ISSN 0361-7688, Programming and Computer Software, 2019, Vol. 45, No. 5, pp. 288–297. © Pleiades Publishing, Ltd., 2019.
Original Russian Text © S.A. Abramov, D.E. Khmelnov, 2019, published in Programmirovanie, 2019, Vol. 45, No. 5.

Package of Procedures for Inverting Matrices Whose Entries are Linear Difference Operators

S. A. Abramov[a],* and D. E. Khmelnov[a],**

[a] *Dorodnicyn Computing Center, Federal Research Center "Computer Science and Control" of Russian Academy of Sciences, ul. Vavilova 40, Moscow, 119333 Russia*

** e-mail: sergeyabramov@mail.ru*

*** e-mail: dennis_khmelnov@mail.ru*

Received August 31, 2018

Abstract—The matrices considered in this paper belong to $\mathrm{Mat}_n(\mathbb{K}[\sigma,\sigma^{-1}])$, i.e., to the ring of $n\times n$-matrices whose entries are scalar difference operators with the coefficients from the difference field $\mathbb{K}$ of characteristic 0 with automorphism ("shift") σ. A family of algorithms is discussed that allow one to check whether there exists an inverse matrix for a given matrix from $\mathrm{Mat}_n(\mathbb{K}[\sigma,\sigma^{-1}])$ in this ring and, if exists, to construct it. These algorithms are made to correspond to complexities in terms of the number of arithmetic operations and the number of shifts (i.e., applications of σ and σ^{-1}) in the field $\mathbb{K}$. The algorithms are implemented in the form of `Maple`-procedures. This makes it possible to experimentally compare them in terms of time spent. The selection of the best algorithm based on these experiments does not always coincide with the complexity-based selection. An attempt is made to find out why this happens. A package of procedures for solving the considered problems is suggested, where the main procedure includes a parameter that specifies which algorithm is to be applied. If this parameter is lacking, than an a priori specified algorithm is selected that is relatively good both from the complexity and experimental standpoint compared to the others.

DOI: 10.1134/S0361768819020026

1. INTRODUCTION

Matrices with entries belonging to different rings and fields are used in all areas of mathematics and in applications. In this paper, the case in point is matrices whose entries belong to the ring of linear difference scalar operators over a difference field $\mathbb{K}$ with an automorphism (shift) σ. The field $\mathbb{K}$ is assumed to have characteristic 0. The main problems discussed are those of checking whether a matrix of the type discussed is invertible and finding the inverse matrix if it exists. We give a brief survey of known algorithms based on regularization, i.e., on finding leading and trailing matrices associated with the given operator matrix whose entries belong to $\mathbb{K}$. In some algorithms, frontal and rear matrices, rather than the leading and trailing ones, are regularized (the entries of these matrices also belong to $\mathbb{K}$; in [1], for both types of matrices, the common term "revealing matrices" was used).

Thus, several algorithms have already been suggested for the matrices discussed, which are listed in Sections 3 and 4. Complexities of these algorithms have been studied. In these studies, complexities of two types were considered: (1) arithmetic complexity, i.e., complexity in the worst case in terms of the number of arithmetic operations in the field $\mathbb{K}$, and (2) shift complexity (also in the worst case) in terms of the number of applications of σ and σ^{-1}.

Some algorithms from this set have already been implemented as `Maple` procedures. We present lacking implementations of those algorithms that have relatively low complexity and results of experimental comparison of these algorithms in terms of their run times. Interestingly, algorithms with low complexity are not always the fastest ones, which is not surprising, of course, since complexity (does not matter whether arithmetic or shift complexity or even complexity in terms of the total number of both operations in the worst case) does not characterize all cost associated with the algorithm execution. Moreover, comparison of cost in the worst case does not cover all possible cases; the "worst case" seldom happens, and the cases included in the test set may not be the worst ones. It is worth noting that, when we compare complexities based on asymptotic estimates, we do not always get adequate understanding of values of one or another complexity for moderate-amount (not astronomical) data.

In the selection of the most convenient, for example, the fastest, algorithm from a set, complexity char-

acteristics of the algorithms from this set play very important role, of course. However, if complexities of the algorithms are close to one another and we have only asymptotic estimates, these estimates are helpful only on the stage of a preliminary selection. It is very desirable to carry out subsequent experimental comparison (testing), especially when sizes of the test tasks are "typical" of the situations for which the algorithms have been designed.

In the paper, we present some complexity estimates for the algorithms from the considered set, as well as results of experimental comparison of these algorithms.

Note that, in [2], we gave a reference to the website containing implementation of the algorithm with the least (in the asymptotic sense) arithmetic and shift complexities and noted that the implementation of the algorithm differs from the algorithm itself in what concerns regularization of the leading and trailing matrices by means of a certain special variant of the EG algorithm ([3, 4]). The selection of the appropriate variant of this regularization, however, was not considered in [2]. In this paper, special attention was paid to the comparison of algorithms at the implementation level.

A package of procedures for solving the considered problems is suggested, where the main procedure includes a parameter that specifies which algorithm is to be applied. If this parameter is lacking, than an a priori specified algorithm is selected that is good both from the complexity and experimental standpoint compared to the others.

This introduction is concluded by several comments and agreements.

When the case in point is operator matrices, the term *unimodular* matrix rather than the inverse matrix is conventionally used. We will also use this term.

Algorithms for checking unimodularity and constructing inverse matrices for the differential case, when K is a differential field of characteristic 0 with the differentiation operation $\delta = '$ and matrix entries are scalar linear differential operators over $\mathbb{K}$, were considered in [5]. For a given operator matrix L, both differential algorithms and difference algorithms (to be discussed below) calculate dimension of the solution space V_L of the corresponding system $L(y) = 0$. This is done under the assumption that components of the solutions belong to an adequate (see Section 2) extension of field $\mathbb{K}$ associated with L. As established earlier (see [6]), an operator matrix L of full rank (rows of L are independent over the ring of scalar linear operators) is unimodular if and only if $\dim V_L = 0$, i.e., if the space V_L itself is zero.

In the sequel, we use the following notation. The ring of $n \times n$ matrices (n is a positive integer) with entries from a ring or field R is denoted as $\mathrm{Mat}_n(R)$. If M is an $n \times n$ matrix, then $M_{i,*}$, $1 \le i \le n$, denotes the $1 \times n$ matrix equal to the ith row of M. The diagonal $n \times n$ matrix with entries $r_1, \ldots, r_n$ on the diagonal is denoted as $\mathrm{diag}(r_1, \ldots, r_n)$, and the identity $n \times n$ matrix, as I_n.

2. PRELIMINARIES

Recall that a *difference ring* is a commutative ring $\mathbb{K}$ with unity and automorphism σ (which will also be referred to as *shift*). If $\mathbb{K}$ is a field, then it is called a *difference field*. In what follows, the difference fields are assumed to be fields of characteristic 0.

A *ring of constants* of a difference field $\mathbb{K}$ is $\mathrm{Const}(\mathbb{K}) = \{c \in K \mid \sigma c = c\}$. If $\mathbb{K}$ is a difference field, $\mathrm{Const}(\mathbb{K})$ is a subfield of the field $\mathbb{K}$ (*the field of constants* of the field $\mathbb{K}$).

Let $\mathbb{K}$ be a difference field with automorphism σ and a ring Λ be a difference extension of field $\mathbb{K}$ (the corresponding automorphism of Λ coincides with σ on $\mathbb{K}$, so that we use the same notation σ for it).

Definition 1. A ring Λ that is a difference extension of field $\mathbb{K}$ is an *adequate* difference extension of field $\mathbb{K}$ if $\mathrm{Const}(\Lambda)$ is a field also for an arbitrary system

$$\sigma y = Ay, \quad y = (y_1, \ldots, y_n)^T,$$

with a matrix $A \in \mathrm{Mat}_n(\mathbb{K})$, the dimension of the space of solutions belonging to Λ^n that is linear over the field $\mathrm{Const}(\Lambda)$ is equal to n.

The existence of an adequate difference extension Λ for an arbitrary difference field is easily proved (see [7, Sect. 5.1]). For an arbitrary adequate extension, the equality $\mathrm{Const}(\Lambda) = \mathrm{Const}(\mathbb{K})$ is not guaranteed; in the general case, $\mathrm{Const}(\mathbb{K})$ is a proper subfield of $\mathrm{Const}(\Lambda)$.

Remark 1. The so-called q-difference case ([8, 9]) is included into the general difference case.

A *scalar* difference operator is an element of the ring $\mathbb{K}[\sigma, \sigma^{-1}]$.

Definition 2. For a nonzero scalar operator $f = \sum a_i \sigma^i$, its *leading* and *trailing orders* are defined as follows:

$$\overline{\mathrm{ord}} f = \max\{i \mid a_i \neq 0\}, \quad \underline{\mathrm{ord}} f = \min\{i \mid a_i \neq 0\},$$

and its *order* is defined as $\mathrm{ord} f = \overline{\mathrm{ord}} f - \underline{\mathrm{ord}} f$. It is assumed that $\overline{\mathrm{ord}} 0 = -\infty$, $\underline{\mathrm{ord}} 0 = \infty$, and $\mathrm{ord} 0 = -\infty$.

For a finite set F of scalar operators (a vector, a matrix, a matrix row, etc.), the leading order $\overline{\mathrm{ord}} F$ is defined to be the greatest leading order of its elements, the trailing order $\underline{\mathrm{ord}} F$, as the least trailing order of its elements, and $\mathrm{ord} F = \overline{\mathrm{ord}} F - \underline{\mathrm{ord}} F$.

A difference operator matrix is a matrix from $\mathrm{Mat}_n(\mathbb{K}[\sigma, \sigma^{-1}])$. In what follows, we associate such an operator matrix, with matrices from $\mathrm{Mat}_n(\mathbb{K})$. For brevity, an operator matrix is further referred to as simply *operator*.

An operator is said to have a *full rank* (or is an operator of full rank) if its rows are linearly independent over $\mathbb{K}[\sigma, \sigma^{-1}]$.

If

$$L \in \mathrm{Mat}_n(\mathbb{K}[\sigma, \sigma^{-1}]), \quad l = \overline{\mathrm{ord}}L, \quad t = \underline{\mathrm{ord}}L,$$

and $L \neq 0$, then L can be written in an *extended* form as

$$L = A_l\sigma^l + A_{l-1}\sigma^{l-1} + \cdots + A_t\sigma^t,$$

where $A_l, A_{l-1}, \ldots, A_t \in \mathrm{Mat}_n(\mathbb{K})$, with matrices A_l, A_t (the *leading* and *trailing* matrices of the original operator) being nonzero.

Definition 3. Let the leading orders of rows of an operator L be $\alpha_1, \ldots, \alpha_n$ and the trailing orders be $\beta_1, \ldots, \beta_n$. The *frontal* matrix of the operator L is the leading matrix of the operator PL, where

$$P = \mathrm{diag}(\sigma^{l-\alpha_1}, \ldots, \sigma^{l-\alpha_n}), \quad l = \overline{\mathrm{ord}}L.$$

Accordingly, the *rear* matrix of the operator L is the trailing matrix of the operator QL, where

$$Q = \mathrm{diag}(\sigma^{t-\beta_1}, \ldots, \sigma^{t-\beta_n}), \quad t = \underline{\mathrm{ord}}L.$$

The operator L is said to be *strictly reduced* if both the frontal and rear matrices of this operator are nonsingular.

Definition 4. An operator $L \in \mathrm{Mat}_n(\mathbb{K}[\sigma, \sigma^{-1}])$ is called *unimodular* or *invertible* if there exists an inverse operator $L^{-1} \in \mathrm{Mat}_n(\mathbb{K}[\sigma, \sigma^{-1}])$: $LL^{-1} = L^{-1}L = I_n$. The set of unimodular $n \times n$ operators is denoted as Υ_n. Two operators $L_1, L_2 \in \mathrm{Mat}_n(\mathbb{K}[\sigma, \sigma^{-1}])$ are said to be *equivalent* if $L_1 = UL_2$ for some $U \in \Upsilon_n$.

Further, a fixed adequate difference extension of the original difference field $\mathbb{K}$ with automorphism σ will be denoted as Λ. The space of solutions of system $L(y) = 0$ belonging to Λ^n will be denoted as V_L. For shortness, we will sometimes speak of V_L as the space of *solutions of the operator* L.

Theorem 1. ([10]) *Let* $L \in \mathrm{Mat}_n(\mathbb{K}[\sigma, \sigma^{-1}])$ *have full rank. Then*,

(i) *if* L *is strictly reduced, then* $\dim V_L = \sum_{i=1}^{n} \mathrm{ord}L_{i,*}$;

(ii) $L \in \Upsilon_n \Leftrightarrow V_L = 0$.

3. REGULARIZATION: FAMILIES OF ALGORITHMS EG AND RR

3.1. Algorithms EG_σ *and* $\mathrm{EG}_{\sigma^{-1}}$

For a full-rank operator L, algorithm EG_σ (see [3, 4, 11]) constructs an equivalent difference operator $L_+ \in \mathrm{Mat}_n(\mathbb{K}[\sigma, \sigma^{-1}])$ such that $\overline{\mathrm{ord}}L_+ = \overline{\mathrm{ord}}L$, $\underline{\mathrm{ord}}L_+ \geq \underline{\mathrm{ord}}L$, and L_+ has a nonsingular leading matrix. If L is not of full rank, the algorithm informs about this.

We present the basic idea of this algorithm without going into detail. Algorithm EG_σ constructs L_+ by modifying L; i.e., L is changed step by step turning gradually into L_+. On each step of the algorithm, it is checked whether rows of the leading matrix are linearly dependent over $\mathbb{K}$; if they are, then, among the rows that have nonzero coefficients in the linear dependency, the row that has the greatest order is selected (if the number of such rows is greater than one, any one of them is taken). Next, the row selected is reduced by means of the other rows of the operator. If the resulting row is zero, then the rank of the original operator is not full. Otherwise, with the help of σ, the reduced row is shifted such that its leading order becomes equal to the leading orders of other rows of the operator.

It was shown that, after several steps, the algorithm terminates: either we get a zero row in the operator or get an operator with a nonsingular leading matrix.

Similarly, algorithm $\mathrm{EG}_{\sigma^{-1}}$ constructs an equivalent difference operator L_- with a nonsingular trailing matrix.

The extended algorithm EG_σ (we call it ExtEG_σ) allows us, in addition to L_+, to find a unimodular operator $U \in \Upsilon_n$ such that $L_+ = UL$. In a similar way, algorithm $\mathrm{ExtEG}_{\sigma^{-1}}$ can be defined.

Proposition 1 ([2, 10]). *The estimate of arithmetic complexity of algorithms* EG_σ *and* $\mathrm{EG}_{\sigma^{-1}}$ *is given by*[1]

$$\Theta(n^{\omega+1}d + n^3d^2),$$

where ω *is the exponent of the matrix multiplication*, $2 < \omega \leq 3$. *The estimate of the shift complexity is given by*

$$\Theta(n^2d^2).$$

The estimates of the arithmetic and shift complexities of the algorithms ExtEG_σ *and* $\mathrm{ExtEG}_{\sigma^{-1}}$ *are given by*

[1] When finding asymptotic complexity estimates as functions of variables n and d (assuming that $n, d \to \infty$), we use not only the O-notation but also the Θ-notation (see [12] and [13, Sect. 2]); the relation $f(n,d) = \Theta(g(n,d))$ is equivalent to the conjunction $f(n,d) = O(g(n,d))$ & $g(n,d) = O(f(n,d))$. In other words, $f(n, d)$ and $g(n,d)$ are quantities of the same order. The relation $f(n,d) = \Theta(g(n,d))$ is stronger than the relation $f(n, d) = O(g(n,d))$.

$$\Theta(n^4d^2), \quad \Theta(n^4d^2).$$

3.2. Algorithms RR_σ and $\mathrm{RR}_{\sigma^{-1}}$

For a full-rank operator L, algorithm RR_σ (see [14, 15]) constructs an equivalent operator $\breve{L}_+ \in \mathrm{Mat}_n(\mathbb{K}[\sigma, \sigma^{-1}])$ such that $\overline{\mathrm{ord}}\breve{L}_+ \le \overline{\mathrm{ord}}L$, $\underline{\mathrm{ord}}\breve{L}_+ \ge \underline{\mathrm{ord}}L$, and $\breve{L}_+$ has a nonsingular frontal matrix. If L is not of full rank, the algorithm informs about this.

Algorithm RR_σ is also based on the idea of searching linear dependence of matrix (frontal, rather than leading) rows and performing shifts.

The extended algorithm ExtRR_σ allows us, in addition to $\breve{L}_+$, to find a unimodular operator $U \in \Upsilon_n$ such that $L_+ = UL$.

Similarly, algorithm $\mathrm{RR}_{\sigma^{-1}}$ constructs an equivalent difference operator L_- with a nonsingular rare matrix.

The extended variant of algorithm $\mathrm{RR}_{\sigma^{-1}}$ is called $\mathrm{ExtRR}_{\sigma^{-1}}$.

Proposition 2 ([10]). *The estimate of arithmetic complexity of algorithms* RR_σ *and* $\mathrm{RR}_{\sigma^{-1}}$ *is given by*

$$\Theta(n^{\omega+1}d + n^3d^2).$$

The estimate of the shift complexity is

$$\Theta(n^3d^3) \tag{1}$$

if results of all shifts of the rows are not stored. If all shift results are stored, then (1) *is replaced with* $\Theta(n^2d^3)$.

The estimates of the arithmetic and shift complexities of the algorithms ExtRR_σ *and* $\mathrm{ExtRR}_{\sigma^{-1}}$ *are given by*

$$\Theta(n^4d^2), \quad \Theta(n^5d^3)$$

if results of all shifts of the rows are not stored. If all shift results are stored, then the complexities are estimated as

$$\Theta(n^4d^2), \quad O(n^4d^3).$$

3.3. Algorithms $\Delta\mathrm{EG}_\sigma$, $\Delta\mathrm{EG}_{\sigma^{-1}}$

Definition 5. Let the ith row of the leading matrix of an operator L have the form

$$(\underbrace{0, \ldots, 0}_{k-1}, a, \ldots, b),$$

$1 \le k \le n$, $a \ne 0$. Then, the number k is called *indent* of the ith row of L. If the ith row of L is zero, then its indent is equal to $-\infty$.

If the rows of L have pairwise different positive indents, then the leading matrix is nonsingular: up to the order of the rows, it is a triangular matrix with nonzero diagonal elements. Let us assume that the rows $r_1 = L_{i,*}$ and $r_2 = L_{j,*}$ have identical positive indents k and $\overline{\mathrm{ord}}L_{i,*} = \overline{\mathrm{ord}}L_{j,*} = d$. Having performed one arithmetic operation in $\mathbb{K}$, one can find $v \in \mathbb{K}$ such that the row

$$r_2 - vr_1 \tag{2}$$

either has an indent that is greater than k, or its order is less than d, or both. One of the two rows r_1, r_2 that has smaller trailing order is replaced in L by row (2); if the orders of both rows are equal, either of them is replaced. If L has a full rank, then, after not more than $n \cdot nd$ such unimodular transformations (each of them is equivalent to the multiplication by a unimodular operator from the left), the leading matrix will become triangular. This technique can be used instead of searching linear dependence of rows of the leading matrix.

Remark 2. Under the assumption that $\mathbb{K}$ is a field of rational functions of x with automorphism $x \to x + 1$, this technique was used in [3] in the first version of algorithm EG (see. also [16]).

The use of this technique results in algorithms $\Delta\mathrm{EG}_\sigma$ and $\Delta\mathrm{EG}_{\sigma^{-1}}$ for finding an equivalent operator with a nonsingular leading or trailing, respectively, matrix.

Remark 3. If one of the algorithms EG_σ or $\Delta\mathrm{EG}_\sigma$ is applied to an operator with a nonsingular rear matrix, we get an operator that, along with a nonsingular leading matrix, has a nonsingular rear matrix. This follows from the replacement rule used: "One of the two rows r_1, r_2 that has lesser trailing order is replaced in L by row (2); if the orders of both rows are equal, either of them is replaced".

Proposition 3 ([2]). *The arithmetic complexity of* $\Delta\mathrm{EG}$ *algorithms is estimated as*

$$\Theta(n^3d^2). \tag{3}$$

The estimate of the shift complexity is

$$\Theta(n^2d^2). \tag{4}$$

The estimates both of the arithmetic and shift complexities of the algorithms $\mathrm{Ext}\Delta\mathrm{EG}_\sigma$ *and* $\mathrm{Ext}\Delta\mathrm{EG}_{\sigma^{-1}}$ *are given by*

Remark 4. In the differential case, the EG-eliminations result, generally, in an operator that is not equivalent to the original one. The operator obtained has all solutions of the original operator and, possibly, some additional solutions. It is because of this reason that, in the differential case, the algorithm for checking unimodularity, as well as the algorithm for constructing the inverse operator, is based on RR_δ [5].

The structure of the first difference algorithm ([10]) was similar to that of the differential algorithm. It is the subsequent use of the EG_σ and $\mathrm{EG}_{\sigma^{-1}}$ algorithms as the basis of the algorithms for checking unimodularity and constructing the inverse operator that made it possible to reduce shift complexity in the difference case.

4. CHECKING UNIMODULARITY, OPERATOR INVERSION

The existing algorithms for checking unimodularity of an operator L and for constructing the inverse operator L^{-1}, if exists, are based on the calculation of $\dim V_L$. This calculation, in turn, relies on the regularization algorithms (see Sect. 3 and Remark 3).

By virtue of Remark 3, the operators

$$\mathrm{EG}_\sigma(\mathrm{EG}_{\sigma^{-1}}(L)), \quad \Delta\mathrm{EG}_\sigma(\Delta\mathrm{EG}_{\sigma^{-1}}(L)) \tag{5}$$

equivalent to the operator L are strictly reduced. By Theorem 1, we can calculate $\dim V_L$ and establish whether L is unimodular. If $\dim V_L = 0$, then, by virtue of Theorem 1(i), the corresponding operator from (5) belongs to $\mathrm{Mat}_n(\mathbb{K})$ and L^{-1} is easily calculated.

The algorithms from [10] are developed similar to the algorithms for the differential case [5] with the help of RR, whereas the algorithm from [2] is based on ΔEG, which made it possible to reduce the shift complexity (see Remark 4). In [2], algorithms Unimodularity Testing and Inverse Operator are proposed, the arithmetic and shift complexities of which coincide with (3) and (4) and are, respectively

$$O(n^4d^2), \quad O(n^3d^2).$$

The inverse operator is constructed by extended versions of the basic algorithms, whereas, for checking unimodularity, basic versions are sufficient.

As shown in [10], the algorithms based on RR have complexities $\Theta(n^4d^2)$ and $\Theta(n^4d^3)$ (if results of all shifts are stored; otherwise, the shift complexity is $\Theta(n^4d^5)$. For the algorithms based on EG, the complexities are $\Theta(n^4d^2)$ and $\Theta(n^4d^2)$).

5. IMPLEMENTATION, EXPERIMENTAL COMPARISON

The considered algorithms were implemented[2] in Maple [17] (a preliminary version was presented in [2]). The implementations rely on the implementations of algorithms EG, ΔEG, and RR for the differential case described in [1]. для дифференциального случая. The procedures were modified for using in the difference case and supplemented by implementations of the extended versions of the algorithms. The algorithm for checking unimodularity and constructing the inverse operator that uses one of the algorithms EG, ΔEG, or RR (at user's option) as the base one is also implemented. In this implementation, the shift is defined to be $\sigma y(x) = y(x-1)$, and a special choice of the equation to be replaced is required when performing the elimination step in algorithms EG and ΔEG to guarantee the termination of the algorithm operation. There exists a version of these algorithms that does not require such a special choice of the equation replaced. In this version, the оператор $\sigma - 1$ is used for the shift step (a similar variant of the EG algorithm for the q-difference case was used [18, Rem.3]). However, when the EG and ΔEG algorithms are used for the implementation of the algorithm for checking unimodularity and searching the inverse operator, the version with the control of the order of eliminations is applied (see Remark 3).

The algorithms are implemented as procedures of the new package `EGRRext`. The package supplies the following procedures:

- `EG`, implements the EG algorithm and its extended version ExtEG;
- `RR`, implements the RR algorithm and its extended version ExtRR;
- `TriangleEG` implements the ΔEG algorithm and its extended version ExtΔEG;
- `IsUnimodular`, implements the Unimodularity Testing and Inverse Operator algorithms.

An operator $L = A_l\sigma^l + A_{l-1}\sigma^{l-1} + \cdots + A_t\sigma^t$ in the input parameters of the procedures is presented as a list

$$[A, l, t],$$

where A is the explicit matrix

$$A = (A_l|A_{l-1}| \ldots |A_t)$$

of size $n \times n(l-t+1)$. The explicit matrix A is presented in the form of a standard `Maple` object `Matrix`. The entries of this explicit matrix are rational functions of one variable, which, in turn, are presented in a standard way. If $t = 0$, then the operator can also be presented by means of only one explicit matrix A.

The `IsUnimodular` procedure returns `true` or `false` as the result of checking of whether the input operator is unimodular. If the optional input parameter—variable name—is specified, then the corresponding inverse operator is also calculated (if exists) and assigned to this variable. The inverse operator is also represented as a list consisting of its explicit matrix and its leading and trailing orders. If the optional variable name is not specified, then the procedure employs the Unimodularity Testing algorithm,

[2] The codes are available at http://www.ccas.ru/ca/egrrext

otherwise, the Inverse Operator algorithm (see Section 4). The procedure has one more optional parameter `method`, which may take values `EG`, `TEG`, or `RR` specifying which of the algorithms EG, ΔEG, or RR is used as the base one (if this parameter is not specified, then the EG algorithm is used). In the first implementation presented in [2], only the EG algorithm was used as the base one.

Example 1. Consider the application of the `IsUnimodular` algorithm to the operator

$$L = \begin{pmatrix} 1 & -\frac{1}{x}\sigma \\ \frac{x^2}{2} & -\frac{x}{2}\sigma + 1 \end{pmatrix} = \begin{pmatrix} 0 & -\frac{1}{x} \\ 0 & -\frac{x}{2} \end{pmatrix}\sigma + \begin{pmatrix} 1 & 0 \\ \frac{x^2}{2} & 1 \end{pmatrix}.$$

The explicit matrix is given by

$$\begin{pmatrix} 0 & -\frac{1}{x} & 1 & 0 \\ 0 & -\frac{x}{2} & \frac{x^2}{2} & 1 \end{pmatrix},$$

with $l = 1$ and $t = 0$. The procedure is called twice for each method: first, to check unimodularity; then, to construct the inverse operator. Additionally, the computation time is output:

```
> L := Matrix([[0, -1/x, 1, 0],
               [0, -x/2, x^2/2, 1]]);
```

$$L := \begin{bmatrix} 0 & -\frac{1}{x} & 1 & 0 \\ 0 & -\frac{x}{2} & \frac{x^2}{2} & 1 \end{bmatrix}$$

```
> st:=time():
  IsUnimodular(L, x, 'method'='EG');
  time()-st;
```

true

0.021

```
> st:=time():
  IsUnimodular(L, x, 'InvL_EG',
                     'method'='EG');
  time()-st;
```

true

0.026

```
> InvL_EG;
```

$$\left[\begin{bmatrix} -\frac{(x+1)^2}{2x} & \frac{1}{x} & 1 & 0 \\ 0 & 0 & -\frac{x^2}{2} & 1 \end{bmatrix}, 1, 0\right]$$

```
> st:=time():
  IsUnimodular(L, x, 'method'='TEG');
  time()-st;
```

true

0.003

```
> st:=time():
  IsUnimodular(L, x, 'InvL_TEG',
                     'method'='TEG');
  time()-st;
```

true

0.008

```
> InvL_TEG,
```

$$\left[\begin{bmatrix} -\frac{(x+1)^2}{2x} & \frac{1}{x} & 1 & 0 \\ 0 & 0 & -\frac{x^2}{2} & 1 \end{bmatrix}, 1, 0\right]$$

```
> st:=time():
  IsUnimodular(L, x, 'method'='RR');
  time()-st;
```

true

0.032

```
> st:=time():
  IsUnimodular(L, x, 'InvL_RR',
                     'method'='RR');
  time()-st;
```

true

0.040

```
> InvL_RR,
```

$$\left[\begin{bmatrix} -\frac{(x+1)^2}{2x} & \frac{1}{x} & 1 & 0 \\ 0 & 0 & -\frac{x^2}{2} & 1 \end{bmatrix}, 1, 0\right]$$

Thus,

$$L^{-1} = \begin{pmatrix} 1 - \frac{(x+1)^2}{2x}\sigma & \frac{1}{x}\sigma \\ -\frac{x^2}{2} & 1 \end{pmatrix}.$$

□

Example 2. The operator

$$M\begin{pmatrix} I_n & M_1 & 0_n \\ 0_n & I_n & M_2 \\ 0_n & 0_n & I_n \end{pmatrix},$$

where 0_n is the zero matrix of size $n \times n$ and M_1, $M_2 \in \mathrm{Mat}_n(\mathbb{K}[\sigma, \sigma^{-1}])$ are arbitrary operators, is unimodular for any M_1 and M_2. The inverse operator is given by

$$M^{-1}\begin{pmatrix} I_n & -M_1 & M_1M_2 \\ 0_n & I_n & -M_2 \\ 0_n & 0_n & I_n \end{pmatrix}.$$

A series of experiments have been executed. For each of them, a pair of 50%-sparse operators M_1 and M_2 with entries given by random polynomials of the degree not greater than two was generated. The operators M_1 and M_2 had equal numbers of rows, the order of the operator M_1 was 2, and the order of the operator took values $d = 3, 5, 7, 9, 11$ (and this was the order of the operator M). The number of rows in M_1 and M_2 was equal to $n = 2, 3, 4$ (accordingly, the number of rows in M was $k = 4, 6, 8$). In each experiment, the inverse operator M was calculated by means of all three methods. The results obtained are presented in Table 1. In a number of experiments, the method based on RR failed: the calculations were stopped when the amount of memory on the computer was insufficient. In the table, this is marked by the sign >, with the stop time being shown. □

Table 1. Results of experiments (in seconds)

		d = 3	d = 7	d = 11	d = 15
k = 6	EG	0.172	0.297	0.500	0.672
	TEG	0.125	0.375	1.032	2.563
	RR	0.937	3.844	5.844	11.500
k = 9	EG	0.610	2.562	6.813	21.422
	TEG	0.985	4.594	11.859	26.469
	RR	4.625	82.953	>5100	>6000
k = 12	EG	1.125	5.319	31.953	17.422
	TEG	3.219	15.735	36.312	59.312
	RR	18.250	170.078	>8300	>19100
k = 15	EG	3.657	49.219	215.329	908.984
	TEG	8.047	41.516	90.531	213.032
	RR	1031.422	>20200	>6100	>21100

6. DISCREPANCY ANALYSIS

From Table 1, it can be seen that, on the whole, the results of the experiments agree well with the theoretical complexity estimates. However, the growth of the computation time with the growth of the number of rows and the order of the operator is much greater than it could be expected based on the theoretical estimates. To understand the cause of these differences, we implemented a mode of operation of the `IsUnimodular` procedure in which characteristic parameters of intermediate computation results are additionally output. These parameters are output after a zero row appears in the revealing matrix (either after performing a reduction in the EG and RR algorithms or after performing one or several replacements of rows in the ΔEG algorithm). For such parameters, we use parameters of the operator row in which the zero row appeared in the revealing matrix, namely, the amount of data in the representation of this row: the standard `Maple` function `length` and the maximum of sums of powers of the numerator and denominator in this operator row are used. This additional data showed that, in the course of intermediate calculations, the coefficients of the operator grow significantly. Even though the sums of powers of the numerator and denominator in the original operator M and its inverse do not exceed 2 and 4, respectively, in the intermediate calculations, this number grew up to hundreds and even thousands. The amount of data grew up in a similar way.

Cumulated data on the same experiments are presented in Table 2. For each experiment, the following five parameters are presented:

- maximum amount of data used among all intermediate rows,
- maximum value of the maximum sum of powers of the numerator and denominator among all intermediate rows,
- the number of the intermediate rows,
- average amount of data used over all intermediate rows,
- average value of the maximum sum of powers of the numerator and denominator over all intermediate rows.

For the experiments that were forced to terminate, parameters calculated for the intermediate results obtained before the calculations were stopped are shown.

The significant growth of the computation time in the experiments with the growth of the number of rows and the operator order shown in Table 1 agrees well with the growth of the coefficients in the intermediate calculations presented in Table 2 by characteristic parameters. As a result of this growth, actual computational cost required for the calculation of one arithmetic operation, as well as for the shift operation, start to grow significantly, and the latter growth turns out to be more considerable compared to the growth of the

Table 2. Characteristic parameters of intermediate calculations in the experiments

		d = 3	d = 7	d = 11	d = 15
k = 6	EG	88	151	227	294
		2	2	2	2
		11	17	25	30
		63	105	154	201
		1	1	1	2
	TEG	138	1169	11188	52729
		3	10	28	51
		8	14	23	31
		85	333	1870	11334
		2	4	9	18
	RR	843	10488	18719	42013
		17	39	46	58
		9	17	23	31
		281	2162	2757	9759
		6	15	12	22
k = 9	EG	1779	36697	174632	556987
		20	145	82	122
		21	33	45	57
		291	6134	32615	116131
		4	26	28	46
	TEG	3411	29855	161977	411703
		31	54	94	129
		15	27	39	51
		1132	11140	51255	114004
		13	30	46	68
	RR	3702	363550	>80150107	>186956920
		27	245	>2532	>2870
		15	27	>19	>14
		1379	119074	12653623	28598185
		16	119	601	693
k = 12	EG	1460	64793	458748	332049
		8	56	113	88
		27	43	59	74
		261	11011	84783	50312
		2	15	38	24
	TEG	9414	63490	202193	340476
		54	96	96	117
		20	36	52	67
		2872	27979	88520	121198
		19	44	64	69
	RR	26360	460957	>190475287	>168181419
		58	190	>2902	>2303
		20	36	>20	>33
		6169	95011	23245248	24965627
		29	74	611	539

Table 2. (Contd.)

		d = 3	d = 7	d = 11	d = 15
k = 15	EG	24060	346779	1383502	3938486
		39	110	170	236
		34	54	74	94
		3023	58838	259553	831962
		10	35	62	93
	TEG	58343	131895	400249	2066923
		110	114	112	222
		24	44	64	84
		11782	60119	115177	600333
		37	66	74	119
	RR	981268	>87692555	>307185043	>472458234
		692	>2130	>2944	>2800
		24	>16	>11	>15
		137728	17253891	39424269	59156406
		157	643	599	595

number of such operations, which was taken into account in the algorithm complexity. For different methods, this growth has different rate. In our experiments, the worst algorithm from this point of view was the RR algorithm. This agrees well with the results of comparison of similar algorithms in the differential case, which were presented in [1]. Note also that, when the growth in different algorithms has similar characteristics (i.e., when the cost required for performing one operation in each algorithm are comparable), the number of such operations plays the most important role, and the relative computation time agrees with the theoretical complexity estimates.

One more specific feature of the EG and RR algorithms should also be noted. To perform reductions in these algorithms, linear dependencies in the revealing matrices are sought. To this end, in the implementation, procedure `NullSpace` of the `LinearAlgebra` package of `Maple` is used. This procedure is capable of finding more than one linear dependence, which results in different continuations of the computation depending on what linear dependence found was excluded. In our implementation, the first linear dependence found was selected after they have been sorted out by means of the `ComplexitySort` procedure from package `SolveTools`. Tables 3 and 4 present results of computations for some of the test systems presented in Tables 1 and 2 that are based on the selection of the last, rather than first, linear dependence after sorting them. The results presented demonstrate that the order of the eliminations also affects the growth of coefficients in the intermediate calculations and, hence, the total computation time. This factor is also not taken into account in the complexity estimates. However, in practice, its effect may be significant.

Table 3. Results of experiments with a different selection of the linear dependence (in seconds)

		d = 3	d = 7
k = 6	EG	0.187	0.359
	RR	0.500	2.406
k = 9	EG	0.829	20.062
	RR	4.047	29.688
k = 15	EG	19.906	16645.953
	RR	29.547	14979.109

Relying on the results of the reported experiments, as well as the of earlier conducted experiments for the differential case ([1]), we may conclude that, from the practical point of view, it is reasonable to use the versions of the Unimodularity Testing and Inverse Operator algorithms that are based on the EG algorithm. It is this version of the implementation that was presented in [2] and is used by default (if the parameter method is not specified) in the implementation reported in this work. These experiments also showed that, from the practical point of view, it is useful to have implementations of different algorithms. It may occur that an algorithm that is worse than others in terms of complexity or computation time is the most efficient one for a particular problem. Therefore, in our implementation, the user has an opportunity to select other algorithms as well.

Table 4. Characteristic parameters of intermediate calculations in the experiments with a different selection of the linear dependence

		d = 3	d = 7
k = 6	EG	2	4
		11	19
		63	155
		1	2
	RR	87	438
		2	4
		6	14
		65	172
		1	2
k = 9	EG	4178	186239
		23	174
		21	28
		570	19817
		5	39
	RR	3717	165811
		23	174
		15	20
		695	24107
		7	54
k = 12	EG	186239	18578353
		174	1096
		28	44
		19817	2422729
		39	233
	RR	165811	15621336
		174	1096
		20	36
		24107	2512118
		54	284

ACKNOWLEDGMENTS

This work was supported in part by the Russian Foundation for Basic Research, project no. 19-01-00032-a.

The authors are grateful to A.A. Ryabenko for useful comments on the first version of this paper.

REFERENCES

1. Abramov, S.A., Ryabenko, A.A., and Khmelnov, D.E., Revealing matrices of linear differential systems of arbitrary order, *Program. Comput. Software,* 2017, vol. 43, pp. 67–74.
2. Abramov, S.A. and Khmelnov, D.E. On unimodular matrices of difference operators, *Proc. of CASC,* 2018, vol. 11077, pp. 18–31.
3. Abramov, S.A., EG-Eliminations, *J. Difference Equations Appl.,* 1999, vol. **5**, pp. 393–433.
4. Abramov, S.A. and Bronstein, M., Linear algebra for skew-polynomial matrices, *Rapport de Recherche INRIA, RR-4420,* March 2002. http://www.inria.fr/RRRT/RR-4420.html.
5. Abramov, S., On the differential and full algebraic complexities of operator matrices transformations, *Lect. Notes Comput. Sci.,* 2016, vol. 9890, pp. 1–14.
6. Abramov, S.A. and Barkatou, M., On the dimension of solution spaces of full rank linear differential systems, *Lect. Notes Comput. Sci.,* 2013, vol. 8136, pp. 1–9.
7. Abramov, S.A. and Barkatou, M., On solution spaces of products of linear differential or difference operators, *ACM Commun. Comput. Algebra,* 2014, vol. 4, pp. 155–165.
8. Kac, V. and Cheung, P., *Quantum Calculus,* New York: Springer, 2002.
9. Andrews, G.E., q-Series: Their development and application in analysis, number theory, combinatorics, physics, and computer algebra, *Pennsylvania: CBMS Regional Conference Series, AMS, R.I.*, 1986, vol. 66.
10. Abramov, S.A., Inverse linear difference operators, *Comput. Math. Math. Phys.,* 2017, vol. 57, no. 12, pp. 1887–1898.
11. Abramov, S.A. and Bronstein, M., On solutions of linear functional systems, Proc. of ISSAC'2001, 2001, pp. 1–6.
12. Knuth, D.E. Big omicron and big omega and big theta, *ACM SIGACT News,* 1976, vol. 8, no. 2, pp. 18–23.
13. Abramov, S.A., *Lektsii o slozhnosti algoritmov* (Lectures on Complexity of Algorithms), Moscow: MTsNMO, 2012, 2nd ed.
14. Barkatou, M.A, El Bacha, C., Labahn, G., and Pflügel, E., On simultaneous row and column reduction of higher-order linear differential systems, *J. Symbolic Computation,* 2013, vol. 49, no. 1, pp. 45–64.
15. Beckermann, B., Cheng, H., and Labahn, G. Fraction-free row reduction of matrices of Ore polynomials, *J. Symbolic Computation,* 2006, vol. 41, pp. 513–543.
16. Mulders, T. and Storjohann, A., On lattice reduction for polynomial matrices, *J. Symbolic Computation,* 2004, vol. 37, no. 4, pp. 485–510.
17. Maple online help: https://www.maplesoft.com/support/help/
18. Abramov, S.A., Ryabenko, A.A., and Khmelnov, D.E., Laurent, rational, and hypergeometric solutions of linear q-difference systems of arbitrary order with polynomial coefficients, *Program. Comput. Software,* 2018, vol. 44, no 2, pp. 120–130.