Normaliz 2.5

Winfried Bruns, Bogdan Ichim and Christof Söger

winfried@math.uos.de
bogdan.ichim@imar.ro
csoeger@math.uos.de

Contents

1 Introduction

1.1 The objectives of Normaliz

The program Normaliz, version 2.5, is mainly a tool for computing the Hilbert basis of a rational cone. The rational cone can be given by

(1) a system of generators $\mathscr{G}$ in a lattice $\mathbb{Z}^n$;
(2) constraints: a homogeneous linear system of equations and inequalities;
(3) generators and relations.

The Hilbert basis of a rational pointed cone C in $\mathbb{R}^n$ is defined with respect to a lattice $L \subset \mathbb{Z}^n$: it is the unique minimal system of generators of the monoid $C \cap L$. The standard choice for L is $\mathbb{Z}^n$ itself, but for Normaliz this choice can be modified in two ways:

(1) L can be chosen to be the sublattice of $\mathbb{Z}^n$ generated by $\mathscr{G}$;
(2) L can be chosen to be the lattice of solutions of a homogeneous system of congruences if the cone is specified by equations and inequalities.

In particular, Normaliz solves combined systems of homogeneous diophantine linear equations, inequalities and congruences. (An extension to nonhomogeneous systems is envisaged.) Conversely, Normaliz computes a system of constraints defining the cone and the lattice for which the Hilbert basis has been computed.

Normaliz has special input types for lattice polytopes (represented by their vertices) and monomial ideals (represented by the exponent vectors of their generators).

The data computed by Normaliz can be augmented if the monoid is homogeneous in a certain sense (see Section 3.6): if asked to do so, Normaliz computes the h-vector and Hilbert polynomial of the monoid (or its associated algebra).

On the other hand, the data computed can also be restricted, for example to the support hyperplanes of the cone or the lattice points of a lattice polytope.

For the mathematical background we refer the reader to [2] and [4]. The terminology follows [2]. For algorithms of Normaliz see [5] and [6]. Some of the recent extensions from version 2.2 to 2.5 are discussed in [3].

The input syntax of Normaliz is always kept backward compatible so that input files for older versions can still be used.

1.2 Access from other systems

We provide a SINGULAR library `normaliz.lib` and the package `Normaliz.m2` for MACAULAY2 that make Normaliz accessible from these systems. Thus SINGULAR or MACAULAY2 can be used as a comfortable environment for the work with Normaliz, and, moreover, Normaliz can be applied directly to objects belonging to the classes of toric rings and monomial ideals.

Normaliz has been made accessible from POLYMAKE (thanks to Andreas Paffenholz).

1.3 Major changes relative to version 2.2

(1) Two new input types for congruences and lattice ideals,
(2) a Hilbert basis algorithm using partial triangulations,
(3) a computation mode restricted to lattice points of polytopes,
(4) parallelization for shared memory systems (if wanted by the user),
(5) significant improvement of the shelling algorithm,
(6) overall improvement in time and memory usage,
(7) reorganization of the main output file,
(8) the graphical user interface jNormaliz by Vinicius Almendra and Bogdan Ichim,
(9) abolition of the setup file and of `norm32`.

1.4 Future extensions

(1) Inhomogeneous systems of equations, inequalities and congruences,
(2) a programming interface,
(3) further development of algorithms,
(4) exploitation of symmetries,
(5) optimization of the source code,
(6) more general h-vector computation,
(7) access from further systems.

2 Getting started

Download

- the zip file with the Normaliz source, documentation, examples and further platform independent components, and
- zip file made with executables for your system

from the Normaliz website

`http://www.math.uos.de/normaliz`

and unzip both in the same directory of your choice. In it, a directory `Normaliz2.5` (called Normaliz directory in the following) is created with several subdirectories. (Some versions of the Windows executables may need the installation of a runtime library; see website.)

In the Normaliz directory open jNormaliz by clicking `jNormaliz.jar` in the appropriate way. (We assume that Java is installed on your machine.) In the jNormaliz file dialogue choose one of the input files in the subdirectory `example`, say `medium.in`, and press `Run`. In the console window you can watch Normaliz at work. Finally inspect the output window for the results.

The menus and dialogues of jNormaliz are self explanatory, but you can also consult the documentation [1] via the help menu.

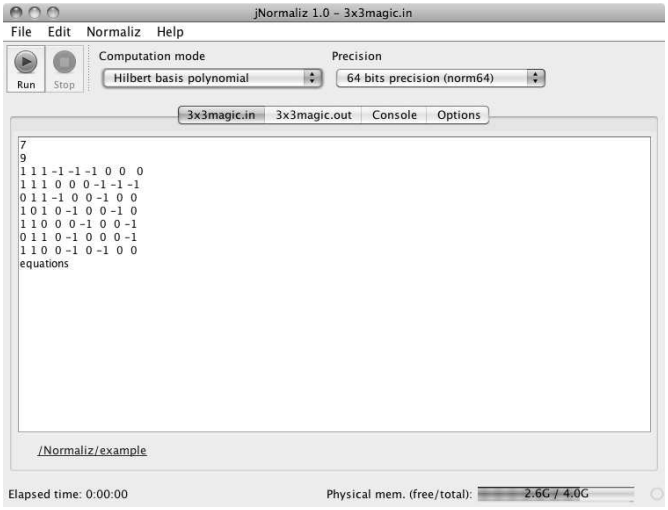

Figure 1: jNormaliz

If the executables prepared cannot be run on your system, then you can download the source files and compile Normaliz yourself (see Section 10).

Moreover, one can, and often will, run Normaliz from the command line. This is explained in Section 4.

If 64 bit integer precision is not sufficient, then one can use `normbig` instead of `norm64`. `normbig` has no restrictions on the integer precision. See Section 4.4. (The integer precision has nothing to do with the address width (32 bit or 64 bit) of your operating system.)

3 The input file

The input file `<projectname>.in` consists of one or several matrices (in version 2.5). Each matrix is built as follows:

(1) The first line contains the number of rows *m*.
(2) The second contains the number of columns *n*.
(3) The next *m* lines of *n* integers each contain the rows.
(4) The last line contains a single integer or word specifying the type of input the matrix presents.

At the moment there are three major types of input matrices, namely *generators*, *constraints*, and *relations*.

For each input type we specify two lattices: the *ambient lattice* $\mathbb{A}$ in which the Hilbert basis "lives" and the *essential lattice* $\mathbb{E} \subset \mathbb{A}$ which is generated by the Hilbert basis.

In this section we assume that Normaliz is run in a computation mode in which the Hilbert basis is actually computed. (See Section 4 for computation modes.)

3.1 Generators

The generator types are 0, 1, 2 and 3. If a matrix of one of these types is in the input file, then it must be the only matrix in the file.

3.1.1 Type 0, `integral_closure`

The rows of an $m \times n$ matrix of type 0 represent m vectors in the ambient lattice $\mathbb{A} = \mathbb{Z}^n$. The essential lattice $\mathbb{E}$ is the smallest direct summand of $\mathbb{Z}^n$ that contains the vectors in the matrix.

The vectors are considered as a system of generators $\mathscr{G}$ of a cone C, and Normaliz computes the Hilbert basis of C with respect to $\mathbb{E}$ (or, equivalently, $\mathbb{Z}^n$).

The nomenclature `integral_closure` is explained by the fact that the Hilbert basis generates the integral closure of the monoid $\mathbb{Z}_+\mathscr{G}$ in $\mathbb{Z}^n$.

A simple example:

```
Input             Hilbert basis
3                 1 0
2                 0 1
2 0
1 1
0 2
integral_closure
```

In this example, the three input vectors clearly generate the positive orthant $\mathbb{R}^2_+$ in $\mathbb{R}^2$, and the two unit vectors clearly are the Hilbert basis of $\mathbb{R}^2_+ \cap \mathbb{Z}^2$.

3.1.2 Type 1, `normalization`

The matrix is interpreted as in type 0, however $\mathbb{E}$ is chosen as the sublattice of $\mathbb{Z}^n$ generated by $\mathscr{G}$.

The choice of the name `normalization` indicates that Normaliz computes the normalization of the monoid $\mathbb{Z}_+\mathscr{G}$. (The computation of such normalizations was the original goal of Normaliz, hence the name.)

We choose the same input vectors as above, but change the type to `normalization`:

```
Input             Hilbert basis
3                 2 0
2                 1 1
2 0               0 2
1 1
0 2
normalization
```

The cone has not changed, but the lattice has: $\mathbb{E}$ is now the sublattice of $\mathbb{Z}^2$ of all (z_1, z_2) with $z_1 + z_2 \equiv 0 \mod 2$.

3.1.3 Type 2, `polytope`

The rows of the matrix are interpreted as integral points of a lattice polytope in $\mathbb{R}^n$, which is their convex hull.

The cone C is the cone over the polytope, i.e. the cone with apex 0 in $\mathbb{R}^{n+1}$ generated by the vectors $(x,1)$ where x represents a row of the input matrix. We want to compute the *Ehrhart monoid* $C \cap \mathbb{Z}^{n+1}$.

The lattice $\mathbb{A}$ is $\mathbb{Z}^{n+1}$, and $\mathbb{E}$ is the smallest direct summand of $\mathbb{A}$ containing the generators of C.

Type 2 is only a variant of type 0. One obtains the same results as in type 0 with the extended vectors $(x,1)$ as input. However, the text in the output file is adapted to the polytopal situation. For an example, see Section 6.

3.1.4 Type 3, `rees_algebra`

In this type the input vectors are considered as exponent vectors of the generators of a monomial ideal I in the polynomial ring $K[X_1,\dots,X_n]$. Normaliz computes the normalization of the Rees algebra of the ideal I (see [4] for the notion of Rees algebra.) This is a monomial subalgebra of the extended polynomial ring $K[X_1,\dots,X_n,T]$ with an auxiliary variable T. Normaliz computes the exponent vectors in $\mathbb{Z}^{n+1}$ of the system of generators. For an example, see Section 6.

In type 3 one has $\mathbb{A} = \mathbb{E} = \mathbb{Z}^{n+1}$.

3.1.5 Preparation of the generators

After the coordinate transformation to the lattice $\mathbb{E}$, Normaliz divides each generator by the greatest common divisor of its components. For example, the extreme rays listed will always be such divided vectors (re-transformed to $\mathbb{A}$).

3.2 Constraints

Inequalities, equations, and congruences defining the cone and the lattice are called constraints. Matrices representing them are of types 4, 5 and 6. All three types can be present in the input file, and there can be several matrices of each type. The order does not matter. Matrices of the same type will be concatenated. The numbers of columns must of course match: for the ambient lattice $\mathbb{Z}^n$ the matrices of types 4 and 5 must have n columns, and those of type 6 must have $n+1$ columns.

If there is no matrix of type 4, then it is assumed that the user wants to compute the nonnegative solutions of the system represented by the matrices of type 5 and/or 6. The input file is therefore compatible with the types 4 and 5 of previous versions of Normaliz.

3.2.1 Type 4, `hyperplanes`

A row $(\xi_1,\dots,\xi_n)$ of the input matrix of type 4 represents an inequality

$$\xi_1 x_1 + \cdots + \xi_n x_n \geq 0$$

for the vectors $(x_1,\dots,x_n)$ of $\mathbb{R}^n$.

Example:

```
Input             Hilbert basis
2                 0 -1
2                 1  1
1 0
1 -1
hyperplanes
```

Normaliz has computed the Hilbert basis of the cone defined by the inequalities $x_1 \geq 0$ and $x_1 - x_2 \geq 0$ with respect to the lattice $\mathbb{Z}^2$.

3.2.2 Type 5, `equations`

A row $(\xi_1,\dots,\xi_n)$ of the input matrix of type 5 represents an equation

$$\xi_1 x_1 + \cdots + \xi_n x_n = 0$$

for the vectors $(x_1,\dots,x_n)$ of $\mathbb{R}^n$.

Example:

```
Input             Hilbert basis
1                 2 0 1
3                 0 2 1
1 1 -2            1 1 1
equations
```

If the input file contains no further matrices, Normaliz has computed the Hilbert basis of the subcone of $\mathbb{R}^3_+$ defined by the equation $x_1 + x_1 - 2x_3 = 0$.

3.2.3 Type 6, `congruences`

We consider the rows of a matrix of type 6 to have length $n+1$. Each row $(\xi_1,\dots,\xi_n,c)$ represents a congruence

$$\xi_1 z_1 + \cdots + \xi_n z_n \equiv 0 \mod c$$

for the elements $(z_1,\dots,z_n) \in \mathbb{Z}^n$.

Example:

```
Input             Hilbert basis
1                 2 0
3                 1 1
1 1 2             0 2
congruences
```

If no other matrix is in the input file, then Normaliz computes the Hilbert basis of the positive orthant intersected with the lattice of all integral vectors (z_1, z_2) such that $z_1 + z_2 \equiv 0 \mod 2$ and the result is the same as in 3.1.2 above.

3.2.4 The constraints combined

Let L be the sublattice of $\mathbb{Z}^n$ that consists of the solutions of the system of congruences defined by the input matrix of type 6 ($L = \mathbb{Z}^n$ if there is no matrix of type 6). Moreover let A be the matrix of type 4 and B be the matrix of type 5. Then the cone C is given by

$$C = \{x \in \mathbb{R}^n : Ax \geq 0,\ Bx = 0\}.$$

and the Hilbert basis of $C \cap L$ is computed.

The ambient lattice $\mathbb{A}$ is $\mathbb{Z}^n$, and the essential lattice is $\mathbb{E} = L \cap \mathbb{R}C$.

If there is no matrix of type 5, then the system of equations is empty, satisfied by all vectors of $\mathbb{R}^n$.

Note that there is always a matrix of type 4, either explicitly in the input, or implicitly, namely the $n \times n$ unit matrix, if there is no matrix of type 4 in the input file (but one of type 5 or 6).

See Section 6.2.3 for an example combining types 5 and 6.

3.3 Relations

Relations are another type of constraints. They do not select a sublattice of $\mathbb{Z}^n$ or a subcone of $\mathbb{R}^n$, but define a monoid as a quotient of $\mathbb{Z}^n_+$ modulo a system of congruences (in the semigroup sense!).

Let U be a subgroup of $\mathbb{Z}^n$. Then the natural image M of $\mathbb{Z}^n_+ \subset \mathbb{Z}^n$ in the abelian group $G = \mathbb{Z}^n/U$ is a submonoid of G. In general, G is not torsionfree, and therefore M may not be an affine monoid. However the image N of M in the lattice $L = G/\text{torsion}$ is an affine monoid. Normaliz chooses an embedding $L \hookrightarrow \mathbb{Z}^r$, $r = n - \operatorname{rank} U$, such that N becomes a submonoid of $\mathbb{Z}^r_+$. In general there is no canonical choice for such an embedding, but one can always find one, provided N has no invertible element except 0. The ambient lattice is then $\mathbb{A} = \mathbb{Z}^r$, and the essential lattice is L, realized as a sublattice of $\mathbb{A}$.

The typical starting point is an ideal $J \subset K[X_1, \ldots, X_n]$ generated by binomials

$$X_1^{a_1} \cdots X_n^{a_n} - X_1^{b_1} \cdots X_n^{b_n}.$$

The image of $K[X_1, \ldots, X_n]$ in the residue class ring of the Laurent polynomial ring $S = K[X_1^{\pm 1}, \ldots, X_n^{\pm 1}]$ modulo the ideal JS is exactly the monoid algebra $K[M]$ of the monoid M above if we let U be the subgroup of $\mathbb{Z}^n$ generated by the differences

$$(a_1, \ldots, a_n) - (b_1, \ldots, b_n).$$

Ideals of type JS are called lattice ideals if they are prime. Since Normaliz automatically passes to $G/\text{torsion}$, it replaces JS by the smallest lattice ideal containing it.

3.3.1 Type 10, `lattice_ideal`

The rows of the input matrix of type 10 are interpreted as generators of the subgroup U, and Normaliz performs the computation as just explained.

As an example we consider the binomials $X_1X_3 - X_2^2$, $X_1X_4 - X_2X_3$:

```
Input            Hilbert basis
2                3 0
4                2 1
1 -2  1 0        1 2
1 -1 -1 1        0 3
lattice_ideal
```

In this example $\mathbb{Z}^4/U$ is torsionfree, but we can replace each of the vectors in the input matrix by a nonzero integral multiple without changing the result.

Type 10 cannot be combined with any other input type—such a combination would not make sense.

3.4 Pointedness

For Hilbert basis computations and triangulations Normaliz requires the cone to be pointed ($x, -x \in C \implies x = 0$). Whenever the condition of pointedness is violated at a step where it is crucial, Normaliz will stop computations.

Pointedness is checked by testing whether the dual cone of C is full dimensional, and if not, then the constructor of the dual cone complains as follows:

```
Full Cone error: Matrix with rank = number of columns needed in the
constructor of the object Full_Cone.
Probable reason: Cone not full dimensional(<=> dual cone not pointed)!
```

3.5 The zero cone

The zero cone with an empty Hilbert basis is a legitimate object for Normaliz. Nevertheless a warning message is issued if the zero cone is encountered.

3.6 Homogeneity

In certain cases Normaliz can compute the h-vector and the Hilbert polynomial of a graded monoid. A *grading* of a monoid M is simply a homomorphism $\deg : M \to \mathbb{Z}^g$ where $\mathbb{Z}^g$ contains the degrees. The *Hilbert series* of M with respect to the grading is the formal Laurent series

$$\sum_{d \in \mathbb{Z}^g} \#\{x \in M : \deg x = d\} T_1^{d_1} \cdots T_g^{d_g},$$

provided all sets $\{x \in M : \deg x = d\}$ are finite. At the moment, Normaliz can only handle the case $g = 1$ if the monoid is *homogeneous* in the following sense: deg is a linear form on the essential lattice $\mathbb{E}$ such that $\deg x = 1$ for all extreme integral generators in the Hilbert basis. If such a linear form exists, it is uniquely determined, and Normaliz finds it.

Homogeneity is always satisfied for lattice polytopes. The Rees algebra is homogeneous in our sense if and only if all the monomials generating the ideal have the same total degree.

Instead of degree we will use *height* in the following because of its geometric flavor.

Note that the notion of homogeneity used here is more general than previous versions of Normaliz. Its use is compatible with that in [2], provided one refers to the monoid generated by the extreme integral generators.

4 Running Normaliz

The syntax for calling Normaliz from the command line is

```
norm64 [-svnh1pSVNdafce] [-x=<T>] [<projectname>]
```

where the options and `<projectname>` are optional. (We assume that the executable `norm64` or `norm64.exe` is in the search path. Otherwise you have to prefix it with a suitable relative or absolute path.) If no `<projectname>` is given, the program will ask you for it or display a help screen.

The option `-x` differs from the other ones: `<T>` represents a positive number assigned to `-x`; see Section 4.3.

The help screen can also be displayed by `norm64 -?`.

Normaliz will look for `<projectname>.in` as input file.

For example, if you input the command

```
norm64 -c -p -a rafa2416     or     norm64 -cpa rafa2416
```

then the program will take the file `rafa2416.in` as input, control data will be displayed on your terminal, the support hyperplanes, the triangulation, the multiplicity, the *h*-vector and the Hilbert polynomial will be computed and all the possible output files will be produced.

If you have inadvertently typed `rafa2416.in` as the project name, then Normaliz will first look for `rafa2416.in.in` as the input file. If this file doesn't exist, `rafa2416.in` will be loaded.

In the following we explain the various options of Normaliz. The full text names given appear in the help screen as well as in the menus of jNormaliz which allows you to choose options interactively.

The default computation mode is `-n`. All options that can be activated are switched off by default.

Options are evaluated from left to right. Therefore the last of mutually exclusive options is used.

4.1 Computation modes

4.1.1 Standard modes

The standard, ascending chain of computation modes is the following:

- `-s support hyperplanes`: only the support hyperplanes of the cone under consideration are computed.
- `-v triangulation`: includes `-s`. In addition, Normaliz computes a triangulation and the multiplicity in the homogeneous case. (For lattice polytopes this is the normalized volume.)
- `-n Hilbert basis triangulation` (previously `normal`): includes `-v`. Normaliz computes the Hilbert basis.
- `-h Hilbert basis polynomial`: includes `-n`. In the homogeneous case, Normaliz computes the *h*-vector and the Hilbert polynomial. This computation mode yields the maximum information Normaliz can produce.

If only the *h*-vector is to be computed, then one uses

`-p Hilbert polynomial`

This mode is much faster than `-n`. It also computes the height 1 elements of the Hilbert basis.

Finally, for the application to lattice polytopes (but also for other homogeneous cases), the computation of Hilbert bases can be restricted to the height 1 elements (without the *h*-vector):

-1 `height 1 elements`: the same as `-n`, but only height 1 elements are computed, using a partial triangulation. See also `-N` below.

The last mode is again faster than `-p`.

4.1.2 Computation modes for large examples

For some challenging examples it has proved extremely efficient to avoid the computation of full triangulations. See [3] and Section 8. The partial triangulation type for the computation of Hilbert bases is activated by

`-N Hilbert basis`.

The only loss of `-N` in comparison with `-n` is that one cannot compute the multiplicity in the homogeneous case (and looses the full triangulation). Therefore, if only the Hilbert basis is of interest, it may be a good idea to use `-N`.

The following input files in the `example` subdirectory should be processed with the option `-N`:

- `A443.in`, `A543.in`, `A553.in`, `A643.in`, `semigraphoid5.in`.

See Section 8 for an indication of computation times. The first of these examples, `A443.in`, can also be run with `-n` (the default) or `-h`.

Moreover, we have implemented variants of -s and -v that are faster for sufficiently large examples.

-S support hyperplanes pyramids: the same as -s based on a different algorithm,
-V triangulation pyramids: similar replacement for -v.

The first example for which -S yields a better computation time than -s is A553.in, but already A443.in shows that -V is advantageous from a certain order of magnitude on.

We plan to develop an automatic choice of algorithms and an extension of the ideas behind -S and -V.

4.1.3 The dual algorithm

In types 4, 5 and 6 it is often faster to use a Hilbert basis algorithm originally due to Pottier [7] that we call the *dual* algorithm, in contrast to the *primal* (triangulation based) algorithm of Normaliz. (See [5] for our version of the dual algorithm.) The dual algorithm is invoked by

-d dual

See Section 8 for a comparison of performance on various examples.

4.2 Control of output files

In the default setting Normaliz writes only the output file <projectname>.out. The amount of output files can be increased in two steps:

-f Normaliz writes the additional output files with suffixes gen, cst, inv and typ, provided the data of these files have been computed.
-a Normaliz writes all available output files.

For the list of potential output files and their interpretation see Section 7.

4.3 Control of execution

The options that control the execution are:

-c activates the verbose ("control") behavior of Normaliz in which Normaliz writes additional information about its current activities to the standard output.
-e activates the overflow error check of norm64. Ignored by normbig.
-x=<T> There <T> stands for a positive integer limiting the number of threads that Normaliz is allowed access on your system. The default value is set by the operating system. If you want to run Normaliz in a strictly serial mode, choose <T>= 1.

The number of threads can also be controlled by the environment variable OMP_NUM_THREADS. See Section 8 for further discussion.

The options -i and -m of version 2.2 have become obsolete. They will be ignored if present.

4.4 Numerical limitations

Even in low dimensions, the range of 64 bit integers may not be sufficient for the computations of Normaliz. Therefore we provide an indefinite precision executable `normbig` in addition to `norm64`.

Computations with `normbig` typically run about 5 times slower than those with `norm64`. In examples that look critical, it may be useful first to try `norm64` with the error check option activated. This costs time, too, but hardly more than 50% extra.

The user should run the example `critical64.in` in the subdirectory `examples` with `norm64 -e` in order to see the failure of 64 bit arithmetic. (Running it with `normbig` takes a while.)

Another way of checking `norm64` by `normbig` in the homogeneous case is to have `norm64 -h` followed by `normbig -p` and to compare *h*-vectors.

Note: The Hilbert polynomial is computed by `norm64` only if the rank is ≤ 21 since $20!$ is the largest factorial representable in 64 bit arithmetic.

5 The output file

The data you will find in the output file depend on the input type and on the computation mode. The output file starts with an "abstract" that collects various numerical and qualitative data, for example the number of elements in the Hilbert basis. The lists of vectors, equations etc. follow the abstract.

In types $\neq 2, 3$ the output file `<projectname>.out` may contain the following data:

- only for type 10: the original system of generators (see below);
- the Hilbert basis H computed;
- the extreme rays of the cone C generated by H;
- the rank of the lattice $\mathbb{E}$;
- the index of the lattice generated by the original input vectors in $\mathbb{E}$;
- the support hyperplanes of C;
- a system of equations defining the vector space generated by C;
- a system of congruences defining $\mathbb{E}$ as a sublattice of $\mathbb{A}$ (together with the equations);

In the homogeneous case the following extra data may be printed:

- the linear form defining the degree;
- the height 1 elements of the Hilbert basis;
- the multiplicity;
- the *h*-vector and the coefficients of the Hilbert polynomial.

The (non)homogeneous case is indicated by the statement that the extreme rays are (not) homogeneous. If the whole Hilbert basis is of height 1, this is indicated as well (despite of the fact that it can be concluded from the numerical data). Moreover, Normaliz tells you whether the original system of generators contains the Hilbert basis by indicating whether the original monoid is integrally closed.

Please note:

(1) The equations and support hyperplanes *together* define the cone C. While support hyperplanes will be always present (except for the zero cone), equations will only be printed if necessary, namely when $\dim C < \operatorname{rank} \mathbb{A}$.
Similarly, congruences will only be printed if the lattice $\mathbb{E}$ is not given by $\mathbb{R}C \cap \mathbb{A}$. This can only happen with input matrices of type 1 or 6. The lattice $\mathbb{E}$ is defined simultaneously by the equations and the congruences.
Even if the cone and the lattice are defined by constraints, the inequalities, equations and congruences will in general not be reproduced, but replaced by an equivalent system.

(2) The extreme rays are given by the first points in $\mathbb{E}$ on them (the extreme integral generators with respect to $\mathbb{E}$).

(3) In order to lift the linear form defining the degree from $\mathbb{E}$ to $\mathbb{A}$ it may be necessary to replace it by a multiple (in order to avoid fractions as coefficients). In this case the evaluation of the linear form on the extreme rays will yield a degree > 1. The h-vector and the Hilbert polynomial do always refer to the degree in $\mathbb{E}$.

(4) Input matrices of types 0,1, 2 or 3 contain an explicit system of generators. For the other types $\neq 10$ the extreme rays computed by Normaliz take their place. For type 10 Normaliz first computes the monoid M generated by the residue classes of the canonical basis of $\mathbb{Z}^n$ (compare Section 3.3), and they are considered the original system of generators.

If type $= 2$ (`polytope`), the following data may be found in the output file:

- the Hilbert basis of the Ehrhart monoid;
- the lattice points of the polytope;
- the dimension of the polytope;
- the extreme points;
- the support hyperplanes;
- a system of equations defining the affine hull of the polytope;
- the normalized volume;
- the h-vector and the coefficients of the Ehrhart polynomial.

In type $= 3$ (`rees_algebra`), the output file may contain the following:

- the generators of the integral closure $\overline{\mathscr{R}}$ of the Rees algebra;
- the extreme rays;
- the generators of the integral closure of the ideal;
- the support hyperplanes;
- if the ideal is primary to the irrelevant maximal ideal, the multiplicity of the ideal (not to be confused with the multiplicity of the monoid).

In the homogeneous case the following extra data may be printed:

- the linear form defining the degree;
- the height 1 elements of the Hilbert basis;
- the multiplicity (of the monoid);
- the h-vector and the coefficients of the Hilbert polynomial.

6 Examples

6.1 Generators

6.1.1 Type 0, `integral_closure`

The file `rproj2.in` contains the following (here typeset in 2 columns):

```
16
 7
1 0 0 0 0 0 0          1 0 1 0 1 0 1
0 1 0 0 0 0 0          1 0 0 1 0 1 1
0 0 1 0 0 0 0          1 0 0 0 1 1 1
0 0 0 1 0 0 0          0 1 1 0 0 1 1
0 0 0 0 1 0 0          0 1 0 1 1 0 1
0 0 0 0 0 1 0          0 1 0 0 1 1 1
1 1 1 0 0 0 1          0 0 1 1 1 0 1
1 1 0 1 0 0 1          0 0 1 1 0 1 1
                       0
```

This means that we wish to compute the Hilbert basis of the cone generated by the 16 vectors

$$(1,0,0,0,0,0,0)\,,\quad (0,1,0,0,0,0,0)\,,\quad \ldots\,,\quad (0,0,1,1,0,1,1)$$

in $\mathbb{R}^7$ with respect to the full lattice $\mathbb{Z}^7$, as indicated by the final digit that specifies the type. (Actually, the vectors generate the full lattice so that a replacement of type 0 by type 1 would not change anything.)

Running `norm64` with option `-h`, `Hilbert basis polynomial` produces the file `rproj2.out` which has the following content (partially typeset in 2 columns):

```
17 Hilbert basis elements
16 height 1 Hilbert basis elements
16 extreme rays
24 support hyperplanes

rank = 7 (maximal)
index = 1
original monoid is not integrally closed

extreme rays are homogeneous via the linear form:
1 1 1 1 1 1 -2

Hilbert basis elements are not homogeneous

multiplicity = 72

h-vector:
```

```
1 9 31 25 6 0 0

Hilbert polynomial:
1/1 97/30 71/15 49/12 13/6 41/60 1/10

**********************************************************************

17 Hilbert basis elements:
 0 0 0 0 0 1 0
 0 0 0 0 1 0 0
 0 0 0 1 0 0 0
 0 0 1 0 0 0 0
 0 0 1 1 0 1 1
 0 0 1 1 1 0 1
 0 1 0 0 0 0 0
 0 1 0 0 1 1 1
 0 1 0 1 1 0 1
 0 1 1 0 0 1 1
 1 0 0 0 0 0 0
 1 0 0 0 1 1 1
 1 0 0 1 0 1 1
 1 0 1 0 1 0 1
 1 1 0 1 0 0 1
 1 1 1 0 0 0 1
 1 1 1 1 1 1 2

16 extreme rays:
 1 0 0 0 0 0 0
 0 1 0 0 0 0 0
 0 0 1 0 0 0 0
 0 0 0 1 0 0 0
 0 0 0 0 1 0 0
 0 0 0 0 0 1 0
 1 1 1 0 0 0 1
 1 1 0 1 0 0 1
 1 0 1 0 1 0 1
 1 0 0 1 0 1 1
 1 0 0 0 1 1 1
 0 1 1 0 0 1 1
 0 1 0 1 1 0 1
 0 1 0 0 1 1 1
 0 0 1 1 1 0 1
 0 0 1 1 0 1 1

24 support hyperplanes:
  0  0  0  1  0  0  0
  0  0  0  0  1  0  0
  0  0  0  0  0  1  0
  0  0  0  0  0  0  1
  0  0  1  0  0  0  0
  0  1  0  0  0  0  0
  0  1  1  0  0  1 -1
  0  1  0  0  1  1 -1
  0  1  0  1  1  0 -1
  0  0  1  1  0  1 -1
  0  1  1  1  1  1 -2
  0  0  1  1  1  0 -1
  1  0  0  0  0  0  0
  1  1  1  1  1  1 -3
  1  0  0  0  1  1 -1
  1  0  0  1  0  1 -1
  1  0  1  0  1  0 -1
  1  0  1  1  1  1 -2
  1  1  1  0  0  0 -1
  1  1  1  1  0  1 -2
  1  1  0  1  0  0 -1
  1  1  1  0  1  1 -2
  1  1  1  1  1  0 -2
  1  1  0  1  1  1 -2

16 height 1 Hilbert basis elements:
 0 0 0 0 0 1 0
 0 0 0 0 1 0 0
 0 0 0 1 0 0 0       0 1 1 0 0 1 1
 0 0 1 0 0 0 0       1 0 0 0 0 0 0
 0 0 1 1 0 1 1       1 0 0 0 1 1 1
 0 0 1 1 1 0 1       1 0 0 1 0 1 1
 0 1 0 0 0 0 0       1 0 1 0 1 0 1
 0 1 0 0 1 1 1       1 1 0 1 0 0 1
 0 1 0 1 1 0 1       1 1 1 0 0 0 1
```

From this, we see that there are 17 elements in the Hilbert basis and 16 extreme rays, that the sublattice generated by the input vectors has index 1 in $\mathbb{Z}^7$, and that the corresponding

support hyperplanes are given by the linear forms $(0,0,0,1,0,0,0)$, $(0,0,0,0,1,0,0)$, ..., $(1,1,0,1,1,1,-2)$. We are also given the information that the monoid is homogeneous and that its multiplicity is 72.

Since we are in the homogeneous case, the height 1 elements of the Hilbert basis, the h-vector and Hilbert polynomial of the monoid generated by the Hilbert basis are also computed. The h-vector is

$$(h_0,h_1,\ldots,h_6) = (1,9,31,25,6,0,0)\,,$$

and the Hilbert polynomial is given by

$$P(k) = \frac{1}{1} + \frac{97}{30}k + \frac{71}{15}k^2 + \frac{49}{12}k^3 + \frac{13}{6}k^4 + \frac{41}{60}k^5 + \frac{1}{10}k^6\,.$$

The Hilbert polynomial gives the number of elements of degree k, starting from degree 0, as is always the case for normal monoids.

We omit an example of type 1 since it does not add anything new.

6.1.2 Type 2, `polytope`

The file `polytop.in`:

```
4
3
0 0 0
2 0 0
0 3 0
0 0 5
polytope
```

The Ehrhart monoid of the integral polytope with the 4 vertices

$$(0,0,0)\,,\quad (2,0,0)\,,\quad (0,3,0)\quad \text{and}\quad (0,0,5)$$

in $\mathbb{R}^3$ is to be computed. (Note the last line, indicating the polytope type 2.)

Running `norm64` with option `-h`, `Hilbert basis polynomial` produces the file `polytop.out`:

```
19 generators of Ehrhart ring
18 lattice points in polytope
4 extreme points of polytope
4 support hyperplanes

polytope is not integrally closed

dimension of the polytope = 3
normalized volume = 30

h-vector:
```

```
1 14 15 0

Ehrhart polynomial:
1/1 4/1 8/1 5/1

************************************************************************

19 generators of Ehrhart ring:          18 lattice points in polytope:
 0 0 0 1                                 0 0 0
 0 0 1 1                                 0 0 1
 0 0 2 1                                 0 0 2
 0 0 3 1                                 0 0 3
 0 0 4 1                                 0 0 4
 0 0 5 1                                 0 0 5
 0 1 0 1                                 0 1 0
 0 1 1 1                                 0 1 1
 0 1 2 1                                 0 1 2
 0 1 3 1                                 0 1 3
 0 2 0 1                                 0 2 0
 0 2 1 1                                 0 2 1
 0 3 0 1                                 0 3 0
 1 0 0 1                                 1 0 0
 1 0 1 1                                 1 0 1
 1 0 2 1                                 1 0 2
 1 1 0 1                                 1 1 0
 2 0 0 1                                 2 0 0
 1 2 4 2

4 extreme points of polytope:         4 support hyperplanes:
 0 0 0                                  -15 -10  -6 >=  -30
 2 0 0                                    1   0   0 >=    0
 0 3 0                                    0   1   0 >=    0
 0 0 5                                    0   0   1 >=    0
```

The desired lattice points are the 18 ones listed above. To complete the picture, we also provide all the generators of the Ehrhart monoid of the polytope. (There are 19 of them in this example.) Furthermore, the original polytope is the solution of the system of the 4 inequalities

$$x_3 \geq 0\,, \quad x_2 \geq 0\,, \quad x_1 \geq 0 \quad \text{and} \quad 15x_1 + 10x_2 + 6x_3 \leq 30\,,$$

and has normalized volume 30.

The last two lines provide the information that the h-vector of the Ehrhart ring is

$$(h_0, h_1, h_2, h_3) = (1, 14, 15, 0)\,,$$

and its Ehrhart polynomial of the polytope is

$$P(k) = 1 + 4k + 8k^2 + 5k^3\,.$$

6.1.3 Type 3, rees_algebra

Next, let us discuss the example rees.in:

```
 10
 6                          0 1 1 0 0 1
 1 1 1 0 0 0                0 1 0 1 1 0
 1 1 0 1 0 0                0 1 0 0 1 1
 1 0 1 0 1 0                0 0 1 1 1 0
 1 0 0 1 0 1                0 0 1 1 0 1
 1 0 0 0 1 1                rees_algebra
```

Comparing with the data in rproj2.in shows that rees is the origin of rproj2.

Here we want to compute the integral closure of the Rees algebra of the ideal generated by the monomials corresponding to the above 10 exponent vectors. The output in rees.out coincides with that in rproj2.out, up to notions and the supplementary information on the integral closure of the ideal:

```
10 generators of integral closure of the ideal:
 0 0 1 1 0 1
 0 0 1 1 1 0
 0 1 0 0 1 1                  1 0 0 1 0 1
 0 1 0 1 1 0                  1 0 1 0 1 0
 0 1 1 0 0 1                  1 1 0 1 0 0
 1 0 0 0 1 1                  1 1 1 0 0 0
```

A brief look at rproj2.out shows that exactly the generators with the last coordinate 1 have been extracted. (So the ideal is integrally closed. This is not surprising because we have chosen squarefree monomials.)

6.2 Constraints

6.2.1 Type 4, hyperplanes

The file dual.in is

```
 24
 7
 0 0 0 1 0 0  0             1 0 0 0 0 0  0
 0 0 0 0 1 0  0             1 1 1 1 1 1 -3
 0 0 0 0 0 1  0             1 0 0 1 0 1 -1
 0 0 0 0 0 0  1             1 0 0 0 1 1 -1
 0 0 1 0 0 0  0             1 0 1 0 1 0 -1
 0 1 0 0 0 0  0             1 0 1 1 1 1 -2
 0 1 0 1 1 0 -1             1 1 0 1 0 0 -1
 0 1 0 0 1 1 -1             1 1 1 0 0 0 -1
 0 1 1 0 0 1 -1             1 1 1 1 0 1 -2
```

```
 0 0 1 1 1 0 -1              1 1 1 0 1 1 -2
 0 0 1 1 0 1 -1              1 1 1 1 1 0 -2
 0 1 1 1 1 1 -2              1 1 0 1 1 1 -2
                             hyperplanes
```

This means that we wish to compute the Hilbert basis of the cone cut out from $\mathbb{R}^7$ by the 24 inequalities. (It is the dual of the cone spanned by the 24 linear forms in $(\mathbb{R}^7)^*$.). The inequalities represent exactly the support hyperplanes from the file `rproj2.out`. The output in `dual.out` coincides with that in `rproj2.out`.

6.2.2 Type 5, `equations`

Suppose that you have the following "square"

x_1	x_2	x_3
x_4	x_5	x_6
x_7	x_8	x_9

and the problem is to find nonnegative values for $x_1,\ldots,x_9$ such that the 3 numbers in all rows, all columns, and both diagonals sum to the same constant $\mathscr{M}$ (called the magic constant). This leads to a linear system of equations

$$\begin{aligned}
x_1+x_2+x_3 &= x_4+x_5+x_6;\\
x_1+x_2+x_3 &= x_7+x_8+x_9;\\
x_1+x_2+x_3 &= x_1+x_4+x_7;\\
x_1+x_2+x_3 &= x_2+x_5+x_8;\\
x_1+x_2+x_3 &= x_3+x_6+x_9;\\
x_1+x_2+x_3 &= x_1+x_5+x_9;\\
x_1+x_2+x_3 &= x_3+x_5+x_7.
\end{aligned}$$

This system of equations is contained in the file `3x3magic.in`. It ends with the input type `5`.

The output file contains the following:

```
5 Hilbert basis elements
5 height 1 Hilbert basis elements
4 extreme rays
4 support hyperplanes

rank = 3
index = 2
original monoid is not integrally closed

extreme rays are homogeneous via the linear form:
```

```
0 0 0 0 1 0 0 0 0

Hilbert basis elements are homogeneous

multiplicity = 4

h-vector:
1 2 1

Hilbert polynomial:
1/1 2/1 2/1

**********************************************************************

5 Hilbert basis elements:
 2 0 1 0 1 2 1 2 0
 1 0 2 2 1 0 0 2 1
 1 1 1 1 1 1 1 1 1
 1 2 0 0 1 2 2 0 1
 0 2 1 2 1 0 1 0 2

4 extreme rays:
 1 2 0 0 1 2 2 0 1
 2 0 1 0 1 2 1 2 0
 0 2 1 2 1 0 1 0 2
 1 0 2 2 1 0 0 2 1

4 support hyperplanes:
  0 -1  0  0  2  0  0  0  0
  0  1  2  0 -2  0  0  0  0
  0 -1 -2  0  4  0  0  0  0
  0  1  0  0  0  0  0  0  0

6 equations:
 -2  1  4 -3  0  0  0  0  0
 -1  0  1 -1  1  0  0  0  0
 -2  0  2 -1  0  1  0  0  0
 -2  0  3 -2  0  0  1  0  0
  0  0 -2  1  0  0  0  1  0
 -1  0  2 -2  0  0  0  0  1

5 height 1 Hilbert basis elements:
 2 0 1 0 1 2 1 2 0
 1 0 2 2 1 0 0 2 1
 1 1 1 1 1 1 1 1 1
 1 2 0 0 1 2 2 0 1
 0 2 1 2 1 0 1 0 2
```

The 5 elements of the Hilbert basis represent the magic squares

2	0	1
0	1	2
1	2	0

1	0	2
2	1	0
0	2	1

1	1	1
1	1	1
1	1	1

1	2	0
0	1	2
2	0	1

0	2	1
2	1	0
1	0	2

All other solutions are linear combinations of these squares with nonnegative integer coefficients.

The next question one may rise is: Given a constant $\mathscr{M}$, how many magic square are there with magic constant $\mathscr{M}$? All generators have magic constant 3, so there are no magic squares if $\mathscr{M} \neq 3k$. If $\mathscr{M} = 3k$, then the answer (in this particular case) is given by the Hilbert polynomial

$$P(k) = 1 + 2k + 2k^2 .$$

Note that the nine inequalities $x_i \geq 0$ shrink to four support hyperplanes of the cone defined by the inequalities and the equations.

6.2.3 Type 6, `congruences`

We change our definition of magic square by requiring that the entries in the 4 corners are all even. Then we have to augment the input file as follows (`3x3magiceven.in`):

```
7                                4
9                                10
1 1 1 -1 -1 -1  0  0  0          1 0 0 0 0 0 0 0 0 2
1 1 1  0  0  0 -1 -1 -1          0 0 1 0 0 0 0 0 0 2
0 1 1 -1  0  0 -1  0  0          0 0 0 0 0 0 1 0 0 2
1 0 1  0 -1  0  0 -1  0          0 0 0 0 0 0 0 0 1 2
1 1 0  0  0 -1  0  0 -1          congruences
0 1 1  0 -1  0  0  0 -1
1 1 0  0 -1  0 -1  0  0
equations
```

The output changes accordingly:

```
9 Hilbert basis elements
4 extreme rays
4 support hyperplanes

rank = 3
index = 4
original monoid is not integrally closed

extreme rays are not homogeneous

******************************************************************

9 Hilbert basis elements:          4 support hyperplanes:
 2 4 0 0 2 4 4 0 2                   1  0  1  0 -1  0  0  0  0
 0 4 2 4 2 0 2 0 4                  -1  0  1  0  1  0  0  0  0
 2 2 2 2 2 2 2 2 2                  -1  0 -1  0  3  0  0  0  0
 4 0 2 0 2 4 2 4 0                   1  0 -1  0  1  0  0  0  0
 2 0 4 4 2 0 0 4 2
 2 5 2 3 3 3 4 1 4                 6 equations:
 4 3 2 1 3 5 4 3 2                  -2  1  4 -3  0  0  0  0  0
 2 3 4 5 3 1 2 3 4                  -1  0  1 -1  1  0  0  0  0
 4 1 4 3 3 3 2 5 2                  -2  0  2 -1  0  1  0  0  0
                                    -2  0  3 -2  0  0  1  0  0
4 extreme rays:                      0  0 -2  1  0  0  0  1  0
 4 0 2 0 2 4 2 4 0                  -1  0  2 -2  0  0  0  0  1
 2 4 0 0 2 4 4 0 2
```

```
 0 4 2 4 2 0 2 0 4                2 congruences:
 2 0 4 4 2 0 0 4 2                 0 0 1 0 0 0 0 0 0 2
                                   1 0 0 0 0 0 0 0 0 2
```

As you can see, the equations make two of the input congruences superfluous: it is enough to require the two corners in the first row to be even. The first congruence is to be read as $x_1 \equiv 0 \mod 2$, the second as $x_3 \equiv 0 \mod 2$.

6.3 Relations

6.3.1 Type 10, `lattice_ideal`

As an example, we consider the binomial ideal generated by

$$X_1^2X_2 - X_4X_5X_6,\ X_1X_4^2 - X_3X_5X_6,\ X_1X_2X_3 - X_5^2X_6.$$

We want to find an embedding of the toric ring it defines.

The input ideal `lattice_ideal.in` is

```
3
6
2 1  0 -1 -1  -1
1 0 -1  2 -1  -1
1 1  1  0 -2  -1
lattice_ideal
```

It yields the output

```
6 original generators
9 Hilbert basis elements
9 height 1 Hilbert basis elements
5 extreme rays
5 support hyperplanes

rank = 3 (maximal)
index = 1
original monoid is not integrally closed

extreme rays are homogeneous via the linear form:
1 -1 1

Hilbert basis elements are homogeneous

multiplicity = 10

h-vector:
1 6 3
```

```
Hilbert polynomial:
1/1 3/1 5/1

*************************************************************************

6 original generators:
 0 1 2
 3 2 0
 0 0 1
 1 1 1
 1 0 0
 1 3 3

9 Hilbert basis elements:
 1 0 0
 0 0 1
 2 1 0
 1 1 1
 0 1 2
 3 2 0
 2 2 1
 1 2 2
 1 3 3

5 extreme rays:
 0 1 2
 3 2 0
 0 0 1
 1 0 0
 1 3 3

5 support hyperplanes:
  1  0  0
  0  1  0
  0  0  1
  6 -9  7
  3 -2  1

9 height 1 Hilbert basis elements:
 1 0 0
 0 0 1
 2 1 0
 1 1 1
 0 1 2
 3 2 0
 2 2 1
 1 2 2
 1 3 3
```

The 6 original generators correspond to the indeterminates $X_1, \ldots, X_6$ in the binomial equations. They represent an embedding of the affine monoid defined by the binomial equations.

7 Optional output files

When one of the options `-f` or `-a` is activated, Normaliz writes additional output files whose names are of type `<projectname>.<type>`. The format of the files (with the exception of `inv`) is completely analogous to that of the input file, except that there is usually no last line denoting the type. The main purpose of these files is to give the user easy access to the results of the Normaliz run and to provide additional information not contained in the standard output file.

The following files may be written, provided certain conditions are satisfied and the informa-

tion that should go into them is available (we denote the files simply by their types):

`gen` contains the Hilbert basis.

`cst` contains the constraints defining the cone and the lattice in the same format as they would appear in the input: matrices of types 4,5,6 following each other. Each matrix is concluded by the integer denoting its type. Empty matrices are indicated by 0 as the number of rows. Therefore there will always be 3 matrices.

Using this file as input for Normaliz will reproduce the Hilbert basis and all the other data computed.

`inv` contains all the information from the file `out` that is not contained in any of the other files.

`typ` This is the product of the matrices corresponding to `egn` and `esp`. In other words, the linear forms representing the support hyperplanes of the cone C are evaluated on the Hilbert basis. The resulting matrix, with the generators corresponding to the rows and the support hyperplanes corresponding to the columns, is written to this file.

The suffix `typ` is motivated by the fact that the matrix in this file depends only on the isomorphism type of monoid generated by the Hilbert basis (up to row and column permutations). In the language of [2] it contains the *standard embedding*.

The 4 files above are produced with the option `-f`. If `-a` is activated, then the following files are written additionally:

`ext` contains the extreme rays of the cone.

`egn,esp` These contain the Hilbert basis and support hyperplanes in the coordinates with respect to a basis of $\mathbb{E}$.

`tgn, tri` These files together describe the triangulation computed by Normaliz. (The computation types `-N` and `-d` do not compute a triangulation.)

The file `tgn` contains a matrix of vectors (in the coordinates of $\mathbb{A}$) spanning the simplicial cones in the triangulation.

The file `tri` lists the simplicial subcones as follows: The first line contains the number of simplicial cones in the triangulation, and the next line contains the number $m+1$ where $m = \operatorname{rank}\mathbb{E}$. Each of the following lines specifies a simplicial cone Δ: the first m numbers are the indices (with respect to the order in the file `tgn`) of those generators that span Δ, and the last entry is the multiplicity of Δ in $\mathbb{E}$, i. e. the absolute value of the determinant of the matrix of the spanning vectors (as elements of $\mathbb{E}$).

`ht1` contains the height 1 elements of the Hilbert basis in the homogeneous case.

The file `3x3magiceven.in` has been processed with the option `-a` activated. We recommend you to inspect all the output files in the subdirectory `example` of the distribution.

8 Performance and parallelization

The executables of Normaliz have been compiled for parallelization on shared memory systems with OpenMP. Parallelization reduces the "real" time of the computations considerably,

even on relatively small systems. However, one should not underestimate the administrational overhead involved.

- It is not a good idea to use parallelization for very small problems.
- On multi-user systems with many processors it may be wise to limit the number of threads for Normaliz somewhat below the maximum number of cores.

The number of parallel threads can be limited by the Normaliz option `-x` (see Section 4.3) or by the commands

```
export OMP_NUM_THREADS=<T>      (Linux/Mac)
```

or

```
set OMP_NUM_THREADS=<T>      (Windows)
```

where `<T>` stands for the maximum number of threads accessible to Normaliz. We use

```
export OMP_NUM_THREADS=16
```

on a multi-user system system with 24 cores.

Limiting the number of threads to 1 forces a strictly serial execution of Normaliz.

First we compare the performance of Normaliz on several processor configurations.

	mode	i5 M520	i7	Xeon	Xeon
cores/threads		2 cor, 4 thr	4 cor, 8 thr	1 cor, 1 thr	24 cor, 16 thr
`medium`	`-h`	15	2.5	9.2	3.9
`A443`	`-h`	–	340	997	495
`A443`	`-N`	6	0.7	2.8	0.6
`A543`	`-N`	120	17.5	96	10
`A553`	`-N`	–	3061	49560	3467

The computation times in the i5 system are based on the Windows version, the others on the Linux version of `norm64`. Because of lack of memory not all examples could be run on all systems.

The small difference in computation times for `A443 -h` in the last two columns results from the still missing parallelization of the step in the algorithm that is the most time consuming for this example: the combinatorial evaluation of the very large shelling. The shelling must be evaluated in strict order. Therefore naive parallelization is not possible, but we have envisaged a solution for this problem for the next version.

Next we compare the options `-s`, `-S`, `-v` and `-V` (computation times measured on the Xeon system with 1 thread).

	-s	-S	-v	-V
A443	2.7	6.8	1860	392
A543	86	144	–	–
A553	37645	10047	–	–
semigraphoid5	1880	648	–	–

In comparing the options -v and -V one must bear in mind that the difference in time arises only from the computation of the triangulation. The time for the computation of the multiplicity is the same for both options (approximately 300 sec for A443).

Finally we compare the primal and the dual algorithm on several examples (computation times measured on the Xeon system with 16 threads).

	-n	-d
dual	0.03	0.03
cut	0.2	0.6
small	3	520
rafad	25	∞
4x4	0.03	0.02
6x6	∞	10538

As a rule of thumb, one should use -d if the number of extreme rays is at least one magnitude larger than that of support hyperplanes. Therefore a previous run with -s (or -S) may help in choosing the right approach.

The example small is discussed extensively in [5]. The time for -n is based on the input file in the example directory whereas the time for -d is based on an input file containing the support hyperplanes.

9 Distribution and Installation

In order to install Normaliz you should first download the basic package containing the documentation, examples, source code, jNormaliz and the packages for Singular and Macaulay 2. Then unzip the downloaded file Normaliz2.5.zip in a directory of your choice. (Any other downloaded zip file for Normaliz should be unzipped in this directory, too.)

This process will create a directory Normaliz2.5 (called Normaliz directory) and several subdirectories in Normaliz2.5. The names of the subdirectories created are self-explanatory. Nevertheless we give an overview:

- In the main directory Normaliz2.5 you should find jNormaliz.jar, Copying and subdirectories.
- In the subdirectory source contains the source files and a Makefile for compilation with GCC.
- Subdirectory doc contains the file you are reading and further documentation.

- In the subdirectory `example` are the input and output files for some examples It contains all input files of examples of this documentation, except the toy examples of Section 3. Some very large output files are contained in an extra zip file accessible from the Normaliz home page.
- The subdirectory `singular` contains the SINGULAR library `normaliz.lib` and a PDF file with documentation.
- The subdirectory `macaulay2` contains the MACAULAY2 package `Normaliz.m2`.
- The subdirectory `lib` contains libraries for jNormaliz.

We provide executables for Windows, Linux (each in a 32 bit and a 64 bit version) and Mac. Download the archive file corresponding to your system `Normaliz2.5<systemname>.zip` and unzip it. This process will store the two executables in the directory `Normaliz2.5`. In case you want to run Normaliz from the command line or use it from other systems, you may have to copy the executables to a directory in the search path for executables.

10 Compilation

10.1 GCC

Produce the executables by calling `make` in the subdirectory `source`. You may have to transport the executables to a directory in your search path. **jNormaliz expects them in its own directory.**

Note that `normbig` needs GMP (including the C++ wrapper). Therefore you must install it first.

The current versions of GCC are compatible with our use of OpenMP. **Exceptions:**

1. One can compile Windows executables with the Cygwin port of GCC. Unfortunately it is not compatible to OpenMP.
2. Mac versions of GCC older than 4.5 have a bug that makes it impossible to use OpenMP.

In both cases, or if you want to avoid parallelization, call `make OPENMP=no`.

10.2 Visual C++

The Windows executables provided by us have been compiled with Visual C++ (as contained in Visual Studio 9).

If you want to compile Normaliz yourself by Visual C++, please unzip the corresponding zip file on the Normaliz home page. This will create a subdirectory `Visual C++` of the Normaliz directory. This directory contains the predefined projects. For both projects (`norm64` and `normbig` we have provided

1. two configurations: `Release` (with OpenMP) and `ReleaseSerial` (without OpenMP), and

2. two platforms, `Win32` and `x64`.

Instead of GMP we use the MPIR library for the Windows versions of `normbig`. For convenience, the MPIR files have been included in the distribution (in the subdirectory `MPIR` of `Visual C++`). Please

- copy the library files for Win32 into the `lib` subdirectory of the Visual C++ compiler,
- the library files for x64 to the subdirectory `amd64` (or `x64`) of `lib`, and
- the two header files to the `include` subdirectory of the compiler.

After the compilation with Visual C++ you must copy the executables to the directories where they are expected (the Normaliz directory or a directory in the search path).

The source files for Visual C++ are identical to those for GCC.

11 Changes relative to version 2.0

Changes in version 2.1:

User control, input and output:

1. The command line option `-i` forces Normaliz to ignore a potentially existing setup file. This is useful if an external program wants to keep complete control (see Section 4). In case the setup file does not exist, `-i` keeps Normaliz from issuing a warning message. (Obsolete)
2. In addition to the choice of the type via a single digit in the last line of the input file, the type can now be specified by a keyword (see Section 3).
3. In the homogeneous case Normaliz also lists the "height 1" elements in the Hilbert basis (and writes them to a file with suffix `ht1` if requested); see Sections 5 and 6.
4. The structure of the file with suffix `inv` (used for the communication with computer algebra systems) has been changed from a SINGULAR command to a neutral format.

Algorithms:

1. In types 4 and 5 in which the input is given as a system of homogeneous linear inequalities or equations resp., it is often (but by no means always) better to use (a variant of) Pottier's algorithm. The user can choose this algorithm by the command line option `-d` representing "dual" (see Section 8).

Access from computer algebra systems:

1. a package for MACAULAY2.
2. library for SINGULAR extended by functions for torus invariants and valuation rings.

Changes in version 2.2:

User control, input and output:

1. New command line option `-e` to activate test for arithmetic errors.
2. New command line option `-m` to save memory at the expense of computation time. This option replaces "optimize for speed" in version 2.1.

3. New command line option `-?` to print a small help text.
4. Name of setup file changed from `setup.txt` to `normaliz.cfg`.
5. It is now possible to give the input file with the ending ".in" (but not recommended).
6. Option "Abort by user" removed. The program exits if an error is detected.
7. Renamed "Run mode" to "Computation type" for clearer distinction to the (run) mode.
8. Renamed "Testing number" to "Overflow Test Modulus", "Lifting constant" to "Lifting bound" and "Use control data" to "Verbose".
9. File extension `.hom` changed to `.ht1`.
10. In type `2=polytope` the vectors in the file `.ext` are given as extreme rays of the cone over the polytope (vertices of the polytope in the previous version).

Changes in version 2.5:

User control, input and output:

1. Introduction of jNormaliz.
2. Setup file abolished. Option `-i` therefore obsolete.
3. Option `-m` obsolete because of improved algorithm.
4. New input types 6 and 10. Moreover, combination of 4, 5 and 6 allowed.
5. Output file reorganized. Equations and congruences added.
6. File `sup` replaced by `cst` containing a full system of constraints.
7. File `tri` supplemented by file `tgn` (necessary since the reference to the input file is not always possible).

Algorithms and implementation:

1. Several details improved. Memory usage reduced.
2. Shelling algorithm improved considerably.
3. Algorithms for large examples added.
4. Parallelization for shared memory systems.
5. `norm32` abolished.
6. More general notion of homogeneity.

Access from computer algebra systems:

1. MACAULAY2 package restructured by Gesa Kämpf.
2. MACAULAY2 package and SINGULAR library adapted.

12 Copyright

published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

It is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with the program. If not, see http://www.gnu.org/licenses/.

Please refer to Normaliz in any publication for which it has been used:

W. Bruns, B. Ichim and C. Söger: Normaliz. Algorithms for rational cones and affine monoids. Available from http://www.math.uos.de/normaliz.

References

[1] V. Almendra and B. Ichim. *jNormaliz 1.0.*.

[2] W.Bruns and J. Gubeladze. *Polytopes, rings, and K-theory*. Springer 2009.

[3] W.Bruns, R. Hemmecke, B. Ichim, M. Köppe and C. Söger. *Challenging computations of Hilbert bases of cones associated with algebraic statistics* . Exp. Math., to appear.

[4] W.Bruns and J. Herzog. *Cohen-Macaulay rings*. Rev. ed. Cambridge University Press 1998.

[5] W.Bruns and B. Ichim. *Normaliz: algorithms for rational cones and affine monoids.* J. Algebra **324** (2010) 1098–1113.

[6] W.Bruns and R. Koch. *Computing the integral closure of an affine semigroup*. Univ. Iagell. Acta Math. **39** (2001), 59–70.

[7] L. Pottier. *The Euclide algorithm in dimension n*. Research report, ISSAC 96, ACM Press 1996.