamental to many computations.

For a n-dimensional polytope P, there are two kinds of face lattice computations:

(1) Enumeration of all orbits of k-dimensional faces for $0 \leq k \leq n$ assuming that we know the list of facets of P (This typically applies to polytope of dimension say, 8 at most).

(2) Enumeration of all orbits of k-dimensional faces for $0 \leq k \leq k_0$ with k_0 small without knowing the set of facets and instead using linear programming (typically, $k_0 = 3, 4$ and the polytope is of dimension about 20)

The functions are thus

`IncompleteSkeletonSearch:=function(GroupFac, FAC, BoundingSet, LevSearch)`

`FAC` is the set of facets, `GroupFac` the permutation group acting on the faces, `BoundingSet` a set of extreme rays (usually taken to be `[]`), `LevSearch` the level at which we go in the enumeration.

`CreateK_skeletton:=function(GroupFac, FAC, EXT)`

`FAC` is the set of facets, `GroupFac` the permutation group acting on the faces and `EXT` the set of extreme rays.

Let us now see the related computations. If P is a n-dimensional polyhedral cone with faces of dimension between 1 and n then a flag is a set of faces f_i:

$$f = (f_1, f_2, \ldots, f_n) \text{ with } f_1 \subset f_2 \subset \cdots \subset f_n$$

Such flags are encoded by the set of extreme rays in which they are contained. For j in $\{1, \ldots, n\}$ there exist a unique flag f' which differs from f only in position j. This flag is obtained by the function

`FlagDisplacement:=function(FlagEXT, EXT, FAC, iMov)`

with `EXT` the set of extreme rays, `FAC` the set of facets, `iMov` the movement index j, and `FlagEXT` the flag f with each face f_i encoded by the list of extreme rays contained in it.

Another use of face lattice computation is in topology for computing boundary operator. The command is then

`BoundaryOperatorsCellularDecompositionPolytope:=function(GroupEXT, EXT, Boundin`

with `EXT` the list of vertices, `GroupEXT` the group acting on the vertices, `BoundingSet` a list of valid inequality (usually set to `[]`) and `kSought` the level at which one wants to compute the skeleton.

6. Lattices functions

6.1. **Basic lattices functions.** By a lattice, we mean a set $\mathbb{Z}v_1 + \cdots + \mathbb{Z}v_n \subset \mathbb{R}^n$. The `polyhedral` package allows to make all sort of computations with it, but the main computation tool is named Gram matrix. It is the positive definite matrix $G = (\langle v_i, v_j \rangle)_{1 \leq i,j \leq n}$ formed by all pairwise scalar products. For more on this point look for example at [7, 2, 10].

(1) The functions

`GetSuperlattices:=function(GramMat, GRP, TheMod)`
`GetSublattices:=function(GramMat, GRP, TheMod)`

enumerates super and sub-lattices of a given lattice with quotient $\mathbb{Z}/TheMod\mathbb{Z}$.

(2) The functions

`ShortVectorDutourVersion:=function(GramMat, eNorm)`
`ShortestVectorDutourVersion:=function(GramMat)`

enumerate the short vector or shortest vectors of a given lattice.

(3) The functions

`SphericalDesignLevel:=function(EXT, GramMat)`
`SphericalDesignLevelGroup:=function(EXT, GroupEXT, GramMat)`

Determine the t-design level of a set of points on a sphere (program originally written by F. Vallentin)

(4) The first function call

```
ClassicalSporadicLattices("E6");
ClassicalSporadicLattices(["ListNames"]);
```

return a Gram matrix of the lattice E_6 while the second call gives all available names.

(5) The function

`ProcEnum:=ProcEnum_SublatticeEnumeration();`

gives functionalities for enumerating sublattice of specific dimension and rank of a given lattice (see the examples for practice).

(6) The function

`ArithmeticAutomorphismGroup:=function(ListGram)`

determine the group of automorphism group of a list `ListGram` of positive definite matrices. That is if `ListGram=[A1, ..., Ap]` then it return the generators of the group

$$\{P \in \mathrm{GL}_n(\mathbb{Z}) \text{ s.t. } PA_1P^T = A_1, \ldots, PA_pP^T = A_p\}$$

The corresponding function for arithmetic isomorphism is

`ArithmeticIsomorphism:=function(ListGram1, ListGram2)`

That is if `ListGram=[A1, ..., Ap]` and `ListGram=[A'1, ..., A'p]` then it return one matrix $P \in \mathrm{GL}_n(\mathbb{Z})$ such that

$$PA_1P^T = A'_1, \ldots, PA_pP^T = A'_p$$

The command uses `AUTOM/ISOM` by B. Souvignier and W. Plesken to make those computations.

6.2. **Delaunay polytopes in lattices.** A Delaunay polytope P in a lattice L is the convex hull of $L \cap S(c,r)$ with $S(c,r)$ the sphere of center c and radius r having no interior point.

Delaunay polytopes form a face-to-face tiling of $\mathbb{R}^n$ and are useful for many applications. The main function call is

```
ListFunc:=DelaunayComputationStandardFunctions(TheGramMat);
```

it computed the Delaunay tesselation and returns a record that contains several functions for mutiple usage:

(1) The function

```
DelCO:=ListFunc.GetDelaunayDescription();
```

It gives a list of record, each one of them corresponding to an orbit of Delaunay polytopes. Each such record contains the list of vertices(`EXT`), the stabilizer (`TheStab`) of the Delaunay presented as a permutation group with a group homomorphism to the corresponding matrix group and the list of orbit of adjacent Delaunay polytopes (`Adjacencies`). That is the dual description of the Delaunay polytope is computed and for each facet, the adjacent Delaunay polytope is given. This information is encoded by a record

```
rec(iDelaunay:=..., eBigMat:=..., eInc:=....)
```

where `iDelaunay` is the orbit number of the adjacent Delaunay, `eBigMat` is the matrix that maps the canonical representative to the Delaunay polytope actually adjacent and `eInc` is the list of vertices incident to the facet defining the adjacency.

(2) The function

```
ListFunc.GetFreeVectors();
```

returns the list of free vectors of the lattice (see [9] for details).

(3) The function

```
ListFunc.GetVoronoiVertices();
```

return the vertices of the Delaunay polytope around 0.

(4) The functions

```
ListFunc.GetRigidityDegree();
ListFunc.GetLspace();
```

return the rigidity degree and the Lspace of the Delaunay tesselation (see [4] for details).

(5) The function

```
ListFunc.GetQuantization();
```

returns the quantization integral of the lattice (see [8] for details).

(6) The function

```
ListFunc.GetCoveringDensity();
```

returns the covering density of the lattice.

(7) The function

```
ListFunc.FlagFunctions();
```

computes the flag system of the Delaunay tesselation and in particular the Delaney symbol (see [3] for details).

(8) The function

```
ListFunc.GetOrbitDefiningVoronoiFacetInequalities();
```

returns the Voronoi vectors of the lattice.

(9) The function

```
LFC:=ListFunc.GetNeighborhood(EXT);
```

returns the functionalities for finding the neighborhood of a Delaunay polytope. (see Example)

For periodic point sets, there are similar functions.

```
ListFunc:=Periodic_DelaunayComputationStandardFunctions(U);
```

where `U` is a record that must contain `GramMat` and `ListCosets`.

6.3. **Perfect form.** A form is called perfect if it is uniquely defined by its shortest vectors. There is a large litterature on perfect forms since they have multiple uses in Mathematics.

(1) The function

```
MossContraction:=function(GramMat)
```

returns the Moss Contracted form from the positive definite matrix `GramMat`. See Proposition 1.2 in [12] for details.

6.4. **Voronoi polytope for polyhedral norms.** In Examples/VoronoiL1 an example of a polytope with

References

[1] D. Bremner, M. Dutour Sikirić and A. Schürmann, *Polyhedral representation conversion up to symmetries*, CRM proceedings & Lecture Notes **48** (2009) 45–72.

[2] J.H. Conway and N.J.A. Sloane, *Sphere Packings, Lattices and Groups* third edition, volume 290 of *Grundlehren der mathematischen Wissenschaften*, Springer–Verlag, 1998.

[3] O. Delgado Friedrichs, A. Dress and D. Huson, *Tilings and symbols: a report on the uses of symbolic calculation in tiling theory*, Sém. Lothar. Combin. 34 (1995), Art. B34a, approx. 16 pp.

[4] M. Deza and V.P. Grishukhin, *Nonrigidity degree of root lattices and their duals*, Geom. Dedicata 104 (2004), 1524.

[5] M. Deza and M. Dutour, *Cones of metrics, hemi-metrics and super-metrics*, Annals of the European Academy of Sciences (2003) 141–162.
[6] M. Dutour Sikirić, D. Bremner and D. Pasechnik, *The symmetry groups of polytopes*, in preparation
[7] M. Dutour Sikirić, F. Vallentin and A. Schürmann, *Classification of eight-dimensional perfect forms*, Electronic Research Annoucements of the AMS **13** (2007) 21–32.
[8] M. Dutour Sikirić, A. Schürmann and F. Vallentin, *Complexity and algorithms for computing Voronoi cells of lattices*, Mathematics of computation **78** (2009) 1713–1731.
[9] V.P. Grishukhin, *Free and nonfree Vorono polyhedra*. (Russian. Russian summary) Mat. Zametki **80** (2006) 367–378; translation in Math. Notes **80** (2006) 355365
[10] J. Martinet, *Perfect lattices in Euclidean spaces*, Springer, 2003.
[11] B.D. McKay, *The nauty program*, http://cs.anu.edu.au/people/bdm/nauty/.
[12] K.N., Moss, *Homology of $SL(n, Z[1/p])$*, Duke Math. J. 47 (1980), no. 4, 803818
[13] W. Plesken and B. Souvignier, *Computing isometries of lattices*, J. Symbolic Comput. **24** (1997) 327–334.

Mathieu Dutour Sikirić, Rudjer Bosković Institute, Bijenicka 54, 10000 Zagreb, Croatia
E-mail address: mdsikir@irb.hr