Regularization of linear recurrence systems[*]

S.A. Abramov[†] M. Bronstein[‡] D.E. Khmelnov[§]

May 28, 2003

Abstract

We consider the problem of transforming a recurrence system

$$P_d(n)z_{n+d} + P_{d-1}(n)z_{n+d-1} + \cdots + P_0(n)z_n = 0$$

where the $P_i(n)$ are polynomial $m \times m$-matrices, into forms where the matrix $P_d(n)$ (resp. $P_0(n)$) is nonsingular. As a basic auxiliary transformation we use a reduction of the system: whenever the leading (resp. trailing) matrix is singular, then such a reduction ensures that a zero row or column appears in it. We consider different algorithms based on different types of reduction, and show that our EG' reduction [2, 3] is the only two-stage reduction. We also present extensive experimental comparisons of EG' with other known reductions, comparisons that confirm the practical superiority of EG' over one-step reductions.

Аннотация

Рассматривается задача приведения рекуррентной системы

$$P_d(n)z_{n+d} + P_{d-1}(n)z_{n+d-1} + \cdots + P_0(n)z_n = 0,$$

где все $P_i(n)$ — полиномиальные $m\times m$-матрицы, к виду, в котором матрица $P_d(n)$ (соответственно, $P_0(n)$) является невырожденной. В качестве основного вспомогательного преобразования выступает редукция системы: если ведущая (соответственно, трейлинговая) матрица вырождена, то редукция обеспечивает появление в ней нулевой строки или столбца. Мы рассматриваем алгоритмы, основанные на различных видах редукции, и показываем, что из них только предложенная авторами EG' редукция [2, 3] является двухэтапной редукцией. Мы также приводим экспериментальное сравнение EG' с другими известными редукциями; эти сравнения подтверждают практическое превосходство EG' над одноэтапными редукциями.

Résumé

Nous considèrons le problème de transformer un système de récurrences linéaires

$$P_d(n)z_{n+d} + P_{d-1}(n)z_{n+d-1} + \cdots + P_0(n)z_n = 0$$

où les $P_i(n)$ sont des matrices $m \times m$ de polynômes, vers des formes où la matrice $P_d(n)$ (resp. $P_0(n)$) est non singulière. La transformation auxiliaire de base que nous utilisons est la réduction du système : chaque fois que la matrice de tête (resp. de queue) est singulière, une telle réduction garantit qu'une ligne ou colonne nulle y apparait. Nous considérons plusieurs algorithmes basés sur des réductions différentes et montrons que notre réduction EG' [2, 3] est la seule réduction à deux stages. Nous présentons aussi des comparaisons expérimentales étendues entre EG' et les autres réductions connues, comparaisons qui confirment la supériorité en pratique de EG' sur les réductions à un stage.

[*]Supported in part by a grant from French-Russian Lyapunov institute, Project No 98-03.

[†]Dorodnicyn Computer Center of the Russian Academy of Sciences, Moscow 119991, Russia, `abramov@ccas.ru`

[‡]INRIA, BP 93, 06902-Sophia Antipolis Cedex, France, `Manuel.Bronstein@sophia.inria.fr`

[§]Dorodnicyn Computer Center of the Russian Academy of Sciences, Moscow 119991, Russia, `dennis_khmelnov@mail.ru`

1 Introduction

Linear recurrences (scalar or system) with variable coefficients appear in many areas of mathematics. Solving systems lead however to specific difficulties which do appear in the scalar case. Consider the scalar equation

$$P_d(n)z_{n+d} + P_{d-1}(n)z_{n+d-1} + \cdots + P_0(n)z_n = 0 \tag{1}$$

where the coefficients $P_0(n), \ldots, P_d(n)$ are polynomial, and $P_0(n), P_d(n)$ are not identically zero. Then those two polynomials vanish only for a finite set of values of n. If (1) is obtained, for example, as a relation satisfied by the coefficients of a series that satisfies a given linear differential equation then the roots of $P_0(n)$ and $P_d(n)$ give important informations on the solutions of that differential equation. If (1) is instead a system, z_n is an m-component vector, and $P_0(n), \ldots, P_d(n)$ are polynomial $m \times m$-matrices then the role, which is played by the roots of the leading and trailing polynomials in the scalar case, can now be played by the roots of the determinants of the matrices $P_0(n)$ and $P_d(n)$, provided that those determinants are not identically zero.

In a similar fashion, linear recurrence systems can appear when handling other types of linear functional systems, for example, difference and q-difference systems with polynomial coefficients. Such functional systems induce recurrence systems of the form (1) for the coefficients of their series solutions, and the roots of the determinants of the matrices $P_0(n)$ and $P_d(n)$, (when those matrices are non-singular) are always important for determining the structure of the solution space.

It may happen however that the matrix $P_0(n)$ or $P_d(n)$ is singular. In that case, not only is it impossible to compute bounds on the orders of the solutions, but it also makes difficult, from a computational standpoint, to use the recurrence (1) to compute the sequence of vectors it generates given initial conditions.

A natural solution in that case is to compute an equivalence transformation of the reccurence system, which transforms it into a form with either the leading or trailing matrix nonsingular. This transformation may be a "quasi–equivalence", in the sense that the eventual changes in the solution set can be easily taken into account.

In this paper we consider algorithms using that equivalence approach and compare our algorithms EG [1, 5] and EG' [2, 3] with the algorithm `FFreduce` [11].

Since our regularizing algorithms are at the core into algorithms for computing the rational solutions of linear systems of difference equations with polynomial coefficients, we also carry out an experimental comparison of EG' with a solver based on a different approach [6, 7].

It should be mentioned that the algorithms EG, EG' and `FFreduce` are applicable not only to systems of recurrences, but more generally to rectangular matrices of generalized difference operators [9, 13, 14, 15], so they can also be applied to ordinary differential, difference and q-difference systems. In addition, the EG' algorithm is applicable to operators over noncommutative coefficient domains [3], so it can be used for example to compute the rank of matrices over multivariate Weyl algebras. However, we limit ourselves in this paper to the case of ordinary recurrence systems, first because solving arbitrary linear functional systems with polynomial coefficients is always reduced to that case, and second, because all the basic ideas of the algorithms can be seen on that case, the more general ones needing additional machinery.

In the course of project 98–03, work on the EG algorithm and its computer implementation was carried out during 1998–2000, while the development, improvement and implementation of the EG' algorithm (package `LinearFunctionalSystems`) were carried out during 2001–2003.

2 The general paradigm: alternating "reduction + shift" steps

Let the coefficients of all the polynomial and rational functions be in a field K of characteristic 0. It will be convenient to consider systems of recurrences in the form

$$P_l(n)z_{n+l} + P_{l-1}(n)z_{n+l-1} + \cdots + P_t(n)z_{n+t} = 0, \tag{2}$$

where $l \geq t$ are arbitrary integers, $z = (z^1, \ldots, z^m)^T$ is a column vector of unknown sequences (such that $z_i = (z_i^1, \ldots, z_i^m)^T$) and $P_l(n), \ldots, P_t(n) \in \mathrm{Mat}_m(K[n])$ are square $m \times m$ matrices with entries in $K[n]$. The nonzero matrices $P_l(n)$ and $P_t(n)$ are called then *leading* and *trailing* matrices of the system (2). In many cases, it is natural to consider the system (2) together with a finite set of *linear constraints*, i.e. linear relations, each of which contains a finite set of variables z_i^j. Let S and S' be systems of the form (2) and C and C' be finite sets of linear constraints. We then say that the systems (S, C) and (S', C') are equivalent if the space of solutions of S that satisfy C is the same than the space of solutions of S' that satisfy C'. Note that each system of the form (2) can be considered as an infinite set of linear constraints induced by the system when substituting integer values of n into its equations. Finitely many linear constraints are easily taken into account when determining various properties of a system and when computing its solutions. We study in this paper the problems of *regularization* of the leading or trailing matrix of the system S, *i.e.* computing a system (S', C') equivalent to $(S, \emptyset)$ and such that the leading or trailing matrix of S' is nonsingular.

We compare several regularization algorithms that all fit within the "reduction + shift" general scheme, which we describe in this section. Let E denote the shift operator: $Ez_n = z_{n+1}$ for any sequence z. Then, the system (2) can be rewritten as

$$P_l(n)E^l z + P_{l-1}(n)E^{l-1}z + \cdots + P_t(n)E^t z = 0$$

and the matrix

$$P(n) = (P_l(n)|P_{l-1}(n)|\ldots|P_t(n)) \tag{3}$$

is called the *explicit* matrix of the system, with each matrix $P_i(n)$ called a *block* of the system.

For the sake of simplicity we assume here that all the equations in our system are *independent*, *i.e.* no equation can be represented as a linear combination of the others with coefficients in $K(n)[E]$. The regularization algorithms that we consider are in fact able to recognize dependence of the equations (and even to compute the rank of the system, see [3]) but we skip that aspect in this presentation. Similarly, the algorithms can also be applied to inhomogeneous systems [2] but for the sake of simplicity we consider only the homogeneous case here.

The general scheme is then the following: if the leading (resp. trailing) matrix is singular, then we left-multiply it by another matrix (obtained for example by elimination, but not necessarily so) in order to zero one of its rows. This stage is called a *reduction* of the block. Suppose that the i-th row of the block is now zero. Then, we shift the i-th row of the transformed explicit matrix, which corresponds to left-multiplication of the i-th equation of the system (2) by E (resp. E^{-1}) after the reduction step (so along with shifting the i-th row, we replace n by $n+1$ (resp. $n-1$) in that row).

Note that the reduction step can generate a set of linear constraints because of multiplications of the transformed rows by polynomials having integer roots.

If we ensure that a series of a series of "reduction + shift" steps terminate (which requires some precaution) then we obtain an equivalent system with a non-singular leading (resp. trailing) matrix. We show now how to ensure termination, and need for that to define the notions of *l-width* and *t-width* of rows. If the i-th row of the explicit matrix $P(n)$ is zero, then its l-width and t-width are both equal to 0. Otherwise, let in (3) the i-th rows of all the matrices $P_l(n), P_{l-1}(n), \ldots, P_{s+1}(n)$ be zero, while the i-th row of $P_s(n)$ is nonzero. Then $s-t+1$ is the t-width of the i-th row of $P(n)$. Similarly, let the i-th rows of $P_t(n), P_{t+1}(n), \ldots, P_{s-1}(n)$ be zero, while the i-th row of $P_s(n)$ is nonzero. Then $l-s+1$ is the l-width of the i-th row of $P(n)$. When we write only *width*, we mean the l-width if we are making the leading matrix nonsingular, the t-width if we are making the trailing matrix nonsingular.

To guarantee termination (without infinite loops) of a series of "reduction + shift" steps, it is sufficient that each reduction does not increase the width of any row: then the sum of all the widths decreases after the shift. There are different reduction algorithms that ensure that no width is increased.

One can also consider a reduction that makes a zero column appear in the leading or trailing matrix: suppose for example that the i-th column of the trailing matrix $P_t(n)$ is zero. Then we replace it with the i-th column of $P_{t+1}(n)$, which is replaced in turn with the i-th column of $P_{t+2}(n)$ and so on, the i-th column of the leading matrix $P_l(n)$ being zeroed. This corresponds to making the following substitution in the i-th component of the unknown vector z_n: $z_n^i = E^{-1}\tilde{z}_n^i$ (when making the leading matrix nonsingular, the corresponding substitution is of the form $z_n^i = E\tilde{z}_n^i$). Column–reduction is therefore connected with substitutions of the unknowns that are then taken into account further in the solving algorithms. As with row–reduction, specific precautions must be taken to ensure termination.

In both ways, reduction is connected to some sort of elimination (since it zeroes a row or column), so it causes growth in the coefficients of the matrix. It is this growth of the degrees of polynomial entries that is the main computational problem induced by multiple "reduction + shift" steps.

Further in this paper, we dwell on three algorithms that fit the above general scheme. Two of them — *EG* [1, 5] and `FFreduce` [11] — use a *one-stage* reduction, where elimination is performed on the whole explicit matrix of the system. The third algorithm — EG' [2, 3] — uses a *two-stage* reduction, where elimination (or an elimination-free alternative) is performed only in the leading or trailing matrix, and appropriate changes are made in the explicit matrix only in a second stage. Our package `LinearFunctionalSystems` includes a procedure for regularizing recurrence systems that is based on the two-stage reduction and shift. Comparisons of that package with programs implementing the algorithms *EG* and `FFreduce` clearly show the practical superiority of two-stage reduction over one-stage reductions.

Note that the algorithms EG and EG' use row–reduction, using the above rule on width to ensure termination, while `FFreduce` uses column–reduction and its own rules to ensure termination.

3 One-stage reductions

3.1 The EG algorithm

3.1.1 Outline of the algorithm

In order to determine if the leading or trailing matrix is singular, we try to transform it into triangular form with non-zero main diagonal elements by applying Gaussian elimination to the *explicit matrix*, whose rows can be multiplied by polynomials during the elimination process. If that goal is achieved in the process then the leading or trailing matrix is non-singular. Otherwise, we obtain a zero row in that block, row that yields a reduction step. After the shift that follows the reduction, we resume our Gaussian elimination, taking into account that a part of the matrix is already in trapezoidal form, and continue. During Gaussian elimination, a row having an element to be eliminated must have a width that is greater or equal to the width of the eliminating row, and we swap rows as needed to enforce that rule. When the elimination of an element of a row is accompanied by multiplication of that row by a polynomial having integer roots, substituting those roots for n in the system yields a finite set of new linear constraints.

3.1.2 Growth of the entries of the explicit matrix

Applying the EG algorithm leads to a noticeable growth of the degree and coefficients of the polynomials in the explicit matrix. In fact, let d_i be the maximal degree of the elements in the i-th row and d_j be the maximal degree of the elements in the j-th row, and let elimination be performed in the i-th row using the elements of the j-th row. Then, the degree of the elements in the resulting row may reach $d_i + d_j$. In order to perform reduction of the leading or trailing matrix, the number of such elimination steps may reach $m - 1$ where m is the total number rows. Since reduction is conducted by row operations on the whole explicit matrix, and since the width of the explicit matrix may be much greater than m, EG can lead to a huge computational work.

3.1.3 Heuristic pivot selection

As mentioned above, the EG algorithm restricts the order of the elimination steps as it takes into account the width of the rows in the choice of pivot, thereby preventing the use of classical pivot selection heuristics. Our restriction can still leave choices for the pivot, so we can adapt those heuristics to EG as follows.

At every step of the reduction, we have some subsets of rows and columns of the matrix, to which the elimination process will be applied. We can choose any column from the subset of columns for elimination, and the corresponding row from the subset of rows must be chosen while observing the width rule (there can be more than one suitable row for each column). In such a way we get a set of candidates for the column in which elimination will take place, and for the eliminating row, *i.e.* a set of candidates for the current pivot. For the final selection of the pivot we have found that the the following heuristic performs well under experiments:

1. For the considered i-th column, we select the j-th row if it observes the width rule, and has the least degree of the i-th element among such rows. In such a way we select the candidate for the pivot for the i-th column. Such candidates are selected for all columns which have not been yet used for elimination on the previous steps.

2. From those candidates we select those that require the least number of eliminations (*i.e.* the number of zeroes among the elements to be eliminated is maximal). This further restricts the set of candidates.

3. From those candidates we select the pivot to be the one of least degree.

That heuristic helps slowing the growth of elements of the explicit matrix through reducing the number of elimination steps to be performed in the explicit matrix.

3.1.4 Adapting the Bareiss algorithm

One of the standard means of controlling the growth of the entries of a polynomial matrix during Gaussian elimination is the Bareiss algorithm [8]. That algorithm eliminates common polynomial factors from the resulting row after cross multiplication and addition of rows. In addition, those common factors do not need to be computed by gcd's, but are predicted by the algorithm. But that algorithm requires that elimination at each step be performed in the whole part of the matrix that is not yet in trapezoidal form, including rows that do not need to be eliminated since they already have zeroes in the appropriate places. In particular, these rows are multiplied by the current pivot. In the context of the EG algorithm, this means unnecessary multiplications of the rows of the whole explicit matrix are performed. As the width of the rows of the explicit matrix may be much larger than its number of rows, this makes using the Bareiss algorithm a disadvantage for EG. We demonstrate that behavior with a simple example: let the explicit matrix be

$$\begin{bmatrix} 1 & 1 & 1 & 1000 & 1 & 1 \\ 1 & 1 & 1 & 1000 & 2 & 2 \\ 1 & 1 & 1 & 0 & 1 & 2 \end{bmatrix}.$$

Transforming the trailing matrix into triangular form by means of ordinary Gaussian elimination yields the steps:

$$\begin{bmatrix} 1 & 1 & 1 & 1000 & 1 & 1 \\ 1000 & 1000 & 1000 & 0 & 1000 & 1000 \\ 1 & 1 & 1 & 0 & 1 & 2 \end{bmatrix},$$

$$\begin{bmatrix} 1 & 1 & 1 & 1000 & 1 & 1 \\ 1000 & 1000 & 1000 & 0 & 1000 & 1000 \\ 0 & 0 & 0 & 0 & 0 & 1000 \end{bmatrix}.$$

While using the Bareiss algorithm for the same purpose yields the steps:

$$\begin{bmatrix} 1 & 1 & 1 & 1000 & 1 & 1 \\ 1000 & 1000 & 1000 & 0 & 1000 & 1000 \\ 1000 & 1000 & 1000 & 0 & 1000 & 2000 \end{bmatrix},$$

$$\begin{bmatrix} 1 & 1 & 1 & 1000 & 1 & 1 \\ 1000 & 1000 & 1000 & 0 & 1000 & 1000 \\ 0 & 0 & 0 & 0 & 0 & 1000 \end{bmatrix}$$

where on the last step we have divided the calculated result 1000000 by 1000 due to the use of the Bareiss algorithm.

In that example, the results are the same for both types of elimination, but the intermediate steps in the Bareiss algorithm produce larger entries and superfluous operations have been required. An explicit matrix may contain a big number of blocks besides its leading or trailing matrix, and the losses due to the intermediate steps and superfluous operations can overcome the gains produced by dividing by known factors.

A modification of the Bareiss algorithm allows us to take into account those "unplanned" zeroes in the matrix: we eliminate a common factor in the i-th after one of its entries has been eliminated by the k-th row, only if the last eliminations in both of those rows have been performed by the same (say m-th) row. The common factor is then the pivot of that m-th row, so we need to keep some additional information about the previous elimination step in that row along with the rows of the explicit matrix.

Experiments with our implementation of the EG algorithm has shown that the use of the above modification gives some gain in comparison with plain EG, but that the use of the ordinary Bareiss algorithm yields a substantial efficiency loss.

3.2 The `FFreduce` algorithm

Although `FFreduce` [11] has a different theoretical foundation, its execution can nevertheless be described in terms of alternating reduction and shift steps. Using that paradigm, regularization of the leading (resp. trailing) matrix of a recurrence system with `FFreduce` can be outlined as follows:

- For each column of the leading (resp. trailing) matrix, repeat:

(a) Select a pivot in the current column using some minimization rule, and compute the coefficients to be used at step (c).

(b) Use the selected pivot to eliminate entries in the selected column, dividing by the pivot of the previous elimination step (*i.e.* using the ordinary Bareiss algorithm, as described in Section 3.1.4).

(c) Shift the eliminating row in the direction opposite to the direction in the algorithm EG. There appears then both a zero row (the shifted one) and a zero column (the selected one) in the leading (resp. trailing) matrix. Substract from the shifted row the linear combination of the other rows with the coefficients computed at step (a) and divide out once more by the previous pivot. The zero column remains, so perform a reduction step with respect to that column.

(d) Perform a vertical shift into the zero column.

- After repeating the above loop for all the columns, stop if the system is in the desired form, repeat the loop otherwise. It is shown in [11] that this algorithm always terminate.

From the above description, it is clear that `FFreduce` is based on a one-stage reduction, since all its eliminating operations are performed in the whole explicit matrix. Therefore, the difficulties mentioned about EG, difficulties connected with the large number of operations and the growth of the entries of the explicit matrix under one-stage reduction, are expected to appear in `FFreduce` as well. The proposed control over the growth of the entries in `FFreduce` is a variation of the Bareiss algorithm, but, as shown in Section 3.1.4, a straightforward use of the Bareiss algorithm within one-stage elimination can cause additional losses of efficiency. Since in addition, the computation of the linear combination of step (c) leads to additional costs not present in EG, we can suppose that `FFreduce` should underperform EG (despite being published three years after EG). Of course, since the mentioned difficulties only appear when the number of rows is large enough, this analysis is not valid for systems of very small size. Nevertheless, our experiments (see Section 6.1) show the practical superiority of EG over `FFreduce` starting at systems with 5 rows. In any event, experiments and theoretical analysis indicate that EG and `FFreduce` are both outperformed by the two-stage reduction of EG', which we describe next.

4 The EG' algorithm

4.1 Description of the two-stage reduction

Within the general framework described above, the essential computational part of the reduction performed by EG' is done only inside the leading or trailing matrix (rather than in the extended matrix). We use any available method to check whether the rows of the leading or trailing matrix are linearly dependent over $K(n)$, and if they are, to find the coefficients $v_1(n), \dots, v_m(n) \in K[n]$ of a dependence. Once such a dependence is found (the leading or trailing matrix is nonsingular otherwise) we select from the nonzero coefficients of the dependence one, say $v_i(n)$, that corresponds to a row of the maximal width (among the rows corresponding to nonzero coefficients). We then replace the i-th row of the whole explicit matrix by the linear combination of all its rows with the coefficients $v_1(n), \dots, v_m(n)$ (new linear constraints appear whenever $v_i(n)$ has integer roots). As a result, the i-th row of the leading or trailing matrix becomes zero (*i.e.* reduction is performed) and no single row has increased its width. We finally shift the i-th row and continue this process until the desired matrix is non-singular.

Searching for the linear dependence $(v_1(n), \dots, v_m(n))$ is equivalent to solving a homogeneous system of linear algebraic equations with polynomial coefficients. This problem is efficiently solved by many different computer linear algebra algorithms, in particular, with modular algorithms that resist well to intermediate coefficient growth of (see for example [12]). If we obtain s linearly independent solutions of the linear algebraic system, then it is possible to use all of them for reductions, which yields s zero rows in the leading or trailing matrix. To do that, we first represent the s dependences as rows of an $s \times m$ matrix V, and use the first row of V to zero the i-th row of the leading or trailing matrix, and perform the shift in the extended matrix. We then transform V by eliminating the i-th element in its rows 2 through s, using the i-th element of the first row as pivot. After this elimination, each remaining row of V contains the coefficients of a linear dependence of the rows $1, \dots, i-1, i+1, \dots, m$ of the leading or trailing matrix, so we can continue the reduction process with them, until we have performed s "reduction + shift" steps. Note that the order in which we use the rows V is in fact arbitrary, so different heuristic strategies can be used to choose the dependence at each step.

Table 1 describes the trailing version of EG' in pseudocode, and can be easily adapted to the leading version.

Table 1: Pseudocode for EG'

```
ALGORITHM EG' (trailing version)
INPUT: P — explicit $m \times dm$ matrix to be transformed
OUTPUT: P' — transformed explicit matrix with nonsingular trailing block, B — set of linear constraints
initialization: Set $P' := P$; $B := \emptyset$
while rank $P'_t < m$ do
  Set $W := I_m$ (the identity matrix of order $m$).
  Construct any $s \times m$-matrix $V$ of rank $s > 0$, such that $VP'_t = 0$.
  Set $A := \{1, \ldots, s\}$.
  while $A \neq \emptyset$ do
    • Choose a row of $V$ with index $k \in A$ and Set $A := A \setminus \{k\}$.
    • Choose a nonzero entry $v_{ki}$ of that row such that the $i$-th row of $P'$ has maximal width (among the
      rows of $P'$ corresponding to nonzero entries of the $k$-th row of $V$).
    • Using $v_{ki}$ as pivot, eliminate all the elements $v_{ji}$ of $V$, such that $j \in A$.
    • For each integer $n_0$ such that $v_{ki}(n_0) = 0$, add to $B$ the result of replacing $n$ by $n_0$ in the $i$-th recurrence
      of (2).
    • Replace the $i$-th row of $W$ by the $k$-th row of $V$.
  end while
  Set $P' := WP'$.
  for each $k$ such that the $k$-th row of $P'_t$ is zero do
    Shift the $k$-th row of $P'$ to the right and replace $n$ by $n - 1$ in it
  end for
end while
return $P', B$;
```

We now illustrate the EG' algorithm on an example. Let the explicit matrix P be

$$\begin{bmatrix} 0 & 0 & n-2 & 0 & n-1 & n-1 & n & n & n \\ 0 & 0 & 0 & 0 & 0 & 0 & n & n & n \\ 0 & 0 & 0 & 0 & n-2 & n-2 & n & n & n \end{bmatrix}$$

whose trailing matrix P_t is

$$\begin{bmatrix} n & n & n \\ n & n & n \\ n & n & n \end{bmatrix}.$$

Since the rank of P_t is 1, let W be the identity matrix of order 3, and V be any matrix such that $VP_t = 0$, for example

$$V = \begin{bmatrix} 1 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix}.$$

Choosing the first row of V, we must select the element $v_{1,1} = 1$ since the first row of P has a larger width than its third row. Eliminating the first column of V using $v_{1,1}$ yields

$$V = \begin{bmatrix} 1 & 0 & -1 \\ 0 & -1 & 1 \end{bmatrix}.$$

Since the equation $v_{1,1}(n) = 0$ has no roots, no new linear constraint appears at this step of the reduction.

Using then the second row of V, we must select the element $v_{2,3} = 1$ since the third row of P has a larger width than its second row. Again, no new linear constraint appears, and W is now

$$W = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix}.$$

Multiplying P by W yields

$$\begin{bmatrix} 0 & 0 & n-2 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & n & n & n \\ 0 & 0 & 0 & 0 & n-2 & n-2 & 0 & 0 & 0 \end{bmatrix},$$

and shifting its first and third rows to the right finally yields

$$P' = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & n-3 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & n & n & n \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & n-3 & n-3 \end{bmatrix}, \tag{4}$$

completing the first "reduction + shift" step. Our new trailing matrix is

$$P'_t = \begin{bmatrix} 0 & 1 & 1 \\ n & n & n \\ 0 & n-3 & n-3 \end{bmatrix}$$

whose rank is 2. Choose anew a matrix V such that $VP'_t = 0$, for example

$$V = \begin{bmatrix} n-3 & 0 & -1 \end{bmatrix}.$$

We must select the element $v_{1,1} = n-3$ since the first row of P' has a larger width than its third row. Since the equation $v_{1,1}(n) = 0$ has the root $n_0 = 3$, a new linear constraint appears by replacing n by 3 into the first row of the current explicit matrix P' (4):

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}. \tag{5}$$

That constraint translates into the condition $z_t^2 + z_t^1 = 0$. Our matrix W is now

$$\begin{bmatrix} n-3 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and multiplying P' by W yields

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & (n-3)^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & n & n & n \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & n-3 & n-3 \end{bmatrix}.$$

Finally, shifting its first row to the right yields

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (n-4)^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & n & n & n \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & n-3 & n-3 \end{bmatrix}$$

completing the second "reduction + shift" step. Our new trailing matrix is

$$\begin{bmatrix} 0 & 0 & (n-4)^2 \\ n & n & n \\ 0 & n-3 & n-3 \end{bmatrix} \tag{6}$$

whose rank is 3, so we are done. Note that determinant of the final trailing matrix is $n(n-3)(n-4)^2$, whose roots are $0, 3$ and 4. But if we take into account the linear constraint (5) for $n = 3$ and add the row

$$\begin{bmatrix} 0 & 1 & 1 \end{bmatrix}$$

to the trailing matrix (6) evaluated at $n = 3$, we get the matrix

$$\begin{bmatrix} 0 & 0 & 1 \\ 3 & 3 & 3 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

whose rank of 3 is maximal, so $n = 3$ can be excluded from the set of roots, for example if the initial recurrence system is being used to bound the degree of the polynomial solutions of another linear functional system.

4.2 Cost of the reduction steps

The first stage (computation of the linear dependencies of the rows of the leading or trailing matrix) consists in solving a system of linear algebraic equations of size equal to the number of rows of the recurrence system, and then in converting the matrix containing the dependences into trapezoidal form. There are efficient polynomial-time algorithms for those problems.

Computation in the whole explicit matrix during the second step is minimized since using each linear dependence leads to changing only one row of the whole explicit matrix, while the other rows are kept in their original form.

5 The superirreducibility approach to solving linear systems

As mentioned earlier, the problem of regularizing a linear recurrence system can appear in the course of solving systems of linear differential, difference or q-difference equations with polynomial coefficients. If the original system induces the system of the linear recurrences (2) and $\det P_t(n)$ is not indentically zero, then the maximal integer root of the equation

$$\det P_t(n-t) = 0$$

is an upper bound of the degree of the polynomial solutions of the original system. Similarly, the determinant of the leading matrix gives information about the order of the rational solutions at $x = 0$.

There is an alternative approach for looking for polynomial, rational and series solutions of such systems, that does not require the construction of an associated system of linear recurrences. Instead, a gauge transformation of the original system into a so-called *simple* form is computed. Once the system is in a simple form (in that special sense) then a bound of the degree of the polynomial solutions, and then the solutions themselves can be computed directly. That method is applicable only to first-order systems.

We outline that method here only in the case of polynomial solutions of difference systems of the form

$$\delta y(x) - M(x)y(x) = f(x)\,, \tag{7}$$

where $\delta y(x) = x(y(x+1) - y(x))$, $M(x) \in \mathrm{Mat}_m(K(x))$ and $f(x) \in K(x)^m$ (any linear difference system can be easily transformed into that form). Let m_i be equal to the minimum of the differences between the degree of the numerator and denominator of every nonzero element of the i-th row of the matrix $M(x)$, $\alpha_i = -\min\{m_i, 0\}$, and

$$D(x) = \mathrm{diag}(x^{\alpha_1}, \ldots, x^{\alpha_n})\,.$$

Multiplying (7) by $D(x)$ on the left yields

$$\mathcal{L}(y) = D(x)\delta y(x) - A(x)y(x) = b(x)\,, \tag{8}$$

where $A = DM$ and $b = Df$. Due to our choice of $\alpha_1, \ldots, \alpha_n$, the elements of the matrices D and A are polynomial in $1/x$, so let

$$D_0 = D(\infty) \quad \text{and} \quad A_0 = A(\infty)\,.$$

One can show that for any integer n and $c \in K$

$$\mathcal{L}(cx^n) = -x^n((-n\ D_0 + A_0)c + O(x^{-1}))\,. \tag{9}$$

Define the system (7) to be simple if $\det(A_0 - nD_0)$ is not identically zero. If the system (7) is simple, then the integer roots of that determinant and the maximal degree of the polynomials in $f(x)$ yield an upper bound for the degree of its polynomial solutions.

The super-irreducible form, introduced in the differential case by A. Hilali and A. Wazner in [10] was generalized by M. Barkatou to difference systems [6, 7]. That form provides a nonsingular matrix $T(x) \in \mathrm{Mat}_m(K[x])$ such that the substitution $y(x) = T(x)\tilde{y}(x)$ transforms the system (7) into a simple form. Although Barkatou has shown that super-irreducibility of a system implies simplicity, the opposite does not always hold. An algorithm based on super-irreducibility for computing the polynomial and rational solutions of linear difference systems was presented in [4].

6 Experimental comparisons

6.1 "reduction + shift" algorithms

We present in this section the results of comparing comparing programs implementing the algorithms EG, Smart EG, EG' and `FFreduce`. Smart EG is the version of the EG improved by the heuristic of Section 3.1.3 and our adaptation of the Bareiss algorithm (see the end of Section 3.1.4).

We applied them all to the same collection of 100 matrices, which are the explicit matrices of recurrence systems induced by randomly generated systems of linear difference equations, of random sizes ranging from 5 to 15. The entire collection of matrices (as well as for the other comparisons reported here) is available at the URL `http://www.ccas.ru/~zavar/abrsa/eg_prime/comparisons.html`.

The results are presented in Table 2, whose rows correspond to the algorithms. The total time (in CPU seconds) for the regularization of all the 100 matrices by the corresponding algorithm is in the column "Total Time", while the following cells indicate the number of examples on which the algorithm corresponding to the row was faster than the algorithm corresponding to the column (as in a tournament table). For example, Smart EG was faster than EG on 89 examples out of 100.

Table 2: Comparison of various reduction algorithms

	Total Time	EG'	**Smart** EG	EG	`FFreduce`
EG'	5.011	-	97	99	100
Smart EG	16.538	3	-	89	100
EG	93.725	1	11	-	100
`FFreduce`	3019.251	0	0	0	-

Overall, EG' had the best performance and `FFreduce` the worst. While the above rankings can change on very small systems that require a very small amount of work, we found that for larger systems, algorithms from the top part of the table can outperform the ones from the bottom part by several orders of magnitude.

6.2 Linear difference system solvers

As mentioned earlier, one of the applications of the regularization algorithms is computing the polynomial and rational solutions of systems of linear functional equations, in particular, difference equations. We therefore present in this section the results of comparing our solver in the package `LinearFunctionalSystems`, based on EG', to the solver in the SIF program, based on transforming the system into super-reducible form.

We first compared those two programs on several systems taken from different sources, in particular, arising from the problem of factoring difference operators. We have made those systems available on the internet at the URL mentioned in the previous section. The results are presented in Table 3, where the same letters in the "System" column denote the same matrices or vectors. The "Order" columns gives the number of unknowns, the column "Type" indicates whether we looked for polynomial (P) or rational (R) solutions, and the remaining columns indicate CPU times in seconds for each program and their ratio.

We then compared the same two programs on randomly generated systems: for each $n \in \{3, 5, 7, 10\}$, a set of 20 random $n \times n$ matrices was generated. For each of those matrices, 3 random n-vectors were generated, whose entries had degrees bounded by $10, 30$ and 50 respectively. Then, for each matrix-vector pair of the same size, an inhomogeneous difference system was constructed having the given vector for polynomial solutions. Here again, we have made those systems available on the internet at the URL mentioned in the previous section.

The results are presented in Table 4, where the rows represent the degree bound on the solution, and the columns represent the size of the system. Each cell of the table corresponds one series of 20 systems and contains 2 fractions: the first is the number of systems solved faster by SIF over the number of systems solved faster by EG', and the second is the total CPU time (in seconds) taken by SIF for the 20 systems over the CPU time taken by EG'.

Unlike in the case of regularizing algorithms, the tables for the linear difference system solvers are not as conclusive. Indeed, since the programs use different approaches, their weak and strong features are displayed on different systems. Furthermore, the difference could happen after the degree bounding phase when the coefficients of the solutions are actually computed. In any case, those two programs appear to perform generally within the same order of magnitude. Unlike in the case of EG' and `FFreduce`, there is no clear dependence of the ratio on the size of the systems (for each size we have examples where SIF performs better

Table 3: Comparison of linear difference system solvers

System	**Order**	**Type**	**SIF**	EG'	**SIF**/EG'
Ey=Ay	4	P	0,359	0,265	1,354716981
Ey=Ay	4	R	0,797	0,406	1,963054187
Ey=Ay+b	4	P	0,281	0,203	1,384236453
Ey=Ay+b	4	R	0,843	0,953	0,884575026
Ey=My	4	P	0,672	0,673	0,998514116
Ey=My	4	R	0,687	0,657	1,045662100
Ey=(M-1)y	4	R	0,312	0,266	1,172932331
Ey=By	4	P	0,422	0,344	1,226744186
Ey=By	4	R	0,687	0,609	1,128078818
Ey=Ny	9	P	0,906	0,407	2,226044226
Ey=Ny	9	R	0,828	0,406	2,039408867
Ey=Cy	9	P	1,625	1,39	1,169064748
Ey=Cy	9	R	1,874	2,563	0,731174405
Ey=Dy	16	P	3,672	1,797	2,043405676
Ey=Dy	16	R	6,609	2,75	2,403272727

Table 4: Comparison of linear difference system solvers for random systems

	3	**5**	**7**	**10**
10	0/20 8.078/3.687	0/20 27.639/6.373	0/20 90.283/13.294	0/20 1074.716/38.266
30	13/7 8.267/11.736	7/13 34.030/29.627	7/13 104.487/68.298	1/19 1587.253/172.221
50	16/4 9.799/76.312	17/3 32.189/164.454	16/4 145.370/311.345	9/11 3434.138/1090.361

than EG' and vice-versa), so it seems like a difficult task to implement a poly-algorithm that would detect automatically the most efficient method to use for a particular input.

We conclude with a final remark: while EG'-program has significantly improved its efficiency after the recent update of some modules, the *SIF*-program has not been updated for quite a long time, so we do not exclude the possibility that further improvements in *SIF* could lead to some changes in the last tables. They nevertheless reflect accurately the current status of those programs.

Acknowledgments

We are grateful to M. Barkatou and G. Labahn for kindly providing us with the implementations of *SIF* and `FFreduce` that we used in our experiments.

References

[1] S.A.Abramov: *EG*-eliminations. *Journal of Difference equations and applications*, 1999, Vol. 5, 393–433.

[2] S.A.Abramov, M.Bronstein: On solutions of linear functional systems, In *Proc. of ISSAC'2001.* ACM press, 2001, 1–6.

[3] S.A.Abramov, M.Bronstein: Linear algebra for skew-polynomial matrices, *Rapport de Recherche INRIA* RR-4420, March 2002, `http://www.inria.fr/RRRT/RR-4420.html`.

[4] S.A.Abramov, M.Barkatou: Rational solutions of first order linear difference systems, In *Proc. of ISSAC'1998.* ACM press, 2001, 124–131.

[5] S.A.Abramov, P.E.Glotov, D.E.Khmelnov: A scheme of eliminations in linear recurrence systems and its applications, In *Proc. of French-Russian Lyapunov Institute.*

[6] Barkatou M.A. (1989): Contribution à l'étude des équations différentielles et aux différences dans le champ complexe, thèse soutenue le 6 juin 1989 à l'institut national polytechnique de Grenoble (France).

[7] Barkatou M.A. (1999): On rational solutions of systems of linear differential equations, *Journal of Symbolic Computation*, **28**, 547–567.

[8] Bareisss E.H. (1968): Sylvestr's identity and multistep integer-preserving Gaussian elimination, *Math. Comp.*, **22**, 565–578.

[9] M. Bronstein, M. Petkovšek. An introduction to pseudo–linear algebra, *Theoretical Computer Science* **157** (1996) 3–33.

[10] Hilali A. , Wazner A. (1987): Formes super-irréductibles des systèmes différentiels linéaires. *Num. Math.***50**, 429-449.

[11] B.Beckermann, H.Cheng, G.Labahn: Fraction-free row reduction of matrices of squew polynomials, In *Proc. of ISSAC'2002*. ACM press, 2002, 8–15.

[12] T.Mulders, A.Storjohann: Rational solutions of singular linear systems. In *Proc. of ISSAC'2000*. ACM press, 2000, 242–249.

[13] O. Ore. Theory of non-commutative polynomials. *Annals of Mathematics* **34** (1933) 480–508.

[14] O. Ore. Formale Theorie der linearen Differentialgleichungen (Erster Teil). *J. für die reine und angewandte Mathematik* **167** (1932) 221–234.

[15] O. Ore. Formale Theorie der linearen Differentialgleichungen (Zweiter Teil). *J. für die reine und angewandte Mathematik* **1** (1932), 233–252.