

Modified Gauss Algorithm for Matrices with Symbolic Entries.

Benno Fuchssteiner
Oberheideweg 19
D 33106 Paderborn
Germany

Abstract

We propose a modification of the usual algorithms for Gaussian elimination, determination of minors and computation of determinants in case of matrices with symbolic data (polynomials, rational functions, general algebraic structures and the like). The proposed modification improves on the Sasaki-Murao algorithm. The new algorithm aims at improving efficiency for a general class algebraic data like polynomials with non exact coefficients, algebraic structures with zero divisors, rational functions and general functions in several variables. The main difference to other approaches is that we replace the diagonal elements not at the beginning of the algorithm but then when they are taken as a pivot elements (independent whether or not these elements are non-zero). Therefore, our method abolishes the steps where pivot elements are checked to be non-zero, and even zero elements can be used as pivots.

1 Introduction

Let M be an $n \times n$ matrix with symbolic entries. By *symbolic entries* we mean that the entries are chosen from a suitable commutative ring (with or without zero divisors). In many applications¹ the problem arises how to invert this matrix or how to perform operations which come *close* to constructing the inverse of this matrix (for example finding the matrix consisting of its minors)². When M is invertible then for some algebraic structures the usual Gauss algorithm is an efficient way to solve both the determination of the minors as well as the inversion.

However, even in case the matrix may be invertible, then for symbolic entries, this algorithm poses a serious efficiency problem, this insofar as costly normalization procedures are needed for quotients, for example for the determination

¹For example when for a dynamical system, having a full set of symmetry generators, the conservation laws on an invariant submanifold shall be computed ([4], [3]).

²The minor $minor_{ik}$ is the determinant of the matrix where the i -th column and the k -th line have been deleted from M .

of whether or not a pivot element is zero. Also for carrying out cancellations, the necessary normalization of quotients can be costly and a considerable data swell has to be expected.

Already in 1968 Bareiss in [2] presented his fraction-free Gaussian algorithm, which was a serious improvement over the usual elimination procedures. Observing, for example, in case of determinants, that by the Laplace expansion method no fractions are to be expected, he carried out all steps where an exact division (i.e. without remainder term) was possible. He applied this idea also to an integer-preserving elimination for solving systems of equations in case of integers or univariate polynomials.

Since the advent of computer algebra systems, and when it became obvious that even in real-life applications (like robotics, only to name one area) other algebraic domains than rational numbers or floating point numbers were useful to consider, the interest of improving on the Gauss algorithm for the determination of inverses and determinants has been growing. Although, in principle, the minor expansion algorithm, is optimal (at least in certain cases), Bareiss' algorithm proved to be of advantage with respect to the use of available memory; this insofar as it reduced intermediate data swell in comparison to the minor expansion algorithm (see for example the discussion of different algorithms in the interesting survey [6]). Improvements for the minor expansion algorithm with respect to the memory problem are due to Smit [12].

In 1982 T. Sasaki and H. Murao [10] improved on Bareiss' algorithm by observing that the computation is less demanding in case the diagonal of a matrix contains independent variables, which do not appear elsewhere. So they replaced the diagonal elements by independent variables, which allowed them to drop those terms, which should cancel anyway. This was possible since by that replacement, at essential steps, only a product of the squares of the first k independent variables should remain. Thus all other terms could be dropped without computing whether or not they really cancel. The computation was even more facilitated by the introduction of a special kind of product, taking these structural features into account in such a way that the terms, which could be dropped anyway, were not created from the beginning.

Even after all this progress in 2006 the authors of [13] remark "Computation of determinants of rational functions seems to be out of thought in computer algebra so far". Unfortunately, this seems to be true. In order to open the avenues for overcoming these difficulties, these authors introduced a special replacement for different denominators, which then were treated as independent variables (or rather their inverses). Another interesting and important development, dealing with a different "algebraic structure" has been considered by Sasaki and Sato

[11]. They take into account incorrect algebraic data insofar as polynomials with floating point coefficients are considered, where rounding errors, or even worse, cancelation errors may occur.

All these algorithms aim at special algebraic structures, almost always unique factorization domains or even univariate polynomials (for example see also [9], [7] and as survey [1]).

However, an approach dealing with all these different algebraic structures and extending the ideas behind the many and important contributions to even more general structures, like non unique factorization domains, remainder classes, and even structures with zero divisors, or structures consisting only of zero divisors is still missing. That is the point where this paper tries to make a contribution. Since we deal with possibly all commutative algebraic structures, that is, a variety of data of highly different complexity, we, as other authors, for good reason do not concentrate on the investigation of the complexity of our algorithm (see the remarks on complexity later on).

In a certain sense, we follow the ideas of Sasaki et al. [10], [13], [11] insofar as we introduce formal variables in order to have a transparent and exact division (without remainders) wherever a division is necessary. However, such a division is only necessary in intermediate and explanatory algorithms, in the final form of the algorithm no division will be necessary at all.

The difference to the replacements suggested in [10] is, that we introduce the new variables not at the beginning of the elimination process, but at those points where new pivots are considered. Furthermore, it turns out that using instead of variables itself the use of their inverses reduces intermediate data swell considerably. This gives a very simple structure with respect to correct divisibility, so that division can be replaced by substitution, thus guaranteeing an division free procedure.

The result is an algorithm which works for all kind of commutative algebraic structures, also structures with incorrect data. The algorithm works in such a way, that the dropping of terms due to incorrect data, becomes a most obvious procedure. Therefore the amazing effect of our algorithm is, among others, that incorrect operations are corrected during the algorithm (see Example 9).

2 The algorithm

Here, we present the modified Gauss algorithm, in a recipe-like form. The Z_1, Z_2, \dots, Z_n are independent formal variables.

Algorithm A (Elimination):**INPUT:** An $n \times n$ matrix M .**OUTPUT:** The modified matrix L and $Mlist$, the list of memorized expressions.**START:** Set $L = (M \ I)$ (augmented matrix) to be the matrix where to M the $n \times n$ unit matrix is appended.Set $Mlist := []$ (memory list) to be an empty list.For k from 1 to n perform the following operations on L and $Mlist$:

- (1) Enter the element $L[k, k]$ as k -th element in the memory list $Mlist$.
- (2) Replace the element $L[k, k]$ by $1/Z_k$.
- (3) In order to obtain a diagonal matrix eliminate the elements above and below $L[k, k]$ by adding a multiple of the k -th row to the corresponding row. That means, replace for $j \neq k$ the $L[j, r]$ by $L[j, r] - Z_k L[k, r] L[j, k]$
- (3.1) Delete in all $L[j, r], j \neq k$ all terms containing any of the $Z_1^m, Z_2^m, \dots, Z_n^k$ with power $m > 1$.
- (alternative) In order to obtain only an upper triangle matrix, perform the same operations as in (3) and (3.1), but only for the $j > k$.
- (4) Multiply the k -th row of the resulting L by Z_k .

end.

END: Return L and $Mlist$.**Algorithm B (Evaluation):****INPUT:** Any polynomial expression P in the variables Z_1, Z_2, \dots, Z_n , and the $Mlist$ from the elimination algorithm.**OUTPUT:** The modified polynomial P .For k from n **down to** 1 perform the following operations on P :

- (1) Decompose $P = P_0 + Z_k P_1 + Z_k^2 P_{\text{rest}}$ (Taylor polynomial of second order), where P_0 and P_1 are independent of Z_k . Let $Mlist_k$ be the k -th element of $Mlist$.
- (2) Set $P := Mlist_k P_0 + P_1$

end.

END: Return P .

Remark 1. (i) In both algorithms it is essential in which order the operations are performed, one has to start with Z_n and then go down until the Z_1 is reached.

(ii) In the evaluation algorithm all terms containing any of the $Z_1^m, Z_2^m, \dots, Z_n^m$ with $m > 1$ are put to zero. This feature is essential for reducing intermediate data swell.

(iii) If only *Mlist* is needed, then of course it suffices to use the alternative in the elimination algorithm since only an upper triangle matrix for the memory list is needed.

(iv) The k -th element of *Mlist* does not contain any of the Z_k, Z_{k+1}, \dots, Z_n .

(v) Hence, after the k -th step of the evaluation algorithm P does not contain any of the variables Z_k, \dots, Z_n and the final result is independent of these auxiliary variables.

Result 2:

(i) Taking the sub matrix given by rows $n + 1$ to $2n$ of the result of the elimination algorithm, then application of the evaluation algorithm to its elements, yields the matrix of minors of M .

(ii) Application of the evaluation algorithm to 1 yields the determinant of M .

Example 1.

In order to understand what the various steps of the algorithm do we present here the most elementary example which is possible: a 2×2 matrix with integers as entries. We consider the matrix

$$M = \begin{pmatrix} 628 & 837 \\ -51 & -28 \end{pmatrix}.$$

The memory list which we obtain for this case is:

$$Mlist = [628, 42687 Z_1 - 28],$$

and algorithm *A* yields for columns 3 to 4 of the augmented matrix

$$\begin{pmatrix} Z_1 & -837 Z_1 Z_2 \\ 51 Z_1 Z_2 & Z_2 \end{pmatrix}.$$

Now, if the evaluation according to algorithm B is performed we obtain as matrix of minors

$$\begin{pmatrix} -28 & -837 \\ 51 & 628 \end{pmatrix}.$$

If we had left out the deletion of higher order terms in algorithm A (steps (3.1)) then before carrying out step (4) the same part of the matrix would have been

$$\begin{pmatrix} 1 - 42687 Z_1 Z_2 & -837 Z_2 \\ 51 Z_1 & 1 \end{pmatrix}.$$

So even in this most simple case there is, before elimination, a reduction in data swell to be seen. This data swell increases tremendously for higher dimension of the matrix.

If one wants to check the advantage over the usual algorithms in a case which still is rather simple then one should use data which are really symbolic data, for example a random matrix with rational functions in one variable: In MuPAD [5] such an example for dimension 6 is generated with the following MuPAD command

```
n:=6: M:=matrix(n,n, (i,j) ->
extop(polylib::randpoly([z]),1)/extop(polylib::randpoly([z]),1)):
```

Or one should try out matrices having only different identifiers in each entry:

```
n:=6: M:=matrix(n,n, (i,j)->m[i,j]);
```

Examples like this can be found on the web page:

www.fuchssteiner.ch/mupprgms/mod_gauss.html

3 Pseudo elimination

The evaluation procedure in algorithm B certainly looks somewhat strange, and is certainly difficult to understand on an intuitive level; therefore we need a proof. However, instead of a proof in the usual sense, we prefer to give an algorithmic proof, i.e. a proof obtained by gradually changing obvious algorithms. We start by presenting an elimination algorithm, which is fairly simple. The computation is performed in the ring of formal polynomials in the variables X_1, X_2, \dots, X_n with coefficients in the algebra under consideration. Observe that we change the variables from Z_i to X_i . This is done in order to obtain later on

useful information by comparison of these variables.

Algorithm 1 (Pseudo elimination):

INPUT: An $n \times n$ matrix M and
a set of independent variables X_1, X_2, \dots, X_n .

OUTPUT: Matrix after pseudo elimination, $Elist$ the list of substitutions,
and - for explanatory purpose - the matrix N .

START: $L := (M \ I)$ an $n \times 2n$ augmented matrix, where
to M the $n \times n$ unit matrix I is appended,
 $Elist := []$ an empty list (for storing substitution equations).
#b1 $N := n \times n$ zero matrix (for memorizing intermediate steps).
For i from 1 to n do:
#i1 Append $X_i = L_{i,i}$ to $Elist$.
#i2 Replace $L_{i,i}$ in L by X_i .
For j from 1 to n with $j \neq i$ do begin:
#bi $N_{j,i} := L_{j,i}$ (for later purpose).
#ij Replace $row_j(L)$ of L by $X_i \times row_j(L) - L_{j,i} \times row_i(L)$
end.
end.
END: Return L , $Elist$ and N .

For the final algorithm we do not need the steps **#b1** and **#bi**, these are only included for explanatory purpose.

Indeed, the algorithm is very similar to all modified Gauss eliminations performed for dealing with symbolic data. The only difference between this elimination algorithm and the one proposed by Sasaki and Murao in [10] is that the formal diagonal elements are not introduced at the beginning but during the elimination process, this however has important consequences. For example, no zero-checking and pivot-changing is necessary. This advantageous feature is achieved by the fact that our initial matrices would differ considerably from those forming the basis for other algorithms. The merit of replacing $L_{i,i}$ by X_i in step **#i2** is that no quotient is computed at all; in Sasaki-Murao's algorithm, quotients are computed by successive subtractions. The backward algorithm included below does, among other purposes, demonstrate this effect. So, for explaining this aspect of algorithm 1, especially the changes obtained from the replacement steps **#i2**, we need some inverse operation, i.e. we need to determine which matrix leads to the same result for L if the steps **#i2** were left out, that is, if the original Gauss elimination were performed.

The result of this inverse operation is simply obtained by running backwards

through the algorithm and leaving out #i2 (for this the memorized elements in N are needed). Although the computation of this inverse is not needed in the final algorithm we consider it an essential step for understanding our method, especially for understanding why fractions can be avoided altogether. Therefore we have to include this simple algorithm.

The domain of computations is the ring of formal polynomials in the variables $X_1, X_2, \dots, X_n, X_1^{-1}, X_2^{-1}, \dots, X_n^{-1}$ with coefficients in the algebra under consideration.

Algorithm 2 (Backwards):

INPUT: The $n \times 2n$ matrix L and the memorized elements in N being the result of the preceding algorithm.
OUTPUT: The modified matrix Λ .
START: Put $\Lambda := L$.
For i from n **down to** 1 do begin
For j with $j \neq i$ from n **down to** 1 do replace row $_j$ (Λ)
#bij of Λ by $(1/X_i \times \text{row}_j(\Lambda) + 1/X_i \times N_{j,i} \times \text{row}_i(\Lambda))$.
end.
end.
END: Return Λ

The result of this Backwards algorithm is essential for understanding the algorithm.

We present some examples of utmost simplicity for these operations, one example where all pivots are non zero and one with only zero pivots.

Example 2.

Let

$$M = \begin{pmatrix} 4 & 1 & 5 \\ 0 & 2 & 3 \\ 2 & 1 & 4 \end{pmatrix}, \tag{1}$$

then we obtain by the pseudo elimination algorithm for Elist

$$Elist = [X_1 = 4, X_2 = 2 X_1, X_3 = (4 X_1 - 10) X_2 - 3 (X_1 - 2) X_1], \tag{2}$$

and $L = (\text{left}(L), \text{right}(L))$ to be the concatenation of the following matrices

$$\text{left}(L) = \begin{pmatrix} X_1 X_2 X_3 & 0 & 0 \\ 0 & X_2 X_3 & 0 \\ 0 & 0 & X_3 \end{pmatrix}, \tag{3}$$

$$\text{right}(L) = \begin{pmatrix} X_2 X_3 - 2 X_2 (3 X_1 - 5 X_2) & -X_1 X_3 - X_1 (3 X_1 - 5 X_2) (X_1 - 2) & X_1 X_2 (3 X_1 - 5 X_2) \\ 6 X_1 X_2 & 3 X_1^2 (X_1 - 2) + X_1 X_3 & -3 X_1^2 X_2 \\ -2 X_2 & -X_1 (X_1 - 2) & X_1 X_2 \end{pmatrix}, \quad (4)$$

where $\text{left}(L)$ is an $n \times n$ matrix transformed from M and $\text{right}(L)$ is a matrix transformed from $n \times n$ unit matrix. Running this concatenation through the Backwards algorithm we obtain

$$\text{left}(\Lambda) = \begin{pmatrix} X_1 & 1 & 5 \\ 0 & \frac{X_2}{X_1} & 3 \\ 2 & 1 & \frac{3X_1^2 - 6X_1 + 10X_2 + X_3}{X_1 X_2} \end{pmatrix}, \quad (5)$$

$$\text{right}(\Lambda) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \quad (6)$$

Running $\text{left}(\Lambda)$ again through the pseudo elimination process we obtain the same result as for M , this is independent whether or not the steps #i2 are left out. Hence L can be obtained from $\text{left}(\Lambda)$ by an honest Gauss elimination. It is no surprise, that $\text{right}(L)$ the right part of the matrix is transformed to the unit matrix. This is clear because in this part no substitutions took place and hence the backward algorithm for that part leads to the exact inverse of the pseudo elimination. \square

Example 3.

Let

$$M = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad (7)$$

then the same steps as before yield

$$E\text{list} = [X_1 = 0, X_2 = 0, X_3 = 0], \quad (8)$$

$$\text{left}(L) = \begin{pmatrix} X_1 X_2 X_3 & 0 & 0 \\ 0 & X_2 X_3 & 0 \\ 0 & 0 & X_3 \end{pmatrix}, \quad (9)$$

$$\text{right}(L) = \begin{pmatrix} X_2 X_3 & 0 & 0 \\ 0 & X_1 X_3 & 0 \\ 0 & 0 & X_1 X_2 \end{pmatrix},$$

$$\text{left}(\Lambda) = \begin{pmatrix} X_1 & 0 & 0 \\ 0 & \frac{X_2}{X_1} & 0 \\ 0 & 0 & \frac{X_3}{X_1 X_2} \end{pmatrix}, \quad (10)$$

$$\text{right}(\Lambda) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \quad (11)$$

□

Carrying out in $\text{left}(\Lambda)$ the substitutions gathered in $E\text{list}$ one should obtain the original matrix M . However, on first view this looks disturbing insofar as a lot of illegal divisions by zero may have to be performed. Even more disturbing looks, that for getting back the zero matrix most of the terms in the diagonal are singular. Nevertheless, performing the substitutions in the **right order** these disturbing aspects disappear.

Remark 2. (*Backsubstitutions*)

If the substitutions contained in $E\text{list}$ for the X_i are carried out in the inverse order of $E\text{list}$, i.e. the X_n first, then X_{n-1} and X_1 last, then no divisions are necessary since the resulting term after the substitution for X_{k+1} is divisible by X_k . Therefore the next division (by X_k) can be carried out by the global substitution

$$X_k^s \rightarrow X_k^{s-1}. \quad (12)$$

Proof. We assume that in $\text{left}(\Lambda)$ all substitutions have been carried out according to (12), down to X_{i+1} . We observe that the division by X_i in the j -th row of the resulting matrix comes from step $\#b\text{ij}$ in the backwards algorithm, this step was reversing step $\#i\text{j}$ in the elimination algorithm. This reversion is done by first adding L_{ji} times $\text{row}_i(L)$ of L (which is the same as Λ at this stage) to $\text{row}_j(\Lambda)$ of Λ . This results according to $\#i\text{j}$ in $X_i \times \text{row}_j(L)$ of L . Hence the resulting row is an X_i multiple of some row in L and the division can be carried out by just reducing the exponents by 1 in any term X_i^s . □

This action of *reducing the exponents* is a most simple and obvious procedure:

Algorithm 3 (Reduction of polynomials): $\text{red_pol}(X_i, s)$

INPUT: A polynomial $P(X_i)$ in the variable X_i , a positive integer s .

OUTPUT: The Polynomial where the replacements are carried out.

START:

$\#r\text{i}$ For $s \leq k$ do replace in $P(X_i)$ all powers X_i^k by $X_i^{(k-s)}$ end.

#di For $s > k$ do replace in $P(X_i)$ all powers X_i^k by 0 end.
 END: Return $P(X_i)$.

Remark 3. *So, when carrying out the substitutions from Elist, then from X_n down to X_1 we have to perform after each substitution for X_{i+1} an application of $\text{red_pol}(X_i, 1)$. The effect of step #di is nil, since there are no terms fulfilling this condition. However, this step is important once we deal with approximated data, for examples when rounding of floating point numbers has been carried out. The bottom line is, that we may avoid divisions in the backwards algorithm altogether.*

4 Evaluation of the pseudo elimination

The matrix $\text{left}(L)$ i.e. the one given by the first n columns of L is by construction a diagonal matrix of the form

$$\text{left}(L) = \begin{pmatrix} \prod_{i=1}^n X_i & 0 & \cdots & \cdots & 0 \\ 0 & \prod_{i=2}^n X_i & \cdots & \cdots & 0 \\ \vdots & \ddots & \ddots & \cdots & \vdots \\ 0 & \cdots & \prod_{i=k}^n X_i & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots & \\ 0 & 0 & 0 & \cdots & X_n \end{pmatrix}. \quad (13)$$

Hence, we get for its determinant

$$\det(\text{left}(L)) = \prod_{i=1}^n X_i^i. \quad (14)$$

Going from Λ to L , some adding and subtracting of rows has been done. These do not alter the determinant. However, in addition, $n-1$ rows have been multiplied by each of the X_i , these multiplications do change the determinant. Hence by use of (14) we obtain

$$\det(\text{left}(\Lambda)) = \prod_{i=1}^n X_i^{i+1-n}. \quad (15)$$

Now, observe that going from Λ to L , an genuine elimination algorithm has been applied, which transformed

$$\begin{aligned} \text{left}(\Lambda) &\rightarrow \text{left}(L), \\ I &\rightarrow \text{right}(L), \end{aligned}$$

where I is the $n \times n$ unit matrix. Since the effect of such an algorithm can always be obtained by multiplication from the left with some suitable matrix S , we have

$$\begin{aligned} S \text{ left}(\Lambda) &= \text{left}(L), \\ S I &= \text{right}(L). \end{aligned}$$

This implies

$$S = \text{right}(L), \quad (16)$$

or

$$\text{right}(L) \text{ left}(\Lambda) = \text{left}(L). \quad (17)$$

The inverse $R = \text{left}(L)^{-1}$ of the diagonal matrix $\text{left}(L)$ is easily seen to be

$$R = \begin{pmatrix} (\prod_{i=1}^n X_i)^{-1} & 0 & \cdots & \cdots & 0 \\ 0 & (\prod_{i=2}^n X_i)^{-1} & \cdots & \cdots & 0 \\ \vdots & \ddots & \ddots & \cdots & \vdots \\ 0 & \cdots & (\prod_{i=k}^n X_i)^{-1} & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots & \\ 0 & 0 & 0 & \cdots & (X_n)^{-1} \end{pmatrix}. \quad (18)$$

From this we conclude

$$(\det(\text{left}(\Lambda)) R) \text{ left}(\Lambda) = \det(\text{left}(\Lambda)) I. \quad (19)$$

Therefore the term in the bracket on the left hand side of (19) is the matrix consisting of the minors of $\text{left}(\Lambda)$. This yields for the matrix of minors

$$\text{minors}(M) = \det(\text{left}(\Lambda)) R \text{ right}(L). \quad (20)$$

Taking into account our knowledge about R and $\det(\text{left}(\Lambda))$ we find that the matrix of minors is obtained by multiplying each line of $\text{right}(L)$ by a suitable factor, i. e. the k -th line by

$$\prod_{i=1}^n X_i^{i+1-n} \prod_{i=k}^n (X_i)^{-1}. \quad (21)$$

Or, if we define the $n \times 2n$ matrix M_{result} by the following concatenation

$$M_{\text{result}} = \left(\begin{array}{ccccc|c} 1 & 0 & \cdots & \cdots & 0 & \\ 0 & X_1 & \cdots & \cdots & 0 & \\ \vdots & \ddots & \ddots & \cdots & \vdots & \\ 0 & \cdots & \prod_{i=1}^{k-1} X_i & \cdots & 0 & \text{right}(L) \\ \vdots & \ddots & \ddots & \cdots & \vdots & \\ 0 & 0 & 0 & \cdots & \prod_{i=1}^{n-1} X_i & \end{array} \right). \quad (22)$$

Then if this matrix is multiplied by

$$\prod_{i=1}^n X_i^{i-n}, \quad (23)$$

we obtain the matrix of minors.

This again looks like a lot of divisions were to be carried out. However, by the same or similar arguments as before (Remarks 2, 3) we see that this is not the case. We only have to use the fact that the minors are sub determinants, and that for the computation of a sub determinant, no division is needed. Another line of reasoning would be that the arguments in Remark 2 carry over to our matrix of minors (only with different powers in the X_i) since they carry over to sub matrices. So back substitution in (20), by use of `red_pol`, leads to the correct results also in case of non invertible matrices and matrices with zero divisors. Again, no divisions are needed. The following algorithm computes `minors(M)`, the matrix of minors of M .

Algorithm 4 (Matrix of minors): `mod_gauss_1`

INPUT: The matrix M_{result} defined in (22), the substitution list $E\text{list}$ obtained by algorithm 1.

OUTPUT: The matrix `minors(M) := Mmin`.

START: $M_{\text{min}} := M_{\text{result}}$

For k from n **down to** 1 do

#kj $M_{\text{min}} :=$ the matrix obtained by applying `red_pol(Xk, n - k)` to the elements of M_{min} .
 $M_{\text{min}} :=$ the matrix obtained by applying the substitution for X_k given by the k -th element of $E\text{list}$.

end.

END: Return M_{min} .

To obtain only the determinant a simpler algorithm can be performed:

Algorithm 5 (Determinant): mod_gauss_2

INPUT: An $n \times n$ matrix M , the substitution list $Elist$ obtained by algorithm 1.

OUTPUT: D the determinant of M .

START: $D := \prod_{i=1}^n X_i$.

$D :=$ the expression obtained by application of the last substitution in $Elist$ to D .

For k from $n - 1$ down to 1 do

$D :=$ application of $\text{red_pol}(X_k, n - k)$ to D .

$D :=$ application of the substitution given by the k -th element of $Elist$ to D .

end.

END: Return D .

The above algorithm is obtained from the fact that for the computation of determinants no divisions are necessary. Hence the applications of red_pol are exact divisions and the result of the algorithm then obviously is equal to the determinant value given in (14).

Of course, if only the determinant shall be computed then instead of computing the minors, we should only go to an upper triangle form. In this case, since only the substitution list is needed, the evaluation of the X_i 's are the same.

Remark 4. *In this setup the problem of dealing with incorrect operations, like in matrices with floating point entries, where each operation generates rounding errors, is especially simple. This comes from the fact that when in algorithm mod_gauss_1, after expansion, negative powers of X_k occur, their coefficients can only be the result of rounding errors. Hence, dropping these terms, as it is done by red_pol , corrects these rounding errors and takes care that even then divisions or pivot changes are not necessary.*

5 Improved efficiency

If we use the pseudo-elimination algorithm in a unsophisticated way, then we observe a considerable internal data swell. This comes from the fact that red_pol deletes a lot of terms, of course only terms which cancel. So a good strategy may be, either not to generate these terms at all or even to add unnecessary

terms which will be deleted by `red_pol` anyway but which, before being deleted, do compactify the expressions.

Therefore we have a look on the difference between algorithm A and the pseudo elimination for finding out what the relation between the variables Z_k and X_k has to be. First, we leave out steps (3.1) and (4) in algorithm A, and call the resulting matrix for the moment Q . Then $Q_{k,k} = 1/Z_k$. In the algorithm leading to Q , apart from having different formal variables, the elimination steps are performed with the same rows as in the pseudo elimination algorithm, only, the multiplication of j -th row with X_k is left out when the element $M[j, k]$ is eliminated. Hence the resulting matrix in algorithm 1 differs by a factor $(\prod_{i=1}^n X_i) / X_k$ in the k -th row, $k = 1, 2, \dots, n$. Now, comparing the elements in the diagonal in the left part we find from (18)

$$1/Z_k * \left(\prod_{i=1}^n X_i \right) * 1/X_k = \prod_{i=k}^n X_i, \quad (24)$$

or

$$Z_k = \left(\prod_{i=1}^{k-1} X_i \right) / X_k. \quad (25)$$

This expresses the Z_k in terms of the X_i . Denote for the moment by M_Z the matrix coming out of algorithm A (leaving out step (3.1)). Hence, by additionally taking into account that each row k has been multiplied by Z_k we find (from the definition in (22) that

$$\left(\prod_{i=1}^n X_i \right) M_Z = M_{\text{result}}. \quad (26)$$

By almost identical arguments we obtain that M_{list_k} (k -th entry of M_{list}) multiplied by $\prod_{i=1}^{k-1} X_i$ yields the right hand side of the equation for X_k in E_{list} . Hence the k -th entry of E_{list} can be written as

$$X_k = M_{\text{list}_k} \prod_{i=1}^{k-1} X_i. \quad (27)$$

Now, we look at what the repeated substitutions and applications of `red_pol`, as required in algorithm 4 and 5, do mean in terms of the variables Z_k .

Consider therefore

$$T_k \left(P \prod_{i=1}^k X_i^{n-k+1} \right), \quad (28)$$

where the operator T_k stands for applying first $\text{red_pol}(X_k, n - k)$ and then substituting the remaining X_k by the equation for X_k in $E\text{list}$ or by (27). Observe that all evaluations in algorithms 4 and 5 are of the form

$$T_1 T_2 \cdots T_k \cdots T_n \left(P \prod_{i=1}^n X_i^{n-k+1} \right), \quad (29)$$

where P is a matrix element of M_Z as given in (26). For obtaining the result of (28) we decompose

$$P = P_0 + Z_k P_1 + Z_k^2 P_{\text{rest}},$$

where P_0 and P_1 are not containing Z_k (i.e. we take the Taylor polynomial of second order in Z_k).

The important argument now is, that $Z_k^2 P_{\text{rest}}$ is annihilated in (28) by red_pol . This follows from the fact that if the Z_k are replaced according to (25) then from $Z_k^2 P_{\text{rest}}$ only powers of X_k are generated with exponents $< n - k$, hence terms which are annihilated by $\text{red_pol}(X_k, n - k)$. Furthermore, $Z_k P_1$ is mapped by $\text{red_pol}(X_k, n - k)$ onto a term where no X_k is left for substitution. So we obtain by use of (27)

$$T_k \left(P \prod_{i=1}^k X_i^{n-k+1} \right) = \left(\hat{P} \prod_{i=1}^{k-1} X_i^{n-k+2} \right), \quad (30)$$

where

$$\hat{P} = (M\text{list}_k P_0 + P_1) . \quad (31)$$

Observe that the mapping

$$P \rightarrow \hat{P} \quad (32)$$

is exactly the evaluation step with index k as described in algorithm B. Finally, we observe that steps (3.1) do not change the result obtained after the application of the evaluation algorithm since the quadratic terms deleted in steps (3.1) are deleted anyway during the evaluation. Hence we have proved that the evaluation algorithms in 4 and 5, by introduction of the variables Z_k , carry over to the evaluation algorithm B and the claims in Result 2 are proved.

Indeed, this form of the algorithm, by deleting in products all second powers of the Z_k , reduces the data swell tremendously.

6 Remarks on complexity

The usual worst case complexity, with respect to the dimension of a matrix, seems to be the same for all kind of Gauss algorithms. An investigation of

complexity with respect to the entries of a matrix is not yet timely since we have no generally agreed measure for the complexity of symbolic data. If there were such a measure, ranging over all kind of algebraic structures, then it could very well turn out, that such a measure were not meaningful. This because real-life complexity of symbolic data depends very much upon how these data structures are internally realized.

Nevertheless such a notion of *symbolic complexity* could be helpful insofar as it would lead to some improvements. For example, in our algorithm a pivot changing at each step, by putting the entry of highest symbolic complexity on the diagonal, certainly would accelerate the algorithm and reduce intermediate data swell.

Anyhow, we believe, that for most existing algebraic structures and for all Gauss algorithms, special examples can be constructed, where one algorithm is better than the others. So the appropriate notion of complexity should be *average case complexity*, instead of worst case complexity. However, here again a specification is missing how the different structures have to be measured when contributing to the average. This especially applies to the algebraic domain for which our algorithm was developed in the first place and which is far more complex than multivariate polynomials. The domain we had in mind is the domain of rational expressions of general functions in several variables as they come up in formal solutions of differential equations. The reason why the Gauss algorithm for such applications is important comes from the fact that for dynamic systems in n -space having $n - 1$ symmetry generators, exact solutions are found by inverting the matrix formed from these generators. And this matrix usually is of a highly complex nature. The reason why this inverse yields the formal solutions of the system lies in the fact that the rows of this inverse are gradients of conserved quantities, hence give rise to constants of motions by integration.

However, examples from this area are too complicated to be presented here. Furthermore, they do not give an insight in the algorithmic structure of the modified Gauss algorithm. So we have to leave out these examples. Instead of that, we present some simple examples. Furthermore, an example giving insight into the feature that our modified Gauss algorithm is capable of correcting rounding errors.

7 Examples

An elementary implementation and **sample code** for these algorithms in MuPAD [5] is to be found on the following web page:

www.fuchssteiner.ch/mupprgms/mod_gauss.html

There however, the algorithms are not optimized for speed since that would require to replace some procedure calls by programming elements implemented in the kernel of the system³. This would make the program less transparent, hence not suitable for its demonstration purpose.

We present here further simple examples for different algebraic structures, still the examples are small in size, they are only meant in order to demonstrate, that the algebraic requirements of the algorithm are almost nil. The examples are taken from the following domains of computation:

- Integer remainder classes
- Floating point data
- Polynomials with coefficients in such classes

Example 4.

First we apply mod_gauss to the matrices in examples 2 and 3.

For the matrix in Example 2, we obtain as result for the matrix of minors:

$$\begin{pmatrix} 5 & 1 & -7 \\ 6 & 6 & -12 \\ -4 & -2 & 8 \end{pmatrix}. \quad (33)$$

For example 3, we obtain the zero matrix.

Example 5.

In order to take an example where the matrix of minors is not zero, although the matrix is not invertible, and therefore there must be zero pivots, we consider

$$\begin{pmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}, \quad (34)$$

and obtain

$$\begin{pmatrix} 1 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}. \quad \square \quad (35)$$

³The complication of this experimental implementation comes only from the fact that a wide variety of algebraic domains have to be covered.

Example 6.

We consider a matrix with entries in the remainder class modulo 8:

$$M = \begin{pmatrix} 1 \text{ mod } 8 & 2 \text{ mod } 8 & 4 \text{ mod } 8 \\ 3 \text{ mod } 8 & 2 \text{ mod } 8 & 1 \text{ mod } 8 \\ 2 \text{ mod } 8 & 0 \text{ mod } 8 & 3 \text{ mod } 8 \end{pmatrix}. \quad (36)$$

In addition to being a matrix in an algebraic structure with many zero divisors, the matrix has determinant zero. However, the algorithm does not care about these obstacles. We obtain for the matrix of minors

$$\text{minors}(M) = \begin{pmatrix} 6 \text{ mod } 8 & 2 \text{ mod } 8 & 2 \text{ mod } 8 \\ 1 \text{ mod } 8 & 3 \text{ mod } 8 & 3 \text{ mod } 8 \\ 4 \text{ mod } 8 & 4 \text{ mod } 8 & 4 \text{ mod } 8 \end{pmatrix}. \quad \square \quad (37)$$

Example 7.

Now we consider formal expressions with a coefficient domain of remainders modulo 5:

$$M = \begin{pmatrix} 2x + 1 & 0 & y \\ 3x^2 + 9x & z & 3x \\ x & 2u + 1 & 0 \end{pmatrix}. \quad (38)$$

The algorithm yields for the matrix of minors

$$\text{minors}(M) = \begin{pmatrix} 2x + 4ux & y + 2uy & 4yz \\ 3x^2 & 4xy & 2x + 3x^2y + 4xy + 4x^2 \\ 4x + ux^2 + 3ux + 4xz + 3x^2 & 3u + 3x + ux + 4 & z + 2xz \end{pmatrix}. \quad (39)$$

The same matrix, if the coefficients are considered to be remainders modulo 2, would become

$$M = \begin{pmatrix} 1 & 0 & y \\ x^2 + x & z & x \\ x & 1 & 0 \end{pmatrix}. \quad (40)$$

And its matrix of minors would then be

$$\text{minors}(M) = \begin{pmatrix} x & y & yz \\ x^2 & xy & x + x^2y + xy \\ x + xz + x^2 & 1 & z \end{pmatrix}. \quad \square \quad (41)$$

Example 8.

We consider a matrix in a numeric domain with incorrect operations, defined

by the condition that only 5 decimal digits are considered:

$$M = \begin{pmatrix} 0.00003 & 2.0 & 6.0 & 5.0 \\ 0.14286 & 0.0003 & 0.66667 & 1.0 \\ 0.66667 & 1.0 & 1.0 & 0 \\ 4000.0003 & 2.0 & 1.0 & 0 \end{pmatrix}. \quad (42)$$

Carrying out the usual Gauss algorithm and rounding after the fifth decimal at each step, we obtain the following list of pivots

$$\text{list_of_pivots} = [0.00003, -0.28572, 0.00005, 0]. \quad (43)$$

Therefore, although the determinant is not so small ($D = -2669.72466$), an unrefined fraction free Algorithm without pivot changing would yield the zero matrix. However, `mod_gauss` yields (without any pivot changing) for the matrix of minors

$$\text{minors}(M) = \begin{pmatrix} -1.0 & 5.0 & 3.3348 & -0.66815 \\ 3999.33363 & -19996.66815 & -10667.31507 & 2.49205 \\ -3998.66696 & 19993.3348 & 7995.42914 & -2.04661 \\ 2664.73436 & -15993.33453 & -5327.58896 & 1.45912 \end{pmatrix}. \quad \square \quad (44)$$

Example 9. There is another area where striking effects of our algorithmic approach can be observed, namely in the area of numerical calculations. This is insofar surprising as the algorithm has been developed for symbolic data without any expectation of its applicability in numerical calculations. Therefore, we present one example of this kind. The correction of rounding errors in this case is due to the fact that in the formal algorithm rounding leads to higher powers of the variables Z_i . Since these are thrown out systematically because they cannot occur in a correct computation without rounding these errors are corrected automatically. However, there is a tradeoff for this remarkable feature, namely that keeping track of the variables Z_i in our algorithm increases its complexity over the usual Gauss algorithm.

$$M = \begin{pmatrix} 0.00003 & 2.0 & 6.0 & 5.0 \\ 0.14286 & 0.0003 & 0.66667 & 1.0 \\ 0.66667 & 1.0 & 1.0 & 0 \\ 4000.0003 & 2.0 & 1.0 & 0 \end{pmatrix}. \quad (45)$$

We perform the computation in a domain, where at each step the numbers are cut off after the 5-th decimal. First we compute the correct matrix of minors

and the correct determinant. This is done by rewriting the entries as rational numbers for which the exact Gauss algorithm is performed. For comparison then the result is rewritten in decimal numbers. Thus we obtain for the determinant:

$$D = -2669.662719, \quad (46)$$

and the correct matrix of minors is

$$M = \begin{pmatrix} -1.0 & 5.0 & 3.3348 & -0.66815 \\ 3999.33363 & -19996.66815 & -10667.31507 & 2.492045555 \\ -3998.66696 & 19993.3348 & 7995.42914 & -2.046609995 \\ 2664.734362 & -15993.33453 & -5327.588959 & 1.459117781 \end{pmatrix}. \quad (47)$$

Performing the usual Gauss algorithm within our domain of cut-off decimals we obtain for these data

$$D = -2669.72466, \quad (48)$$

and the correct matrix of minors is

$$M = \begin{pmatrix} -0.9878 & 4.99239 & 3.33716 & -0.66743 \\ 3999.99506 & -19999.94862 & -10668.88717 & 2.69642 \\ -3999.30094 & 19996.53137 & 7996.89325 & -2.24257 \\ 2665.23952 & -15995.94897 & -5328.74372 & 1.60183 \end{pmatrix}. \quad (49)$$

For some entries the rounding error amounts up to 10 %, no wonder since the cut off is expected to have drastic consequences for numbers so far apart. However, our modified Gauss algorithm yields:

$$D = -2669.66272, \quad (50)$$

and for the matrix of of minors we obtain

$$M = \begin{pmatrix} -1.0 & 5.0 & 3.3348 & -0.66815 \\ 3999.33363 & -19996.66815 & -10667.31507 & 2.49205 \\ -3998.66696 & 19993.3348 & 7995.42914 & -2.04661 \\ 2664.73436 & -15993.33453 & -5327.58896 & 1.45912 \end{pmatrix}. \quad (51)$$

These results are, although all computation steps were done in the domain of cut-off decimals, are remarkably correct, they differ from the correct results only by rounding in the fifth decimal.

A systematic study of this phenomenon, together wit a purely numerical formulation (i.e. relying on variables) of the algorithm will be done in the near future

8 Concluding remark

Another intriguing aspect is that in certain cases it is advantageous, from the point of view of memory space and speed of use, to store these matrices not as symbolic data but in a form similar to the one given in (10) together with its substitution list⁴. This kind of storage also could be extended to the original matrix, such that matrix multiplication and the like are done by symbolic matrices like (10) together with a change of the rules for the substitution list. We leave these aspects to further research.

References

- [1] A. G. Akritas and G. I. Malaschonok: *Computations in Modules over Commutative Domains*, in: Lecture Notes in Computer Science, Verlag Springer Berlin / Heidelberg, 4770, p.11-23, 2007
- [2] E. H. Bareiss: *Sylvester's Identity and Multistep Integer-Preserving Gaussian Elimination*, Mathematics of Computation, 22, p.565-578, 1968
- [3] B. Fuchssteiner: *An extension of the Liouville-Arnold theorem to the non-Hamiltonian case*, preprint, 2007
- [4] B. Fuchssteiner and K. Gehrs: *Constellation Problems and Integrability for linear ODE's with constant Coefficients*, preprint, 2007
- [5] J. Gerhard, W. Oevel, F. Postel and S. Wehmeier: *MuPAD Tutorial*, Springer Verlag, Berlin-Heidelberg-New York, 2000
- [6] M. Matooane: *Parallel systems in symbolic and algebraic computation*, Technical Report No. 537, University of Cambridge, UCAM-CL-TR-537, ISSN 1476-2986, p.139, 2002
www.cl.cam.ac.uk/techreports/UCAM-CL-TR-537.pdf
- [7] S. Moritsugu and K. Kuriyama: *Fraction-free Method for Computing Rational Normal Forms of Square Matrices, in Japanese*, Trans.Japan Soc.Indust.Appl.Math., 6, p.253-264, 1998
In english: <http://citeseer.ist.psu.edu/context/436939/62659>

⁴Then however a slight change of the algorithm is advisable, a change such that the two inner loops in *mod_gauss* reduce to one inner loop.

- [8] T. MULDER: *A generalized Sylvester identity and fraction-free random Gaussian elimination*, Technical Report No. 272 of the Department of Computer Science, ETH Zürich, p.13, 1997
- [9] G. C. Nakos, P. R. Turner and R. M. Williams: *Fraction-free algorithms for linear and polynomial equations*, ACM SIGSAM Bulletin, 31, p.11-19, 1997
- [10] T. Sasaki and H. Muraio: *Efficient Gaussian Elimination Method for Symbolic Determinants and Linear Systems*, ACM Transactions on Mathematic Software 277-289, 1982, 8, Number 3, p.277-289, 1982
- [11] T. Sasaki and T. Sato: *Cancellation Errors in Multivariate Resultant Computation with Floating-point Numbers*, ACM SIGSAM Bulletin archive, 32, p.13-20, 1998
- [12] J. Smit: *A cancellation free algorithm, with factoring capabilities, for the efficient solution of large sets of equations*, In: Paul S.Wang, editor, Proceedings of the 1981 ACM Symposium on Symbolic and Algebraic Computation, p.146-154, 1981
- [13] Y. Umeda and T. Sasaki: *Computing Determinants of Rational Functions*, ACM SIGSAM Bulletin, 40, p.7, 2006