

THE PACKAGE NORMALIZ FOR NORMALIZ 2.1

WINFRIED BRUNS, GESA KÄMPF

The package `Normaliz` provides an interface for the use of `normaliz 2.1` within `Macaulay 2`. The structure is similar to the one of the corresponding `Singular` library `Normaliz.lib`. The exchange of data is via files, the only possibility offered by `normaliz` in its present version. In addition to the top level functions that aim at objects of type `ideal` or `ring`, several other auxiliary functions allow the user to apply `normaliz` to data of type `Matrix`. Therefore `Macaulay 2` can be used as a comfortable environment for the work with `normaliz`.

The package is loaded by

```
i1 : loadPackage("Normaliz");
```

(Of course it must be in a directory where `Macaulay 2` looks for packages.)

In order to save space some of the examples below are typeset in two or three columns.

Note: The order in which the vectors or monomials in the examples are computed, depends sometimes on random parameters. Therefore your own computations may produce them in a different order.

1. A SIMPLE EXAMPLE

The typical application of `Normaliz` in commutative algebra is the computation of the integral closures of monomial ideals and monomial subalgebras of polynomial ring. In the following example we consider the ideal $I = (x^2, y^2, z^3)$ in the polynomial ring $S = K[x, y, z]$ and want to compute the integral closure of the ideal and the integral closure of the Rees algebra $\mathcal{R}(I)$ of I in the ring $R = S[t]$ of which $\mathcal{R}(I)$ is naturally a subalgebra.

`Macaulay 2`, version 1.1.99

with packages: `Classic`, `Core`, `Elimination`, `IntegralClosure`, `LLLBases`,
`Parsing`, `PrimaryDecomposition`, `SchurRings`, `TangentCone`

```
i1 : loadPackage "Normaliz"
o1 = Normaliz
o1 : Package
i2 : setNmzExecPath("./")
i3 : R=QQ[x,y,z,t];
i4 : I=ideal(x^2,y^2,z^3);
o4 : Ideal of R
i5 : intclMonIdeal(I)
o5 = {ideal (y2, x*y, x2, z3, y*z2, x*z2),
      ideal (z, y t, x*y*t, x t, z t, y, x, y*z t, x*z t)}
o5 : List
```

Date: 6 February 2009.

We load the package `Normaliz` as usual. By `setNmzExecPath("./")` we tell Macaulay 2 where to find the executable of `Normaliz`. In our case Macaulay 2 is run on a Unix system, and we run Macaulay 2 in the directory where the `Normaliz` package has been unzipped. This command is superfluous if the executable for `Normaliz` is in your search path.

We define R and I as indicated above, and compute what we wanted by `intclMonIdeal(I)`. The output is a list of two monomial ideals, of which the integral closure of I is the first. The second ideal is to be considered as the list of monomials generating the integral closure of the Rees algebra. We continue by retrieving the numerical information computed by `Normaliz`. The last two last lines tell us that our ideal I is primary to the maximal ideal of S with multiplicity 12. The other lines give information about the cone generated by (the exponent vectors of) the monomials in the Rees algebra. (Think about their ring theoretic interpretation!)

```
i6 : showNumInvs();           number support hyperplanes : 5
hilbert basis elements : 9   homogeneous : false
number extreme rays : 6     primary : true
rank : 4                    ideal multiplicity : 12
index : 1
```

Let us clean up:

```
i7 : rmNmzFiles();
```

This removes the files in which Macaulay 2 and `Normaliz` have exchanged data. In this example we have used a filename created automatically by the library. You can also define the filename yourself.

2. PATHS AND FILES

If `normaliz` is not in the search path for executables, then its path must be made known to Macaulay 2. Furthermore one can set the file name to be used for the exchange of data, set the path to the directory where `normaliz` and Macaulay 2 exchange data, choose the executable (if `normbig` is needed), and remove the files created.

The path names need to be defined only once since they can be written to the hard disk and retrieved from there in subsequent sessions.

The package defines the following functions for these purposes:

- `setNmzExecPath(s)`

The function sets the path to the executable for `normaliz`. This is absolutely necessary if it is not in the search path. To retrieve the value stored in the global variable call `getNmzExecPath()`.

```
i2 : setNmzExecPath("d:/Normaliz2.1Windows"); -- Windows
i3 : getNmzExecPath() -- returns the global variable holding the path name
o3 = d:/normaliz/bin/ -- the last / is added (if necessary)
i2 : setNmzExecPath($HOME/Normaliz2.1Linux"); -- Unix
```

The path given in the example under Windows is the necessary path if the `Normaliz` package has been unzipped in the root directory of drive `d:`.

- `setNmzVersion(s)`

The function chooses the version of the executable for `normaliz`. The default is `norm64`, and nothing needs to be done if it is sufficient. To retrieve the value stored in the global variable call `getNmzVersion()`.

```

i4 : setNmzVersion("normbig"); -- choose normbig
i5 : getNmzVersion()      -- returns the variable holding the version name
o5 = normbig
i6 : setNmzVersion("norm32"); -- now it is norm32

```

- `setNmzDataPath(s)`

The function sets the directory for the exchange of data. By default it is the current directory (or the home directory of Macaulay 2). If this choice is o.k., nothing needs to be done. To retrieve the value stored in the global variable call `getNmzDataPath()`.

```

i7 : setNmzDataPath("d:/Normaliz2.1Windows/example"); -- Windows
i8 : getNmzDataPath()      -- returns the variable holding the path name
o8 = d:/normaliz/example/
i7 : setNmzDataPath("../MyFiles/normaliz"); -- Unix

```

- `writeNmzPaths()`

The function writes the path names into two files in the current directory. If one of the names has not been defined, the corresponding file is written, but contains nothing.

```

i9 : writeNmzPaths();          i11 : get("nmzM2Data.path")
i10 : get("nmzM2Exec.path")    o11 = d:/normaliz/example/;
o10 = d:/normaliz/bin/
i11 : get("nmzM2Data.path")
o11 = d:/normaliz/example/;

```

- `startNmz()`

This function reads the files written by `writeNmzPaths()`, retrieves the path names, and types them on the standard output (as far as they have been set). Thus, once the path names have been stored, a `normaliz` session can simply be opened by this function.

```

i1 : startNmz();              i3 : writeNmzPaths();
nmzExecPath is d:/normaliz/bin/  i4 : startNmz();
nmzDataPath is d:/normaliz/example/  nmzExecPath is d:/normaliz/bin/
i2 : setNmzDataPath("");      nmzDataPath not set

```

- `setNmzFilename(s)`

The function sets the filename for the exchange of data. By default, the package creates a filename `nmzM2_pid` where `pid` is the process identification of the current Macaulay 2 process. If this choice is o.k., nothing needs to be done. Call `getNmzFilename()` to obtain the current filename.

```

i5 : setNmzFilename("VeryInteresting");  i7 : setNmzFile()
i6 : getNmzFilename() -- the file name  o7 = d:/normaliz/
o6 = VeryInteresting                example/VeryInteresting

```

- `rmNmzFiles()`

This function removes the files created for and by `normaliz`, using the last filename created. These files are *not* removed automatically. See Section 1 for an example.

3. INTEGRAL CLOSURES OF MONOMIAL IDEALS AND TORIC RINGS

There are 4 functions, corresponding to the modes 0,1,2,3 of `Normaliz`. In all cases the parameter of the function is an ideal. Its elements need not be monomials: the exponent vectors of the leading monomials form the input of `Normaliz`. Note: the functions return nothing if one

of the options `supp`, `triang`, or `hvect` has been activated. However, in this case some numerical invariants are computed, and some other data may be contained in files that you can read into Macaulay 2 (see Section 8).

- `normalToricRing(I)`

Computes the normalization of the toric ring generated by the leading monomials of the elements of I . The function returns an ideal listing the generators of the normalization.

A mathematical remark: the toric ring (and the other rings computed) depends on the list of monomials given, and not only on the ideal they generate!

```
i1 : loadPackage "Normaliz";
i2 : R=ZZ/37[x,y,t];
i3 : I=ideal(x^3,x^2*y,y^3);
o3 : Ideal of R
i4 : setNmzExecPath("d:/Normaliz2.1Windows");
i5 : normalToricRing(I)
o5 = ideal (x^3, x^2 y, y^3, x^2 y)
o5 : Ideal of R
```

- `intclToricRing(I)`

Computes the integral closure of the toric ring generated by the leading monomials of the elements of I in the ring of I . The function returns an ideal listing the generators of the integral closure.

```
i6 : intclToricRing(I)
o6 = ideal (x, y)
o6 : Ideal of R
```

- `ehrhartRing(I)`

The exponent vectors of the leading monomials of the elements of I are considered as generators of a lattice polytope. The function returns a list of ideals:

(i) If the last ring variable is not used by the monomials, it is treated as the auxiliary variable of the Ehrhart ring. The function returns a list of two ideals, the first containing the monomials representing the lattice points of the polytope, the second containing the generators of the Ehrhart ring.

(ii) If the last ring variable is used by the monomials, the list contains only one ideal, namely the monomials representing the lattice points of the polytope.

```
i7 : ehrhartRing(I)
o7 = {ideal (x^3, x^2 y, y^3, x^2 y), ideal (x t, x y t, y t, x^2 y t)}
o7 : List
i8 : J=I+ideal(x*y^2*t^7)
i9 : ehrhartRing(J)
o9 = {ideal (x^3, x^2 y, y^3, x^2 y t, x y^2 t, x y t^2, x y^2 t, x^2 y t, x^2 y t, x^2 y t,
-----
x^2 y t, x^2 y t, x^2 y t, x^2 y t)}
o9 : List
```

- `intclMonIdeal(I)`

The exponent vectors of the leading monomials of the elements of I are considered as generators of a monomial ideal whose Rees algebra is computed. The function returns a list of ideals:

(i) If the last ring variable is not used by the monomials, it is treated as the auxiliary variable of the Rees algebra. The function returns a list of two ideals, the first containing the monomials generating the integral closure of the monomial ideal, the second containing the generators of the Rees algebra.

(ii) If the last ring variable is used by the monomials, the list contains only one ideal, namely the monomials generating the integral closure of the ideal.

```
i10 : intclMonIdeal(I)
      3 2 3 2          3 2 3 2
o10 = {ideal (x , x y, y , x*y ), ideal (x, y, x t, x y*t, y t, x*y t)}
o10 : List
i11 : intclMonIdeal(J)
      3 2 3 2
o11 = {ideal (x , x y, y , x*y )}
o11 : List
```

4. TORUS INVARIANTS, VALUATION RINGS AND IDEALS

Let $T = (K^*)^r$ be the r -dimensional torus acting on the polynomial ring $R = K[X_1, \dots, X_n]$ diagonally. Such an action can be described as follows: there are integers a_{ij} , $i = 1, \dots, r$, $j = 1, \dots, n$, such that $(\lambda_1, \dots, \lambda_r) \in T$ acts by the substitution

$$X_j \mapsto \lambda_1^{a_{1j}} \dots \lambda_r^{a_{rj}} X_j, \quad j = 1, \dots, n.$$

In order to compute the ring of invariants R^T one must specify the matrix (a_{ij}) .

- `torusInvariants(T,R)`

The function returns an ideal representing the list of monomials generating R^T where in the function T stands for the matrix (a_{ij}) .

```
i1 : R=QQ[x,y,z,w];
i2 : T=matrix({{-1,-1,2,0},{1,1,-2,-1}});
      2 4
o2 : Matrix ZZ <--- ZZ
i3 : torusInvariants(T,R)
      2 2
o3 = ideal (x z, x*y*z, y z)
o3 : Ideal of R
```

It is of course possible that $R^T = K$. At present, Normaliz cannot deal with the zero cone and will issue the (wrong) error message that the cone is not pointed. The function also gives an error message if the matrix T has the wrong number of columns.

A discrete monomial valuation v on R (as above) is determined by the values $v(X_j)$ of the indeterminates. The following function computes the subalgebra $S = \{f \in R : v_i(f) \geq 0, i = 1, \dots, r\}$ for several such valuations v_i , $i = 1, \dots, r$. It needs the matrix $V = (v_i(X_j))$ and the polynomial ring R as its input.

- `valRing(V,R)`

The function returns a monomial ideal, to be considered as the list of monomials generating S .

```
i1 : R=QQ[x,y,z,w];
i2 : V0=matrix({{0,1,2,3},{-1,1,2,1}});
i3 : valRing(V0,R)
      2
o3 = ideal (w, x*w, y, x*y, z, x*z, x z)
o3 : Ideal of R
```

Again it is possible that $S = K$, and then the same (wrong) error message as above will be issued.

One can simultaneously determine the S -submodule $M = \{f \in R : v_i(f) \geq w_i, i = 1, \dots, r\}$ for integers w_1, \dots, w_r . (If $w_i \geq 0$ for all i , M is an ideal of S .) The numbers w_i form the $(n+1)$ th column of the input matrix:

```
i4 : V=matrix(0,1,2,3,4,-1,1,2,1,3);
      2          5
o4 : Matrix ZZ <--- ZZ
i5 : valRingIdeal(V,R)
      2
o5 = {ideal (y, x*y, w, x*w, z, x*z, x z),
      2 2 2 2 2 4 4 2 3
      ideal (z*w, x*z, z, y w, y z, x*y z, y, x*y, y*w, w )}
o5 : List
```

We have just seen an example for

- `valRingIdeal(V,R)`

The function returns two ideals, both to be considered as lists of monomials. The first is the system of monomial generators of S , the second the system of generators of M .

Note: The functions in this section use the modes 4 and 4 of `Normaliz`. For large matrices T or V it could be useful to set the `dual` option. See Section 5.

5. SETTING OPTIONS

The package always uses the options `-f` and `-i` for `Normaliz` (the latter can be deactivated; see below). The options are set as follows:

- `setNmzOption(optname, onoff)`

If `onoff=true` the option is activated, and if `onoff=false` it is deactivated. The function returns `true` if there is an option of name `optname` and returns `false` otherwise.

- `showNmzOptions()`

Prints the activated options on the standard output.

The `Normaliz` options are accessible via the following names:

```
-s: supp          -a: allf
-v: triang        -c: control
-p: hvect         -i: ignore
-n: normal
-h: hilb
-d: dual
```

Note: It makes no sense to activate more than one of the options in the left column. The option `normal` is hardly ever necessary because `Normaliz` uses it automatically (provided the setup file does not say something else—but that is ignored, unless you deactivate `ignore`).

```
i1 : setNmzOption("hilb",true);          o2 = false
o1 = true                                i3 : showNmzOptions();
i2 : setNmzOption("hulb",true)          The following options are set:
setNmzOption: Invalid option hulb      -f -h -i
```

6. RETRIEVING NUMERICAL INVARIANTS

The following functions make the numerical invariants computed by `normaliz` accessible to Macaulay 2 (as far as they are computed and available in the output file of `normaliz`). Note that some of the numerical invariants are also computed with the volume option. While the output file of `normaliz` interprets the numerical invariants according to the mode, such interpretation is not taken care of by the functions below.

- `getNumInvs()`

The function returns a list whose length depends on the invariants available. The order of the elements in the list is always the same. Each list element has two parts. The first is a `String` describing the invariant, the second is the invariant, namely an `Integer` for rank, index, multiplicity, a `Sequence` for the weights, the h-vector and the Hilbert polynomial and a `Boolean` for homogeneity and primary (to the maximal ideal).

```
i1 : setNmzOption("hilb",true);
i2 : intclMonIdeal(I);
i3 : getNumInvs()
o3 = {{hilbert basis elements, 6}, {number extreme rays, 4}, {rank, 3}, {index,
-----
1}, {number support hyperplanes, 4}, {homogeneous, true}, {height 1
-----
elements, 6}, {homogeneous weights, (1, 1, -2)}, {multiplicity, 4},
-----
{h-vector, (1, 3, 0)}, {hilbert polynomial, (2, 6, 4)}, {primary, true},
-----
{ideal multiplicity, 9}}
o3 : List
```

Note: Only the data computed by `normaliz` are read. There are no "blank" entries in the result of `getNumInvs()`.

- `showNumInvs()`

This function types the numerical invariants on the standard output, but returns nothing. (It calls `getNumInvs()`.)

```
i4 : showNumInvs()
hilbert basis elements : 6
number extreme rays : 4
rank : 3
index : 1
number support hyperplanes : 4
homogeneous : true
height 1 elements : 6
homogeneous weights : (1,1,-2)
multiplicity : 4
h-vector : (1,3,0)
hilbert polynomial : (2,6,4)
primary : true
ideal multiplicity : 9
```

- `exportNumInvs()`

This function exports the data read by `getNumInvs()` into numerical Macaulay 2 data that can be accessed directly. For each invariant a variable of type `Integer`, `Sequence` or `Boolean` is created whose name is the first entry of each list element shown above, prefixed by `nmz`. If the `Print` option is set to `true`, the variables are created and printed to the standard output. The default value of the `Print` option is `false`.

```
i5 : exportNumInvs()
i6 : nmzHilbertBasisElements
o6 = 6
i7 : nmzHomogeneousWeights
nmzNumberSupportHyperplanes=4
nmzHomogeneous=true
nmzHeight1Elements=6
nmzHomogeneousWeights=(1,1,-2)
```

```

o7 = (1, 1, -2)          nmzMultiplicity=4
o7 : Sequence          nmzHVector=(1,3,0)
i8: exportNumInvs(Print=>true) nmzHilbertPolynomial=(2,6,4)
nmzHilbertBasisElements=6 nmzPrimary=true
nmzNumberExtremeRays=4 nmzIdealMultiplicity=9
nmzRank=3              i9 : nmzHilbertBasisElements
nmzIndex=1            o9 = 6

```

7. RUNNING `normaliz` ON DATA OF TYPE `Matrix`

There are functions to write and read files created for and by `normaliz`. Note that all functions in Section 3 as well as the function `normaliz` below write and read their data automatically to and from the hard disk so that `writeNmzData` will hardly ever be used explicitly.

- `writeNmzData(sgr, nmzMode)`

Creates an input file for `normaliz`. The rows of `sgr` are considered as the generators of the semigroup. The parameter `nmzMode` sets the mode.

```

i1 : sgr=matrix(1,2,3,4,5,6,7,8,10);
i2 : writeNmzData(sgr,1);
i3 : get(setNmzFile()|.in")
o3 = 3
      3
      1 2 3
      4 5 6
      7 8 10
      1

```

- `normaliz(sgr, nmzMode)`

The function applies `normaliz` to the parameter `sgr` in the mode set by `nmzMode`. The function returns the `Matrix` defined by the file with suffix `gen`.

```

i4 : normaliz(sgr,0)
o4 = | 7 8 10 |
      | 1 2 3 |
      | 2 3 4 |
      | 3 4 5 |
      | 4 5 6 |
           5      3
o4 : Matrix ZZ <--- ZZ

```

- `readNmzData(suffix)`

Reads an output file of `normaliz` containing an integer matrix and returns it as a `Matrix`. For example, this function is useful if one wants to inspect the support hyperplanes. The filename is created from the current filename in use and the suffix given to the function.

```

i5 : readNmzData("sup")
o5 = | -2 -2 3 |
      | -4 11 -6 |
      | 1 -2 1 |
           3      3
o5 : Matrix ZZ <--- ZZ

```

8. MONOMIALS TO/FROM Matrix

The transformation of data between an Ideal and a Matrix is carried out by the following functions:

- `mons2intmat(I)`

Returns the Matrix whose rows represent the leading exponents of the elements of I. The length of each row is the numbers of variables of the ring of I.

```
i1 : m=mons2intmat(J)
o1 = | 3 0 0 |
      | 2 1 0 |
      | 0 3 0 |
      | 1 2 7 |
      4      3
o1 : Matrix ZZ <--- ZZ
```

- `intmat2mons(m,R)`

The converse operation. The ring whose elements the monomials shall be has to be specified by R.

```
i2 : intmat2mons(m,R)
      3 2 3 2 7
o2 = ideal (x , x y, y , x*y t )
o2 : Ideal of R
```