J. Symbolic Computation (2000) **11**, 1–000

On the Computation of Elementary Divisors of Integer Matrices

Frank Lübeck

Lehrstuhl D für Mathematik, RWTH Aachen, Germany

e-mail: *Frank.Luebeck@Math.RWTH-Aachen.De*

(Received original version May 1998, revised version May 2000)

We describe a "semi-modular" algorithm which computes for a given integer matrix A of known rank and a given prime p the multiplicities of p in the factorizations of the elementary divisors of A. Here "semi-modular" means that we apply operations to the integer matrix A but the operations are driven by considering only reductions of row vectors modulo p.

1. Notations and introduction

Let A be an $m \times n$-integer matrix of rank r. We denote the rows of A by $a_1, \dots, a_m$. For the following proposition there are many formulations, depending on the context. The proof can be found in many textbooks, see, e.g., (van der Waerden, 1967, §85).

Proposition 1.1. *(a) There exist invertible integer matrices $L \in GL_m(\mathbb{Z})$ and $R \in GL_n(\mathbb{Z})$, such that $\tilde{A} = LAR$ is diagonal and the diagonal entries $\varepsilon_i = (\tilde{A})_{i,i}$, $1 \leq i \leq \min(m, n)$ fulfill the following properties:*

(i) $\varepsilon_i \in \mathbb{N}_0$
(ii) $\varepsilon_i \mid \varepsilon_{i+1}$ *for* $i < \min(m, n)$ *(in particular* $\varepsilon_i = 0$ *for* $i > r$*)*

(b) The ε_i are uniquely determined by the matrix A. The product $\delta_i = \varepsilon_1 \cdots \varepsilon_i$ equals the (non-negative) greatest common divisor of all determinants of $i \times i$-submatrices of A.

The ε_i are called the *elementary divisors* of A and $\tilde{A}$ is called the *Smith normal form* or *elementary divisors form* of A.

In Section 3 we give an algorithm for computing for a given matrix A, its rank r and a prime p the p-parts of the elementary divisors of A. During the algorithm we perform row operations on the matrix A which are driven by only considering certain reductions of row vectors modulo p. The method does not give transforming matrices L and R as in the proposition.

A crucial point for the application of our algorithm is that one needs to know the prime numbers dividing ε_r (or δ_r). These are often known from the mathematical context. In

0747–7171/90/000000 + 00 \$03.00/0 © 2000 Academic Press Limited

Section 3.2 we shortly comment on an algorithm for computing the inverse of an invertible matrix with rational entries using l-adic approximation for some prime l. Applying this to a square integer matrix A of full rank we can compute the biggest elementary divisor ε_r of A.

This note came out of an application concerning representations of reductive algebraic groups. There we computed matrices describing integral bilinear forms on certain weight spaces, the details of this will be given elsewhere. The matrices appearing in this context are dense with small entries and many nontrivial elementary divisors which are products of small primes (and these primes are known in advance). We wanted to compute the elementary divisors of such matrices with rank up to a few thousands. But when a first version of this note was written, in 1998, we had no implemented algorithm available to handle cases with rank over 250. On the other hand a first implementation of the algorithm described here could handle all of our cases.

In the meanwhile we developed a share package EDIM (Lübeck, 1999) for GAP (GAP, 1999), which contains implementations of several algorithms for Smith normal forms of integer matrices. In Section 4 we describe these and some other new implementations. In Section 5 we look at two (for us) typical examples.

Another application of our program was the computation of Jantzen filtrations of Specht modules of symmetric groups. The results are contained in (Mathas, 1999, Appendix B.4).

Acknowledgements. I wish to thank George Havas for giving some hints to the literature on the computation of Smith normal forms and Arne Storjohann for his comments and suggestions on the original version of this paper.

2. The key lemma

The main ingredient of our algorithm is the following lemma, which is a generalization of the fact that the number of ε_i not divisible by p equals the rank of the reduction of A modulo p over $\mathbb{F}_p$, the finite field with p elements.

Definition 2.1. Let p be a prime number. We call the matrix A as above *p-adjusted*, if there exist $d \in \mathbb{N}$ and $r_j \in \mathbb{N}_0$ for $-1 \leq j \leq d+1$ with the following properties.

(i) $r_{-1} = 0 \leq r_0 \leq \ldots \leq r_d = r \leq r_{d+1} = m$.
(ii) If $r_{l-1} < i \leq r_l$ then $a_i = p^l a'_i$ for some row $a'_i \in \mathbb{Z}^n$.
(iii) The reductions of $a'_1, \ldots, a'_r$ modulo p are linearly independent over $\mathbb{F}_p$.

Hence the rows $a_{r_{l-1}+1}, \ldots, a_{r_l}$ are divisible by p^l and if we divide the first r rows by these p-powers, we get rows which are linearly independent modulo p.

For a p-adjusted matrix we can easily determine the multiplicity of p in the factorizations of the δ_i (and hence of the ε_i).

Lemma 2.2. *Let A be p-adjusted and $r_{l-1} < i \leq r_l \leq r$. Then the multiplicity of p in the factorization of δ_i is*

$$m_i := (r_1 - r_0) + 2(r_2 - r_1) + \cdots + (l-1)(r_l - r_{l-1}) + l(i - r_{l-1}).$$

PROOF. From the definition of p-adjusted it follows immediately that each determinant of an $i \times i$-submatrix, and so δ_i, is divisible by p^{m_i} (even by higher powers of p if the submatrix uses rows a_j with $j > r_l$).

On the other hand consider the matrix with the i rows $a'_1, \ldots a'_i$. Since this has rank i modulo p it contains i columns such that the determinant of the corresponding submatrix is not divisible by p. Hence the corresponding submatrix of A divided by p^{m_i} is not divisible by p. This shows that δ_i is not divisible by a higher power of p than p^{m_i}. □

3. The algorithm

The previous lemma shows that in order to find the highest p-powers dividing δ_i or ε_i we only need to transform A into a p-adjusted matrix. This is achieved by the following algorithm which uses only row operations and permutations of the columns of A. More precisely we construct in the k-th main step of the algorithm rows $a'_{r_{k-1}+1}, \ldots, a'_{r_k}$, $k = 0, \ldots, d$, using the notation of Definition 2.1. In the following description of the algorithm we consider matrices as lists of row vectors.

It is shortly explained as follows. We triangulize A modulo p in the obvious way by row operations and column permutations. But all transformations are actually done with the original rows of A. If we find a row vector which is zero modulo p we divide all its entries by p and use it again in the next main step.

Here is a more detailed description.

ALGORITHM 3.1.
Input: An $m \times n$-integer matrix A, its rank r and a prime p.
Output: d and $r_0, \ldots, r_d$ as in Definition 2.1
Initialization:

set A' to the empty list *(used to collect rows a'_i of p-adjusted transform of A)*
set B to the input matrix A
set k to 0 *(numbering the main steps, used as index for the r_k)*

Main loop: (we are done if A' has r rows)

while number of rows of A' is smaller than r do

set B' to the empty list *(used to collect vectors for the next step)*
for each row $v = (v_1, \ldots, v_n)$ in B do

(reduce v modulo p with rows of A')
for i from 1 to number of rows of A' do

let a'_i the i-th row of A'
determine $c \in \mathbb{Z}$, $|c| \leq p/2$ with p divides i-th entry of $v - ca'_i$
substitute v by $v - ca'_i$

end for
if all entries of v are divisible by p then

append $\frac{1}{p} \cdot v$ as new row to B'

else

set i to the number of rows of A' plus 1
determine minimal j such that p does not divide j-th entry of v
append v as new row to A'
if $i \neq j$ then exchange the i-th and j-th column of A', B and B'
(so A' is always triangular modulo p)

end if

end for
set r_k to the number of rows of A'
set k to $k+1$
set B to B'

end while
set d to $k-1$
return d and $r_0, \dots, r_d$

3.1. Remarks

(1) If we are not finished after step number k of the algorithm we see as in the proof of Lemma 2.2 that each determinant of an $r \times r$-submatrix of A is at least divisible by p^k. This shows the termination of the algorithm.

(2) In practice we do not perform the column permutations in the algorithm but just store a list which tells us the order in which the entries of the vectors v must be reduced. To find the constants c for the reductions we compute once for each diagonal entry of A' its inverse modulo p (via the extended Euclidean algorithm) and store it.

(3) Instead of using the rank of A it is also possible to use the highest power of p dividing δ_r as a criterion for finishing the algorithm. In this case we can already stop the algorithm after finding an $r_k = r-1$.

(4) (Thanks to Arne Storjohann for this remark.) With a bit more overhead one can further reduce the size of the matrix entries occurring during the computations in steps where $r_k - r_{k-1}$ is not too small: In the for-loop inside the Main loop do first all operations with rows of B and A' just modulo p. Do not append reduced vectors to A' or B' directly but remember how they were computed as linear combinations of the rows of B and A' from the previous step. Now reduce the coefficients of these linear combinations modulo p in the range $]-p/2, p/2]$ and use these to recompute the new vectors appended to A', B'.

(5) If we know a $d' > d$ (for example if we know ε_r or δ_r) then we can reduce all matrix entries modulo $p^{d'}$ during the algorithm. It is clear that this will not change the output of the algorithm. In practice it will be sufficient to do the reduction only from time to time, e.g., before storing the reduced vector v in A' or B'. Note that we can safely guess some d' and then try the algorithm. If after step number $d'-1$ the matrix A' still has less than r rows, our guess was too small and we have to redo the calculations with a bigger d'.

(6) In the case that we know the highest p-power p^m dividing δ_r and m is at most 3 our algorithm is not needed. For $m=1$ clearly only ε_r is divisible by p and for $m = 2, 3$ it is sufficient to reduce the whole matrix modulo p and to compute its rank modulo p.

(7) It is not difficult to estimate the number of operations needed in Algorithm 3.1 and the coefficient growth during the computation. We will do this in terms of the numbers $m, p, r_0, \dots, r_d$.

Neglecting some bookkeeping the algorithm consists of operations of type $v - cw$ where $v, w \in \mathbb{Z}^n$ are row vectors and $c \in \mathbb{Z}$ with $|c| \le p/2$. In step k, $0 \le k \le d$, we have to reduce $m - r_{k-1}$ rows in B' with at most r_k rows from A'. Adding up we see that there are less than

$$mr_0 + (m - r_0)r_1 + \ldots + (m - r_{d-1})r_d$$

such row operations during the algorithm.

Let M_{-1} be the maximal absolute value of all entries in A and M_k, $0 \le k \le d$, the maximal absolute value of all entries in A' and B' after step k of the algorithm. In step k we first have to reduce all rows in B' with the rows in A' which have been found in the previous step. The resulting rows have entries of absolute value at most $M_{k-1}(1+r_{k-1}p/2)$. These rows have then to be triangulized modulo p leading to $r_k - r_{k-1}$ new rows of A'. From this we see

$$M_k \le M_{k-1}(1 + r_{k-1}p/2)(1 + p/2)^{r_k - r_{k-1}}.$$

With the variant of the algorithm given in (4) one must do all row operations twice, one time modulo p and one time over the integers. But the entries in the new rows of A' and B' can be considerably smaller in steps with big $r_k - r_{k-1}$. Here each of these new rows is computed as linear combination of at most r_k of the old rows and with coefficients of absolute value at most $p/2$. In this case we have

$$M_k \le M_{k-1} r_k p/2.$$

And, of course, in the variant of the algorithm given in (5) each $M_k < p^{d'}$. This is what we use mostly in practice.

3.2. Finding the relevant primes

For applying the modular methods discussed above it is necessary to know the (possible) prime divisors of δ_r or ε_r. In many practical situations (like in those mentioned in the Introduction) these are known from the mathematical context, but if not we need some precomputation to find a set of primes to consider.

Note that in case one knows δ_r or ε_r it still can be very difficult to find the prime divisors of this number. But this will often not be necessary: Assume that we know those small prime divisors of this number which are easy to obtain. Then it is very probable (in particular for almost random matrices) that the prime divisors of a remaining big factor m only appear in the last elementary divisor ε_r. If this is true it can be proved by computing the rank of the matrix modulo m (as if m was a prime) and finding $r - 1$. It doesn't matter if we run into an error because some number is not invertible modulo m (which will almost never happen) since in that case we have found a factor of m.

One idea for finding a multiple of δ_r is to compute the greatest common divisor of some $r \times r$-submatrices of the given matrix. These can be computed without any problem of entry explosion modulo sufficiently many primes and combined using the Chinese remainder theorem. See (Havas and Sterling, 1979) and (Havas et al., 1993) for more details.

Assume now that we are given a square integer matrix A of full rank r. Then the biggest elementary divisor ε_r equals the least common multiple of the denominators of the entries of the inverse matrix A^{-1} where A is considered as matrix over the rational numbers $\mathbb{Q}$. Let l be a prime such that A is also invertible modulo l. In (Dixon, 1982) a method is described to compute the rational solution of a linear system $xA = v$ for a vector v with integer entries. One first computes the inverse of A modulo l, then an l-adic approximation to a solution and finally reconstructs the rational entries of the solution vector with a variant of the extended Euclidean algorithm.

We found that this can be used for computing efficiently the inverse of large integer matrices (and so their largest elementary divisor). The algorithm above is essentially applied to v running through the standard basis vectors. But instead of taking the standard vectors themselves we always multiply them with the least common denominator of the already determined entries of A^{-1}. Very often we find the largest elementary divisor of A already after computing the first or first few rows of A^{-1}. From then on one finds the next rows of A^{-1} in the l-adic approximation step and the reconstruction step is not necessary. In the manual of (Lübeck, 1999) we give some more details on this.

4. Implementations of algorithms for Smith normal forms

In this section we assume that the reader is familiar with the standard algorithm for computing the Smith normal form. (One has to repeat steps of the kind: choose an entry as pivot, permute rows and columns to get this entry to the upper left position and reduce the other entries in the first row and column by row and column operations.) We list here some interesting current implementations of algorithms for computing the Smith normal form of integer matrices.

(1) Our software package EDIM (Lübeck, 1999) contains implementations of Algorithm 3.1 and several variations mentioned in the Remarks 3.1. There is a particularly well performing version of the variant 3.1(5) with reductions modulo $p^{d'}$ which is used if $p^{d'+1}$ fits into machine integers. (This is often the case for the matrices in our applications.)

(2) A good overview, including a long reference list, of methods for computing Smith normal form is given in the article (Havas et al., 1993). It is explained that the main problem in these computations is the effect of *entry explosion*, i.e., even if the entries of the input matrix as well as the elementary divisors are small numbers, a naive implementation of the standard algorithm can lead to very large numbers during the computation.

(3) There are also newer articles on the topic. See for example (Giesbrecht, 1995), (Storjohann and Labahn, 1996) and (Storjohann, 1997) which contain asymptotically fast algorithms including a complexity analysis. In particular the algorithms given by Storjohann turn out to be of practical importance. They are now available in GAP (GAP, 1999) and there is also an implementation in form of a stand alone program (thanks to Arne Storjohann for sending this).

(4) Another practical algorithm is given in (Havas and Majewski, 1997), where a certain pivoting strategy for the standard algorithm is discussed which tries to reduce the coefficient growth during the computation. This is also available via GAP (GAP, 1999).

(5) In (Havas et al., 1998) the standard algorithm is combined with an LLL-lattice reduction algorithm. This is particularly interesting for finding transforming matrices to the Hermite and Smith normal form with small entries. Also all matrix entries stay small during the computation. We have implemented this algorithm in (Lübeck, 1999).

(6) Another modular algorithm can be found in (Havas and Sterling, 1979). We have implemented this algorithm (with a slight improvement) in (Lübeck, 1999), too. The idea is as follows: Given an integer matrix A, a prime p and a $d' \in \mathbb{N}$ such that $p^{d'}$ does not divide ε_r (similar to the setup in Remark 3.1(5)). Then we can essentially compute the

p-parts of the elementary divisors using the standard algorithm, with two simplifications. For the pivot search we only have to find an element with minimal p-part. The many greatest common divisor computations in the standard algorithm are substituted by just one for each pivot - the p'-prime part of the pivot is inverted modulo $p^{d'}$. Furthermore all computations can be done modulo $p^{d'}$. The inversion modulo $p^{d'}$ in this algorithm causes that most numbers appearing in the computation have roughly the same size as $p^{d'}$. Note that in this algorithm in the i-th main step the p-part of ε_i is computed, whereas in our Algorithm 3.1 we work in the "orthogonal" direction and compute in the k-th main step the number of ε_i divisible by p^k.

It is crucial for this algorithm that $p^{d'}$ is not too big. Its performance becomes worse if large powers of p are concentrated in the last few elementary divisors.

To compare: In our Algorithm 3.1 all row reductions are made with coefficients of absolute value less or equal $p/2$ and not those of size $p^{d'}$ or even δ_r. It even works nicely for large matrices with small entries and small p in the bare version described in Algorithm 3.1, i.e., without any reductions modulo some $p^{d'}$.

(7) Finally we mention that Magma (Bosma et al., 1997) also contains an algorithm for computing elementary divisors which performed well for examples we have tried. I was told that the code is based on the experiences described in (Havas et al., 1993).

5. Examples

Since we are mainly interested in using our algorithm for large matrices we give here some details how the implementations of elementary divisors algorithms mentioned in the previous section behave for two of our typical input matrices. They have large rank and many non-trivial small elementary divisors. (Such matrices with small ε_r also appear in many examples in the articles cited above.)

Matrix A_1 is a 242×242-matrix of full rank which is a Gram matrix of a weight space of a highest weight representation of some reductive group. It has determinant $2^{357}3^{260}5^{122}$, entries in the range $[-63, 178]$ and the largest elementary divisors are $2^9 3^2 5^2$.

Matrix A_2 is a 2002×2002-matrix of full rank which is the Gram matrix of the Specht module of the symmetric group S_{15} parameterized by the partition (222222111) of 15. It has determinant $2^{19930}3^{11425}5^{4381}7^{1652}$, entries in the range $[-29030400, 261273600]$ and the largest elementary divisors are $2^{10}3^6 5^3 7^1$.

All timings given below (in hours (h), minutes (m) and seconds (s)) are determined on a 500MHz-Pentium III PC with 256 Megabyte of RAM, running under a GNU/Linux operating system.

For both matrices we actually know the prime divisors of the determinant in advance from the mathematical context. Ignoring this and using the idea described in 3.2 for computing ε_r by p-adic approximation we could compute the largest elementary divisors of A_1 and A_2 within 5.6s respectively 9h07m10s. Actually we found all relevant primes already after the computation of the first row of the inverse matrices, after 1.2s respectively 3m27s. So, most of the time was needed to confirm them. (This approach works particularly well in cases like here where the largest elementary divisor is very small compared to that of a random matrix of similar size.)

In the following table we give running times of the programs mentioned above, when determined. An entry *not enough memory* means that the program tried to use more than the available 256 Megabytes. In such cases we stopped the computation. The modular algorithm by Havas-Sterling, see Section 4(6), was tried with minimal possible exponents

d' - in this case all computations are done with machine integers - and with the power of p appearing in the determinant, as suggested in (Havas and Sterling, 1979).

Algorithm	Time for A_1	Time for A_2
Section 4(3), Storjohann, GAP	1m32s	not enough memory
Section 4(3), Storjohann, C	31.5s	not enough memory
Section 4(7) Magma	140.5s	47h31m27s
Section 4(4), Havas et.al., GAP	3 days	not enough memory (after 5 days)
Section 4(5), LLL, EDIM	3h12m	estimated several weeks
Section 4(6), Havas-Sterling, primes known, d' minimal	30.4s	5h34m21s
Section 4(6), Havas-Sterling, primes known, d' from det	13m05s	not enough memory
Algorithm 3.1, EDIM	29.8s	47m24s for $p = 2, 3$, not enough memory for $p = 5, 7$
Algorithm 3.1, Remark 3.1(4), EDIM, primes known	9.8s	2h32m16s
Algorithm 3.1, Remark 3.1(5), EDIM, primes known	0.6s	3m49s

In the case of Algorithm 3.1, Remark 3.1(4), we have also determined the maximal absolute values of entries in A', as estimated in 3.1(7). For example for A_2 they were between 132524100 for $p = 2$ and 76204800000 for $p = 7$. For A_1 and $p = 5$ the maximal entry was 184340 whereas in the bare Algorithm 3.1 the maximal entry had 55 decimal digits.

The matrix A_1 was one for which we did not have any program to compute its elementary divisors before we started to think about the algorithm presented in this note. Remarkably, now all of the programs considered here could handle it. But from our examples it becomes clear that a modular approach is very appropriate for large matrices of a similar type, whenever there is a chance to find the relevant primes in reasonable time. Our algorithm in one of the variations is the only convenient one which still works nicely for even larger cases (our largest practical case so far had dimension 7700, which is the highest dimension of a Specht module of the symmetric group S_{12}).

References

Bosma, W., Cannon, J., and Playoust, C. (1997). The Magma algebra system I: The user language. *Journal of Symbolic Computation*, 24:235–265. (see `http://www.maths.usyd.edu.au:8000/u/magma/`).

Dixon, J. D. (1982). Exact solution of linear equations using p-adic expansions. *Numer. Math.*, 40:137–141.

GAP (1999). *GAP – Groups, Algorithms, and Programming, Version 4.2*. The GAP Group, Aachen, St Andrews. (see `http://www-gap.dcs.st-and.ac.uk/~gap`).

Giesbrecht, M. (1995). Fast computation of the Smith normal form of an integer matrix. In *ISSAC'95*, pages 110–118. ACM Press.

Havas, G., Holt, D. F., and Rees, S. (1993). Recognizing badly presented Z-modules. *Linear Algebra and its Applications*, 192:137–164.

Havas, G. and Majewski, B. S. (1997). Integer matrix diagonalization. *Journal of Symbolic Computation*, 24:399–408.

Havas, G., Majewski, B. S., and Matthews, K. R. (1998). Extended gcd and Hermite normal form algorithms via lattice basis reduction. *Experimental Mathematics*, 7:125–135.

Havas, G. and Sterling, L. S. (1979). Integer matrices and abelian groups. In *Symbolic and algebraic computation*, volume 72 of *Lecture Notes in Computer Science*, pages 431–451. Springer-Verlag, Berlin.

Lübeck, F. (1999). *EDIM - elementary divisors and integer matrices*. Lehrstuhl D für Mathematik, RWTH Aachen. (a refereed share package for (GAP, 1999), see `http://www.math.rwth-aachen.de/~Frank.Luebeck/EDIM`).

Mathas, A. (1999). *Iwahori-Hecke algebras and Schur algebras of the symmetric group*, volume 15 of *University Lecture Series*. American Mathematical Society.

Storjohann, A. (1997). A solution to the extended GCD problem with applications. In *ISSAC'97*. ACM Press.

Storjohann, A. and Labahn, G. (1996). Asymptotically fast computation of Hermite normal forms of integer matrices. In *ISSAC'96*, pages 259–266. ACM Press.

van der Waerden, B. L. (1967). *Algebra*. Springer-Verlag, Berlin, Heidelberg, New York.