Sagbi combinatorics of maximal minors

An experimental approach

Winfried Bruns

FB Mathematik/Informatik
Universität Osnabrück
wbruns@uos.de

Genova, October 2022

Preliminary report on a joint project with

Aldo Conca

Normaliz 3.10.0

So far Normaliz could only compute invariants of normal monomial algebras A: the set of exponent vectors of the monomials in A is the intersection of a cone and a lattice. This allows algorithms based on triangulations and Stanley decompositions.

Version 3.7.10 will add functions for arbitrary monomial algebras (corresponding to arbitrary monoids of exponent vectors):

- Hilbert basis (minimal generating subset of the given generators)
- check for normality
- Markov basis (set of generators) and Gröbner basis for the defining binomial ideal
- Hilbert series
- singular locus, nonormal locus
- automorphism groups

Sagbi bases

More fashionable name: *Khovanskii bases*

Let $R = K[X_1, \ldots, X_n]$ be a polynomial ring over the field K, endowed with some monomial (or term) order $<$. Let $A \subset R$ be a K-subalgebra. The initial algebra is the vector space

$$\mathrm{in}(A) = \mathrm{in}_<(A) = \sum_{f \in A} K\,\mathrm{in}_<(f).$$

It is automatically a subalgebra. Introduced by Robbiano–Sweedler and Kapur–Madlener $\sim$ 1989.
General problem: $\mathrm{in}(A)$ ned not be finitely generated, even if A is.
General advantage: $\mathrm{in}(A)$ is generated by monomials.

$B \subset A$ is a Sagbi basis if the monomials $\mathrm{in}(f)$, $f \in B$, generate $\mathrm{in}(A)$.

If $\mathrm{in}(A)$ is finitely generated, then it is a toric deformation of A.

The Grassmannian

Let K be a field, $X = (X_{ij})$ be an $m \times n$, $m \leq n$, matrix of indeterminates and $R = K[X] = K[X_{ij} : i = 1, \ldots, m, j = 1, \ldots n]$. Set

$$\mathcal{M} = \mathcal{M}_{m\times n}\{\delta : \delta \text{ is an } m\text{-minor of } X\}$$

The homogeneous coordinate ring of the Grassmannian $G(m, n)$ is the subalgebra

$$K[\mathcal{M}] = K[\mathcal{M}_{mxn}] \subset R.$$

The best monomial orders on R for the exploration of $K[\mathcal{M}]$ are the diagonal ones: in(δ) is product of the diagonal elements of δ.

Hodge's standard bitableaux theory $\Longrightarrow$ the maximal minors form a Sagbi basis w.r.t. a diagonal order.

For a diagonal order, in(A) has all good properties that one can reasonably want: normal, Gorenstein, Koszul, rational singularities, ... $\Longrightarrow$ the same for $K[\mathcal{M}]$.

Univeersality ?

By a theorem of Bernstein–Sturmfels–Zelevinsky $\mathcal{M}$ is a universal Gröbner basis of the ideal $I_m(X) \subset R$ generated by $\mathcal{M}$. (Now a simple proof by Conca, De Negri and Gorla.) Universal means: for every monomial order on R.

Question: is $\mathcal{M}$ a universal Sagbi basis of $K[\mathcal{M}]$?
In other words, $K[\text{in}(\mathcal{M})] = \text{in}(K[\mathcal{M}])$? for all monomial orders?
(Note: always $K[\text{in}(\mathcal{M})] \subset \text{in}(K[\mathcal{M}])$.

True for $m = 2$. The answer "no" for $m = 3$ was given by Speyer–Sturmfels (2004): already for 3×6 there exist lexicographic orders for which the m-minors are not a Sagbi basis.

Our starting question: are the m-minors a universally revlex Sagbi basis? What can we say about $K[\text{in}(\mathcal{M})]$ and $\text{in}(K[\mathcal{M}])$?

Experimental approach via CoCoA, Singular and Normaliz.

Findings so far

Test for $K[\mathrm{in}(\mathcal{M})] = \mathrm{in}(K[\mathcal{M}])$: equality of Hilbert series
(necessary: equality of multiplicities, faster to test)

true, false Not (yet)known

Universally revlex Sagbi basis: 3×6, 3×7, 3×8.

$K[\mathrm{in}(\mathcal{M})]$ normal
revlex: 3×6, 3×7, 3×8, 3×9
lex: 3×6, 3×7, 3×8, 3×9, 3×10.

$K[\mathrm{in}(\mathcal{M})] = \mathrm{in}(K[\mathcal{M}])$ and not normal: revlex 3×9

$\mathrm{in}(K[\mathcal{M}])$ finitely generated ???

Refined question: compare $\mathcal{R}(\mathrm{in}(\mathcal{M}))$ and $\mathrm{in}(\mathcal{R}(\mathcal{M}))$.

Experimental approaches

Two experimental approaches: fix format, choose lex or revlex and

1. run a C++ program that creates all candidates and checks them through libnormaliz,
2. create many random orders of the variables and run Singular with the help of Normaliz through normaliz.lib.

To check whether $\text{in}(K[\mathcal{M}]) = K[\text{in}(\mathcal{M})]$ or $\text{in}(K[\mathcal{M}]) = K[E]$ for a set $E \supset \text{in}(\mathcal{M})$ we compare Hilbert series:

- $\text{in}(K[\mathcal{M}])$ has the same Hilbert series as the Grassmannian and can easily be computed by Normaliz since $\text{in}(K[\mathcal{M}])$ is a normal monoid algebra for a diagonal order.
- as long as $K[\text{in}(\mathcal{M})]$ or $K[E]$ is normal, its Hilbert series can also be computed quickly. In the nonnormal case a Gröbner basis of a binomial ideal is needed.

Creating all candidates I

It is impossible to run through all monomial orders, lex or revlex: for a 3×6 matrix we have 18! orders of each kind, and even if one takes symmetries into account, there remain too many.

Instead we reverse the process: we select a *matching field*, i.e. a choice of one monomial in each of the minors, and check them for compatibility with lex, revlex or weight orders.

Helpful fact, proved by Sturmfels and Zelevinsky: if the matching field comes from a monomial order, then each row has $m-1$ entries that do not appear in any initial monomial.

For $m = 3$, $n \geq 6$ the choices of the omitted variables, fall into 4 classes up to symmetry:

Creating all candidates II

$$\begin{bmatrix} & 0 & 0 & \dots \\ 0 & & 0 & \dots \\ 0 & 0 & & \dots \end{bmatrix} \qquad \begin{bmatrix} & 0 & 0 & & \dots \\ 0 & & & 0 & \dots \\ 0 & 0 & & & \dots \end{bmatrix}$$

$$\begin{bmatrix} & 0 & 0 & & & \dots \\ 0 & & & 0 & & \dots \\ 0 & & & & 0 & \dots \end{bmatrix} \qquad \begin{bmatrix} & & & & 0 & 0 & \dots \\ & & & 0 & 0 & & \dots \\ 0 & 0 & & & & & \dots \end{bmatrix}$$

A matching field is built iteratively: after the choice of a monomial we check compatibility with lex or revlex orders, resp., and as soon the partial choice is incompatible, we backtrack.

In addition to lex and revlex we have also tried monomial orders defined by a weight vector in general. But in our experiments all weight compatible matching fields turned out lex or revlex compatible.

Tête-a- tête and subduction

For the computation of Sagbi bases one has an algorithm (which need not stop) similar to the Buchberger algorithm for Gröbner bases. Let $\mathcal{F} = \{F_1, \ldots, F_n\}$ be a set of *monic* polynomials in the polynomial ring R with a monomial order.
Two operations are needed (terminology of Robbiano–Sweedler):

- (S-polynomial equivalent): find monomials M_1, M_2 in n new variables such that $\text{in}(M_1(F_1, ..., F_n)) = \text{in}(M_2(F_1, ..., F_n))$: the "virtual initial monomial" of $M_1(\mathcal{F}) - M_2(\mathcal{F})$ cancels. ((M_1, M_2) is a tête-a- tête)
- (reduction equivalent): for a monic polynomial $G \in R$ find a monomial M in n variables such that $\text{in}(G) = \text{in}(M(\mathcal{F}))$ and pass to $G - M(\mathcal{F})$ $\quad$ ($M(\mathcal{F})$ subduces G).

Starting from a system of generators of a subalgebra $A \subset R$, a suitable iteration of tête-a- tête and subduction computes a finite Sagbi basis, provided such exists.

A Hilbert series controlled Sagbi algorithm

Basic assumption: A is a graded subalgebra of a polynomial ring R over a field K, generated by a set G of monic homogeneous polynomials. G is completely subduced, and the Hilbert function $\mathrm{HF}(A,k)$, $k \in \mathbb{N}$, has been computed.

1. Let $B = K[\mathrm{in}(G)]$ and Compute $\mathrm{HF}(B,k)$.
2. If $\mathrm{HF}(B,k) = \mathrm{HF}(A,k)$ for all k, stop and output G.
3. Otherwise find the smallest degree c for which $\mathrm{HF}(B,c) \neq \mathrm{HF}(A,c)$ and the defect $d = \mathrm{HF}(A,c) - \mathrm{HF}(B,c)$.
4. Find the the degree c tête-a-têtes of G and evaluate them on G to obtain a system T of polynomials. Augment G by T.
5. Check whether $\mathrm{in}(G)$ has d new monomials. If so, go to (1).
6. Otherwise apply subduction to G, augment it and go to (5).

Note: The subduction loop (5)–(6) must stop after finitely many iterations with d new initial monomials.

Discussion I

The implementation makes sure that at each step (1) we have a system of generators G of A that is completely subduced: the initial monomials are a minimal system of generators of $B = K[\text{in}(G)]$. Most critical steps:

- The Gröbner basis computation of the binomial defining ideal of $K[B]$, needed always for tête-a-tête and Hilbert function in the nonnormal case.
- Evaluation of monomials M on the system G. For example: degree 11 monomial evaluated on the 84 maxinal minors of a 3×9 matrix. (A degree bound must be set.)

At present we use "broad" evaluation of tête-a-têtes and subduction. Better than go polynomial by polynomial.

Implementation in a Singular script that calls Normaliz for all monomial and binomial computations. Singular is only used for polynomial arithmetic and the monomial order.

Discussion II

For the binomial computations one must note that it is not a Gröbner basis computation starting from a system of generators (the latter would be enough for tête-a-tête). The main problem is to find a system of generators of the defining ideal of a monomial algebra, often called a *Markov basis*. There are three approaches:

- Classical elimination.
- Finding a system of generators for the defining ideal of the group ring and saturate with respect to all variables. Various techniques to speed it up (CoCoA).
- The project-and-lift algorithm of Hemmecke-Malkin in 4ti2 and now reimplemented in Normaliz.

The critical degree c gives a degree bound for tête-a-tête. But only for elimination one can use it. Not yet tried.

Singular and normaliz.lib

For 20 years we had the Singular library normaliz.lib (currently under revision and extension). In 2002 it started in a 5th floor apartment in Camogli without radio, TV or internet.

Main disadvantage: it exchanges files with Normaliz for input and results. By itself not so bad, but Singuar needs much time to read large input files, and Singular cannot keep Normaliz objects alive.

Similar approach: the Macaulay2 package. Help wanted !!

Other interfaces (CoCoA, GAP, PyNormaliz, Sage) use the C++ class library libnormaliz—the better solution.