Convex hull computation in Normaliz

Winfried Bruns

FB Mathematik/Informatik
Universität Osnabrück

wbruns@uos.de

Berlin, December 2016

Based on

W. Bruns, B. Ichim and C. Söger

The power of pyramid decomposition in Normaliz

J. Symb. Comp. 74 (2016), 513–536

Lattice points in a polyhedron

Normaliz computes the set N of lattice points in a rational polyhedron.
Main computation goals:

- Generation Describe N by generators.
- Enumeration Given a grading, count the elements of degree k.

A rational polyhedron is defined by (inhom)

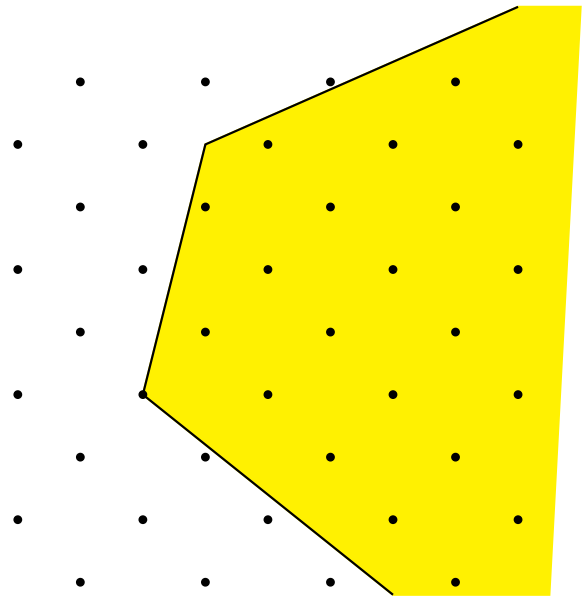
- linear inequalities with coefficients from $\mathbb{Z}$.

An affine lattice is defined by (inhom)

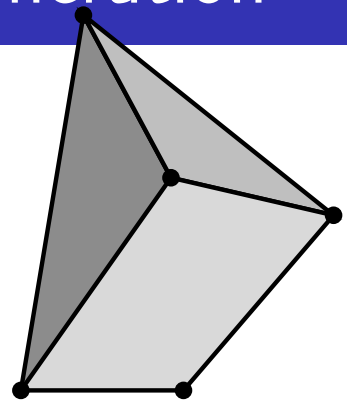
- diophantine linear equations and
- linear congruences.

I.e., Normaliz solves linear diophantine systems.

$\Longrightarrow$ Applications in many fields



Convex hull computation and vertex enumeration



A preparatory task for the lattice point computations of Normaliz can best be visualized in the language of polytopes.

They have two descriptions:

Theorem (Minkowski)

For a bounded set $P \subset \mathbb{R}^d$ the following are equivalent:

1. $P = \text{conv}(x_1, \ldots, x_n)$ *for* $x_1, \ldots, x_n \in \mathbb{R}^d$;
2. $P = \{x \in \mathbb{R}^d : \lambda_i(x) \geq 0\}$ *for linear functions* $\lambda_1, \ldots, \lambda_s$ *on* $\mathbb{R}^d$.

The conversion of the "V(ertex)-representation" (1) into the "H(alf space)-representation" (2) is called convex hull computation and the converse is vertex enumeration. (Equivalent via dualization.)

If chosen minimal, the x_i are the vertices of P, the $H_i = \{x : \lambda_i(x) = 0\}$ are the support hyperplanes, the intersections $F_i = P \cap H_i$ are the facets.

Performance in large computations

according to M. Köppe, Y. Zhou, *New computer-based search strategies for extreme functions of the Gomory–Johnson infinite group problem*, arXiv:1506.00017v3

				Running time (s)					
q	dim	ineq	vert	PPL	Porta	cddlib	lrslib	Panda	Normaliz
...									
11	4	78	18	0.003	0.016	0.031	0.009	23	0.007
13	5	105	40	0.007	0.018	0.11	0.021	4604	0.011
15	6	136	68	0.017	0.037	0.21	0.14		0.017
17	7	171	251	0.14	0.20	1.2	0.71	—	0.047
19	8	210	726	0.91	1.6	5.0	2.3	q	0.16
21	9	253	1661	6.6	13	24	13		0.67
23	10	300	7188	166	558	785	74	1.43	4.9
25	11	351	23214	1854	10048	12129	471	1.47	21
26	12	378	54010				2167	1.51	62
27	12	406	68216					1.32	89
28	13	435	195229					1.31	326
29	13	465	317145	$q = v \cdot d^3/(10^6 \cdot t)$ for Normaliz by WB				1.08	644
30	14	496	576696					0.93	1693
31	14	528	1216944					0.98	3411

Normaliz computation time linear in the output ?

Another case of linearity

Cones (or polytopes) coming from contingency tables

$q = s \cdot d^3/(10^6 \cdot t)$

Input	Dim	Vert	Supp	time	q
$5 \times 4 \times 3$	36	60	29,387	32 s	42.8
$5 \times 5 \times 3$	43	75	306,955	573 s	42.6
$6 \times 4 \times 3$	42	72	153,858	277 s	41.2

Computation times serial

Incremental polytope building

We use an an incremental algorithm that builds a polytope $P \subset \mathbb{R}^d$ of dimension d by successively extending the system of generators $x_1, \ldots, x_n$ and determining the support hyperplanes in this process.

Start: We may assume that $x_1, \ldots, x_{d+1}$ are affinely independent. The computation of the support hyperplanes is then simply the inversion of the matrix with rows $(x_1, 1), \ldots (, x_{d+1}, 1)$. (In principle superfluous.)

Extension: We add $x_{d+2}, \ldots, x_n$ successively: from the support hyperplanes of $P' = \text{conv}(x_1, \ldots, x_{n-1})$ we must compute the support hyperplanes of $P = \text{conv}(P', x_n)$.

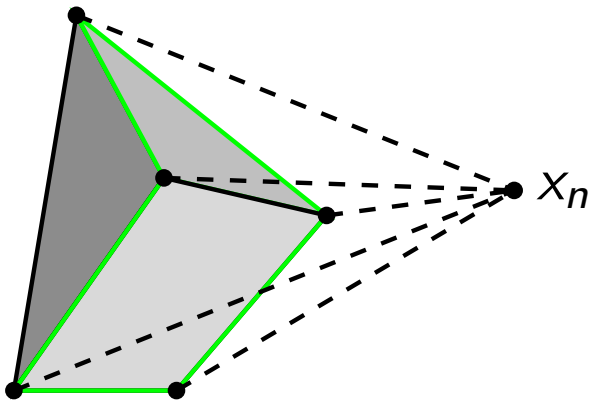
We describe this process geometrically.

Disadvantage of incremental algorithms: Large memory requirements.

The geometry

We determine the boundary V of the part of P' that is visible from x_n and its decomposition into subfacets. Together with x_n these span the new facets of P. The facets of P' that are visible from x_n ($\lambda_i(x_n) < 0$), are discarded.

For a 3-dimensional polytope:



We have to find V: pair visible ($\lambda_i(x_n) < 0$) and invisible ($\lambda_j(x_n) > 0$) facets: Each segment (in this dimension) of V is the intersection of such a pair. Problem: Many pairs.

Fourier–Motzkin elimination: the treacherous double loop

Input: $P' = \text{conv}(x_1, \ldots, x_{n-1}) = \{\lambda_i(x) \geq 0,\ i = 1, \ldots, s'\}$.
Want: $P = \text{conv}(P', x_n) = \{\mu_j(x) \geq 0,\ j = 1, \ldots, s\}$.

Sort the λ_i: $\lambda_i(x_n) < 0$, $i = 1, \ldots, n$, $\lambda_i(x_n) > 0$,
$i = n+1, \ldots, n+p$,, $\lambda_i(x_n) = 0$ else.

$k = 0$.
for $i = 1, \ldots, n$
  for $j = 1, \ldots, p$
    $\mu = \lambda_{j+n}(x_n)\lambda_i - \lambda_i(x_n)\lambda_{j+n}$
    if $P' \cap \{x : \mu(x) = 0\}$ is a subfacet of P'
      then $k = k+1$, $\mu_k = \mu$

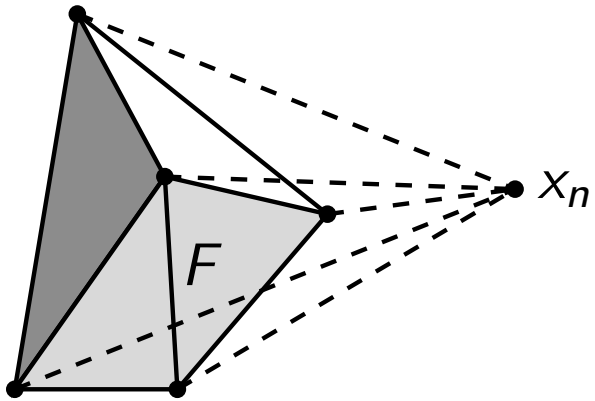
Return $\{\mu_1, \ldots, \mu_k\} \cup \{\lambda_i : i = n+1, \ldots, s'\}$

Very simple, but very treacherous: running time (at least) $\approx np$, and therefore quadratic in s (depending further on d).

The even worse double loop of placing triangulation

A short excursion into triangulations.

Input: $P' = \text{conv}(x_1, \ldots, x_{n-1}) = \{\lambda_i(x) \geq 0,\ i = 1, \ldots, s'\}$, triangulation Δ'
Want: triangulation Δ of $P = \text{conv}(P', x_n)$.



$\Delta = \emptyset$.

for all visible facets F of P
  for all simplices $\sigma \in \Delta'$
    if $\dim(\sigma \cap F) = d - 1$,
      then $\Delta = \Delta \cup \{\text{conv}(x_n, \sigma \cap F)\}$

Return Δ

Again very simple and the test of $\dim(\sigma \cap F) = d - 1$ is very fast. But Δ' can be very large, and one runs quickly out of memory: we must localize the computation of Δ.

How to find the subfacets

Back to convex hulls.

P polytope of dimension d, support hyperplanes $H_1, \dots, H_s$ (defined by $\lambda_1, \dots, \lambda_s$), facets $F_1, \dots, F_s$.

Theorem

Suppose that $i \neq j$ and $|\operatorname{vert}(F_i \cap F_j)| \geq d-1$. Then the following are equivalent:

- *$F_i \cap F_j$ is a subfacet,*
- *$\dim(F_i \cap F_j) = d-2$ (definition, but has algorithmic meaning)*
- *if $F_i \cap F_j \subset F_k$ for a nonsimplex F_k, then $k = i$ or $k = j$.*

Moreover, if F_i or F_j is a simplex (contains exactly d vertices), then $F_i \cap F_j$ is a subfacet.

Depending on d, $|\operatorname{vert}(F_i \cap F_j)|$ and s Normaliz decides which test to use.

Simplicial facets

The intersections of visible and invisible simplicial facets F_i and F_j can be found much faster!

Reason: One knows the subfacets of P contained in a simplicial facet.

Strategy: Make an ordered list of the subfacets of the invisible facets, and search this list for a coincidence with a subfacet of a visible facet.

Running time

$$\approx n(d+1)\log((d+1)n) + p(d+1)\log((d+1)n)$$
$$= (n+p)(d+1)\log((d+1)n)$$

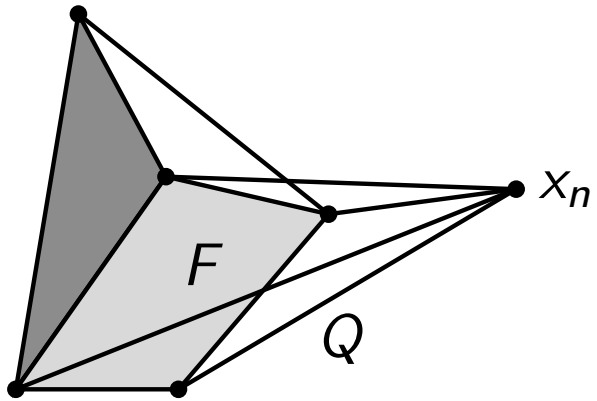
Disregarding d: $\approx (n+p)\log(n)$.

Much, much bettter than np.

Pyramid decomposition

The main technique of Normaliz to break the treacherous double loop is pyramid decomposition, originally for triangulations.

Instead of matching a visible facet with all invisible ones, we make the pyramid $Q = \text{conv}(F, x_n)$. Then we compute the support hyperplanes of Q and check which of them are support hyperplanes of P.



Can be used recursively and is parallelization friendly. It localizes triangulations.

The hybrid algoritm

- If np is small, use the treacherots double loop (with special care of simplicial facets)
- Otherwise go over the visible facets F and decide for each of them whether
  - to compute the support hyperplanes of the pyramid $Q = \text{conv}(F, x_n)$ or
  - to pair F with the invisible facets if Q is too big.

The decision whether Q is too big is based on a criterion that Normaliz "learns" during the run.

Another noteworthy point: Normaliz uses the information on the subfacets that it gets for free.

From a log file

The last extension step of the case $q = 28$ of Köppe & Zhou:

```
Neg 54504 Pos 111500 NegSimp 12717 PosSimp 21988
Building pyramids
.................................................
large pyramids 139
.................................................
gen=157, 195229 hyp          <-------- ORIGINALLY 435 gen
Checking pointedness ... done.
Select extreme rays via comparison ... done.
-------------------------------------------------------------

real 1m34.583s
user 5m37.500s
sys 0m4.684s
```

Fourier-Motzkin vs. the hybrid algorithm

Input	Dim	Vert	Supps	Fourier-Motzkin	hybrid
lo6	16	720	910	39.3 s	35.0 s
cyclo60	17	60	656,100	–	1:57 m
A553	55	75	306,955	2:48 h	9:33 m

lo6 shows that the advantage of the hybrid algorithm is negligible if there are few support hyperplanes: All intermediate cones have $< 10,000$ support hyperplanes.

A somewhat mysterious aspect: the order of the generators. Normaliz orders them lexicographically.