A note on computing the regular solutions of linear differential systems *

Sergei A. Abramov, Denis E. Khmelnov
Russian Academy of Sciences,
Dorodnicyn Computing Centre,
Vavilova 40, 119991, Moscow GSP-1,
Russia
`abramov@ccas.ru, khmelnov@ccas.ru`

Abstract

We present an approach to find all regular solutions of a system of linear ordinary differential equations using EG'-algorithm [2, 3] as an auxiliary tool.

1 Introduction

Let

$$L = Q_\rho(z)D^\rho + \cdots + Q_1(z)D + Q_0(z), \tag{1}$$

where $D = d/dz$.

Assume that $Q_\rho(x), \ldots, Q_0(x)$ are polynomials in z over $\mathbb{C}$. A regular solution of the equation $Ly = 0$, or, the same of the operator L, at a fixed point $z_0 \in \mathbb{C}$, is a solution of the form

$$(z - z_0)^\lambda F(z) \tag{2}$$

with $F(z) \in \mathbb{C}((z - z_0))[\log(z - z_0)]$, where $\mathbb{C}((z - z_0))$ is the field of Laurent series (here we do not consider convergence problems; all series are formal). The value λ is the *exponent* of regular solution (2). W.l.g. we will suppose that $z_0 = 0$. The problem of constructing all regular solutions of L at 0 can be solved, e.g., by Frobenius algorithm ([6]), which is based on using the indicial equation $f(\lambda) = 0$ of L at 0 (a right-hand side which contains, in particular, factors $f(\lambda)$ and z^λ, must be constructed for L; the corresponding solutions must be differentiated by λ and so on). Not only the values of roots of $f(\lambda) = 0$, each taken separately, are substantial for Frobenius' algorithm, but also multiplicities of the roots and the existence of roots differing by integers. To apply Frobenius' algorithm to a system of linear difference equations one has to transform the system into a large order scalar differential equation (e.g. by the cyclic vector method). The scalar equation may have huge coefficients, that makes the approach quite unpractical.

In [4, Section 5] another algorithm for constructing regular solutions of a first order system of the form

$$y' = Ay, \quad A \in \mathrm{Mat}_N(\mathbb{C}(z)) \tag{3}$$

was described (in [5, Section 3.3] an extended version of the same algorithm was presented). This algorithm is direct, i.e., it uses neither the cyclic vector method nor any other decoupling procedure. For a given value λ the algorithm constructs a basis of regular solutions at 0 whose exponent is λ (if there exists no such solution then the basis is empty). The algorithm from [4, 5] does not need any information neither on multiplicity of λ nor on the existence of other roots with integer distance from λ. This algorithm constructs step by step a sequence of first order linear differential systems, enumerated by $0, 1, \ldots$, which are inhomogeneous starting

*The work is partially supported by the ECO-NET program of the French Foreign Affairs Ministry.

from the system with the number 1 (the corresponding right-hand sides contain solutions of the preceding systems). If

$$z^\lambda \left(g_0(z) + g_1(z)\frac{\log z}{1!} + g_2(z)\frac{\log^2 z}{2!} + \cdots + g_m(z)\frac{\log^m z}{m!} \right) \tag{4}$$

is a regular solution of (3) then $g_i(z)$ is a Laurent series solution of the constructed i-th system; the process of finding of regular solutions of (3) is terminated when the current constructed system has no non-zero Laurent series solutions. It is necessary to be able to find Laurent series solutions of a given system (the recognizing of the existence included). To do this in [4, 5] a transformation of the system into a so-called *super-irreducible* form ([7]) is computed. Once the system is in a super-irreducible form then a bound of the "pole order" of the Laurent series solution, and then the coefficients of the solutions themselves can be computed directly (in turn). Additionally, if the original system is in a super-irreducible form, then one can find all possible exponents $\lambda_1, \lambda_2, \dots$ of its regular solutions.

We describe in this paper a modification of the algorithm from [4, 5]. This modification does not use the transformation of a system into a super-irreducible form. Instead, we use EG'-algorithm from [2, 3] (see Section 3). Note that sometimes the transformation into a super-irreducible form as well as the application of EG'-algorithm is not fast. When we need to solve a linear differential system of a large size, then it could make a sense to try both approaches; if we are lucky, at least one (it is possible that only one) of them will solve the problem.

It is convenient for this purpose to reorganize the algorithm from [4, 5] in such a manner that it would be applicable to any linear system

$$Ly = 0 \tag{5}$$

where L has the form (1) with

$$Q_i(z) \in \mathrm{Mat}_N(\mathbb{C}[z]), \quad i = 0, \dots, \rho; \tag{6}$$

in particular, L can be a scalar operator of arbitrary order (in this case $N = 1$). This is done in Section 2; some useful properties of this version of the algorithm are described as well. In Section 4 the algorithm is summarized and in Section 5 we describe some computing remarks useful for implementation of the algorithm. Detailed example of the application of the algorithm is presented in Section 6. The implementation of the algorithm in Maple and related experiments are described in Section 7.

Acknowledgements. We would like to thank Prof. M. Barkatou (University of Limoges) for valuable discussions about regular solutions of linear differential systems.

2 Linear differential systems of arbitrary order

First, consider the problem of the search for Laurent series solutions of (5). We can construct the associated recurrent system $Rc = 0$ with

$$R = P_l(n)E^l + \cdots + P_t(n)E^t, \quad P_j(n) \in \mathrm{Mat}_N(\mathbb{C}[n]), \quad j = t, \dots, l \tag{7}$$

for the coefficients of any such solution. If $\det P_l(n)$ is the zero polynomial, then it is possible to transform the recurrent system into a system with a non-zero $\det P_l(n)$. This can be done by EG'-algorithm (see Section 3). In the rest of this section we suppose that $\varphi(n) = \det P_l(n)$, $\varphi(n) \in \mathbb{C}[n] \setminus \{0\}$.

Set $\psi(n) = \varphi(n - l)$ and n_0, n_1, resp., minimal and maximal integer roots of $\psi(n)$ (if there is no integer root, then (5) has no Laurent series solution). Any Laurent series solution of (5) has no term $c_k z^k$, $c_k \in \mathbb{C}^n$, with $k < n_0$. Using the recurrence $Rc = 0$ and the constructed constraints, we can, by a linear algebra procedure, compute a basis of the linear space of initial segments

$$c_{n_0} z^{n_0} + c_{n_0+1} z^{n_0+1} + \cdots + c_M z^M,$$

where M is a fixed integer such that $M \geq n_1$ and M is greater than all indexes involved into the constraints.

Observe, that if our differential system is inhomogeneous with a Laurent series right-hand side (the coefficients of that right-hand side are given using a recurrence), then similarly we will be able to construct a basis of the affine space of Laurent series solutions.

If $\psi(n)$ has a non-integer root λ, then the preliminary change of the depended variable $y = x^\lambda \bar{y}$ will produce a new equation $\bar{\psi}(n) = 0$, where $\bar{\psi}(n) = \psi(n-\lambda)$. Therefore we always can work with integer roots.

Apparently, the result of application of L to

$$g(z)\frac{\log^m z}{m!} \tag{8}$$

where $m \geq 0$ can be represented in the form

$$L_{m,m}(g)\frac{\log^m z}{m!} + L_{m,m-1}(g)\frac{\log^{m-1} z}{(m-1)!} + \cdots + L_{m,1}(g)\frac{\log z}{1!} + L_{m,0}(g), \tag{9}$$

where the coefficients of differential operators $L_{i,j}$ belong to $\mathrm{Mat}_N(\mathbb{C}(z))$.

Proposition 1 *The coefficients of all operators $L_{i,j}$ belong to $\mathrm{Mat}_N(\mathbb{C}[z, z^{-1}])$ and, additionally,*

$$\begin{aligned} &L_{0,0} = L_{1,1} = L_{2,2} = \ldots = L, \\ &L_{1,0} = L_{2,1} = L_{3,2} = \ldots, \\ &L_{2,0} = L_{3,1} = L_{4,2} = \ldots, \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \end{aligned} \tag{10}$$

in (9).

Proof: After the applying of L to (8) one gets, e.g., $L_{m,m-1}(g)$ by gathering together all terms that contain one time differentiated factor (8); but

$$\left(\frac{\log^m z}{m!}\right)' = \frac{1}{z}\cdot\frac{\log^{m-1} z}{(m-1)!}$$

and the new factor $1/z$ does not depend on m (due to considering $(\log^m z)/m!$ instead of $\log^m z$). ■

Set

$$\begin{aligned} &L_0 = L_{0,0} \ (= L_{1,1} = L_{2,2} = \ldots = L), \\ &L_1 = L_{1,0} \ (= L_{2,1} = L_{3,2} = \ldots), \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \end{aligned}$$

If $\mathrm{ord} L = d$, then $\mathrm{ord} L_i = d - i$, $i = 0, \ldots, d$, $L_{d+1} = L_{d+2} = \ldots = 0$. We obtain

$$L\left(\sum_{m=0}^{k} g_{k-m}(z)\frac{\log^m z}{m!}\right) = L_0(g_0)\frac{\log^k z}{k!} + (L_1(g_0)+L_0(g_1))\frac{\log^{k-1} z}{k-1!} + \cdots + (L_k(g_0)+L_{k-1}(g_1)+\cdots+L_0(g_k)).$$

Therefore the equality

$$L\left(\sum_{m=0}^{k} g_{k-m}(z)\frac{\log^m z}{m!}\right) = 0$$

is valid iff

$$\begin{aligned} &L_0(g_0) = 0, \\ &L_0(g_1) = -L_1(g_0), \\ &L_0(g_2) = -L_1(g_1) - L_2(g_0), \\ &L_0(g_3) = -L_1(g_2) - L_2(g_1) - L_3(g_0), \\ &\ldots\ldots\ldots\ldots \end{aligned} \tag{11}$$

Denote S_i the system of first $i+1$ equations from (11), i.e., of the equations whose left hand sides are $L_0(g_0), \ldots, L_0(g_i)$. Hence we have the following proposition.

Proposition 2 *The equality (5) has regular solution*

$$\sum_{m=0}^{k} g_{k-m}(z)\frac{\log^m z}{m!} \tag{12}$$

iff $(g_0(z),\dots,g_k(z))$ *is a Laurent series solution of* S_k.

Note the following. We find $g_0(z)$ using the first equation from (11). This solution may contain some arbitrary constants. When we use $g_0(z)$ in the right-hand side of the second equation from (11), some of those arbitrary constants have to be specified to make the second equation solvable in non-zero Laurent series; if this is possible, then we get $g_1(z)$ that in its turn may contain arbitrary constants and so on. So when Laurent series solutions of S_i are constructed and we solve S_{i+1}, we, in general situation, decrease the number of the arbitrary constants in solutions of S_i and find new Laurent series g_{i+1} (which may contain some arbitrary constant as well).

Denote G_k the set of all regular solutions of the form (12) of equation (5) (the case $g_0(z) = 0$ is not excluded).

Proposition 3 $G_0 \subset G_1 \subset \cdots \subset G_k \subset \cdots$.

Proof: Suppose that $(g_0^*,\dots,g_i^*)$ is an $(i+1)$-tuple of Laurent series which is a solution of S_i. Set $(g_0^{**},\dots,g_{i+1}^{**}) = (0, g_0^*,\dots,g_i^*)$. It is easy to check that $(g_0^{**},\dots,g_{i+1}^{**})$ then satisfies the system S_{i+1}. The claimed follows from Proposition 2. ■

Apparently, if (12) is a solution of (5) $(g_0,\dots,g_k) \in \mathbb{C}((z))^{k+1}$ with $g_0 \neq 0$, then $k \leq \mathrm{ord} L - 1$. Therefore, starting from some non-negative integer k, all systems S_m, $m > k$, have only such solutions in $\mathbb{C}((z))^{k+1}$ that contain $g_0 = 0$. If k is such non-negative integer and we have constructed the set U of all solutions of S_k in $\mathbb{C}((z))^{k+1}$, then using the elements of this set we can construct all wanted regular solutions of (5). We will obtain U in the form of a vector $(g_0(z),\dots,g_k(z))$ whose entries may contain some arbitrary constants.

It is very valuable, that all equations from (11) have in the left-hand side the operator $L_0 = L$, and we have the corresponding recurrent operator for it with the non-singular leading matrix.

3 EG'-algorithm as an auxiliary tool for constructing regular solutions

Linear recurrences with variable coefficients are of interest for many applications (e.g. in combinatorics and numeric computation). Consider the recurrence of the form

$$P_l(n)z_{n+l} + P_{l-1}(n)z_{n+l-1} + \cdots + P_t(n)z_{n+t} = r_n \tag{13}$$

where $l \geq t$ are arbitrary integers, $z = (z^1,\dots,z^N)^T$ is a column vector of unknown sequences (such that $z_i = (z_i^1,\dots,z_i^N)^T$), the right-hand side r_n is a vector of polynomials in n and the matrix coefficients $P_t(n),\dots,P_l(n)$ are polynomial in n, and $P_t(n), P_l(n)$ are non-zero. Note that it's often convenient to regard the matrix $P(n) = (P_l(n)|\dots|P_t(n))$, which is referred to as the *explicit matrix* of the recurrence. Each of the matrices $P_t(n),\dots,P_l(n)$ is called a *block* of the explicit matrix (resp. of the system (13)) and the matrices $P_l(n)$ and $P_t(n)$ are called *leading* and *trailing* matrices of the explicit matrix (resp. of the system (13)).

The roots of the determinants of the matrices $P_t(n)$ and $P_l(n)$ (when those matrices are non-singular over $\mathbb{C}(n)$) are always important for determining the structure of the solution space (e.g. bounds on the orders of the solutions). It may happen however that either $P_t(n)$ or $P_l(n)$ is singular. In that case, not only it is impossible to compute bounds on the orders of the solutions, but it also makes difficult, from a computational standpoint, to use the recurrence (13) to compute the sequence of vectors it generates.

A natural solution in that case is to compute an equivalence transformation of the recurrence 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.

Such EG-algorithm was developed in [1] and later improved (EG'-algorithm) in [2]. It allows transforming the recurrence (13) to the form with the non-singular leading (resp. trailing) matrix. The given system is equivalent to the transformed system accompanied by a set of linear constraints (the set may be empty).

The general scheme of the algorithm is 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 (13) by the shift operator 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). Obviously, all the corresponding transformations are performed on the right-hand side as well. Note that the reduction step may generate a set of linear constraints because of multiplications of the transformed rows by polynomials having integer roots. Each of the constraints is a linear relation that contains a finite set of variables z_i^j. The process terminates if some special precautions are taken. To guarantee termination, 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.

One of the important applications of the algorithms is solving linear functional (e.g. differential) systems with polynomial coefficients. The systems induce recurrence systems for the coefficients of their series solutions in some basis. This associated recurrence is of the form (13). EG'-algorithm is shown to be efficient enough ([3]) for the purpose and used for finding polynomial, rational, and formal series solutions of linear functional systems.

4 Algorithm

Summarizing the above information, the general scheme of finding regular solutions of the system (5) is the following:

1. For a given system S in the form (5), construct the associated matrix recurrence in the form (13). Using EG'-algorithm transform it into the recurrence of the same form but such that $\varphi(n) = \det P_l(n)$ is not identically zero. Compute all roots of $\varphi(n)$, divide them into the groups of ones having integer differences, and construct the set Λ consisting of representatives of the groups (one representative out of each group).

2. For each $\lambda \in \Lambda$ compute regular solution whose exponent is λ:

   (a) Compute system S_λ by substituting $y = x^\lambda y_\lambda$. Construct the associated matrix recurrence in the form (13). Using EG'-algorithm transform it into the recurrence R_λ of the same form but such that $\varphi_\lambda(n) = \det P_l(n)$ is not identically zero. The transformed recurrence includes a set of additional constraints (the set may be empty) and the transformed right-hand side in a generic form.

   (b) Determine the number M_λ of needed initial terms of Laurent series such that all integer roots of $\varphi_\lambda(n)$ and all indices of constraints are less than the number.

   (c) Successively solve systems (11) for the needed number of initial terms of Laurent series using the recurrence R_λ while it's possible. It gives regular solutions y_λ of S_λ in the form (12).

3. Combine all solutions $\{y_\lambda\}_{\lambda\in\Lambda}$ into general regular solution $y = \sum_{\lambda\in\Lambda} x^\lambda y_\lambda$.

5 Computing remarks

5.1 Associated recurrence

The main part of the algorithm (see Section 4) is solving a single system from the sequence (11). As it is mentioned above all systems from (11) have in the left-hand side the same operator $L_0 = L$. Hence the associated matrix recurrences have the same left-hand side as well. But the right-hand sides of the recurrences are different. In order to regard the different right-hand sides at once during EG'-algorithm transformations we may apply all the transformations to a generic right-hand side. Then as a result of EG'-algorithm we have the transformed recurrence in the form (13) with non-singular $P_l(n)$, set of linear constraints and the transformed right-hand side in a generic form. Each component of this generic transformed right-hand side is a linear combination of possibly shifted components of the right-hand side before transformations (this is consequence of the corresponding operations in the recurrence during EG'-algorithm). In this way we can use the same transformed recurrence for solving any single system from the sequence (11) specifying the concrete right-hand side by substituting corresponding values into the generic right-hand side.

5.2 Computing the right-hand side

Computing the right-hand side is not so simple since the right-hand side for the m-th system before transformations is in the form

$$-\sum_{k=1}^{m} L_k(g_{m-k}), \tag{14}$$

where $g_0, \dots, g_{m-1}$ are Laurent series solutions of the preceding systems in the sequence (11). Since in practice we represent Laurent series solution by segment of initial terms, we need to determine the needed numbers of initial terms of $g_0, \dots, g_{m-1}$.

In its turn the numbers depend on the number M_λ of initial terms of transformed right-hand side which is determined on the step 2b of the algorithm (see Section 4) for all systems in the sequence (11) and ensures that next terms of the series are computed from preceding ones by simple use of the recurrence.

So we compute the transformed right-hand side in the following way:

1. Taking into account M_λ and the form of the components of transformed generic right-hand side, compute the numbers of needed initial terms of the components of right-hand side before transformations to ensure the number of initial term in the transformed right-hand side being equal to M_λ.

2. Taking into account form of the operators $L_1, \dots, L_m$, compute the numbers of initial terms of $g_0, \dots, g_{m-1}$ to ensure the needed numbers of initial terms of the components of right-hand side before transformations.

3. Compute the corresponding initial segments of $g_0, \dots, g_{m-1}$.

4. Compute the initial segment of the right-hand side before transformations substituting the initial segments of $g_0, \dots, g_{m-1}$ into (14).

5. Compute the initial segment of the transformed right-hand side substituting the initial segment of right-hand side before transformations into the transformed right-hand side in a generic form.

5.3 Extending solution component

Computing the transformed right-hand side depends on computing initial segments of $g_0, \dots, g_{m-1}$ (step 3 in Section 5.2). Since the required number of initial terms of the solution component g_k may be greater than the number of the initial terms computed on the preceding steps of the algorithm, we need to extend the component. In order to do it we need to compute next terms using the associated recurrence. It means we need to extend the corresponding transformed right-hand side, computing it using approach from Section 5.2 substituting M_λ by new number. Note that again it may require extending other solution components. So this procedure is recursive.

5.4 Computing initial segment

When the transformed right-hand side of the recurrence is computed, solving a system from the sequence (11) for the needed number of initial terms of Laurent series may be performed one by one using the recurrence. In each step one of the following options occurs:

- the next term is computed being expressed as a function of the previous terms;
- a linear constraint on the previous terms appears, which can be either resolved or is inconsistent that means that there is no Laurent series solution;
- the next term is a new arbitrary constant (it may be defined on the next steps of the computations by resolving constraints).

After all steps either all initial terms are computed (some of them being arbitrary constants) or it's determined that there is no Laurent series solution.

Note the following:

1. Since each of the $g_0, \ldots, g_{m-1}$ may have arbitrary constants, the right-hand side for m-th system in the sequence (11) may have the same arbitrary constants. It leads to the fact that during computing the initial segment of the Laurent series solution of the m-th system some of the constants may be defined due to resolving appearing constraints. It may lead to transforming the initial segment of g_0 to zero. Since the number of terms in the segment is determined in such a way that all the rest terms are computed in turn by the associated recurrence with the non-singular leading matrix, it gives that g_0 is identically zero. As it is noted above, if the case happens it means that all solution components are computed and m-th system has no proper Laurent series solutions.
2. As it follows from the Proposition 3 we can use only the last found solution of the form (12) as the regular solution whose exponent is λ, since it contains all previously found solutions of the form as well.

6 Example

Consider the following system:

$$\begin{array}{l} \frac{13}{2}x^2D^2y1(x) + \frac{33}{4}xDy1(x) + \frac{9}{8}y1(x) + x^3D^3y1(x) - x^2D^2y2(x) - 3xDy2(x) - \frac{3}{4}y2(x) = 0 \\ x^2Dy2(x) + \frac{3}{2}y2(x) - x^2y2(x) = 0 \end{array} \tag{15}$$

The explicit matrix of the associated recurrence is

$$\begin{pmatrix} (n-2)(n-1)n + \frac{13}{2}(n-1)n + \frac{33}{4}n + \frac{9}{8} & -(n-1)n - 3n - \frac{3}{4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & n+\frac{1}{2} & 0 & -1 \end{pmatrix},$$

with leading index of the recurrence $l = 0$, and trailing one $t = -2$. EG'-algorithm gives

$$\begin{pmatrix} (n-2)(n-1)n + \frac{13}{2}(n-1)n + \frac{33}{4}n + \frac{9}{8} & -(n-1)n - 3n - \frac{3}{4} & 0 & 0 & 0 & 0 \\ 0 & n+\frac{3}{2} & 0 & -1 & 0 & 0 \end{pmatrix},$$

with no constarints. The determinant of the leading matrix is $\frac{1}{16}(8n^3 + 28n^2 + 30n + 9)(2n + 3)$. The roots are $-\frac{1}{2}$ and $-\frac{3}{2}$ and they form one group. Let set of representatives $\Lambda = \{-\frac{1}{2}\}$.

After substitution $y = x^{-\frac{1}{2}}\bar{y}$, explicit matrix of the associated recurrence is

$$\begin{pmatrix} (n-2)(n-1)n + 5(n-1)n + 4n & -(n-1)n - 2n & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & n & 0 & -1 \end{pmatrix},$$

with leading index of the recurrence $l = 0$, and trailing one $t = -2$. EG'-algorithm gives

$$\begin{pmatrix} (n-2)(n-1)n + 5(n-1)n + 4n & -(n-1)n - 2n & 0 & 0 & 0 & 0 \\ 0 & n+1 & 0 & -1 & 0 & 0 \end{pmatrix}, \tag{16}$$

with no constraints. The right-hand side before transformations in a generic form is

$$\begin{pmatrix} r_n^1 \\ r_n^2 \end{pmatrix},$$

and the transformed one is

$$\begin{pmatrix} r_n^1 \\ r_{n+1}^2 \end{pmatrix} \tag{17}$$

The determinant of the leading matrix is $n(n^2+2n+1)(n+1)$. The roots are 0 and -1. Taking into account $l = 0$, it means that the initial segment needs to be from x^{-1} till x^0.

Solving first system from the sequence (11) means solving the recurrence (16) with the zero right-hand side for terms from -1 till 0. It gives

$$g_0 = \begin{pmatrix} c_{0,1}x^{-1} + c_{0,2} + O(x) \\ c_{0,3}x^{-1} + c_{0,3} + O(x) \end{pmatrix}$$

Here and below $O(x^k)$ means the tail of the formal series, i.e. the terms of the power greater than or equal to k.

Compute the operator needed for right-hand side of the second system from the sequence (11):

$$L_1(y(x)) = \begin{pmatrix} y_1(x) + 7xDy_1(x) + 3x^2D^2y_1(x) - y_2(x) - 2xDy_2(x) \\ xy_2(x) \end{pmatrix} \tag{18}$$

Taking into account (17), we compute that the initial terms of the right-hand side before transformations should be up to x^1. Then from (18) we conclude that the initial terms of g_0 should be up to x^1 as well. Extending g_0 gives

$$g_0 = \begin{pmatrix} c_{0,1}x^{-1} + c_{0,2} + \frac{1}{4}c_{0,3}x + O(x^2) \\ c_{0,3}x^{-1} + c_{0,3} + \frac{1}{2}xc_{0,3} + O(x^2) \end{pmatrix}$$

That leads to the transformed right-hand side being equal to

$$\begin{pmatrix} -c_{0,3}x^{-1} - c_{0,2} + c_{0,3} - \frac{1}{2}c_{0,3}x + O(x^2) \\ -c_{0,3}x^{-1} - c_{0,3} - \frac{1}{2}c_{0,3}x + O(x^2) \end{pmatrix} \tag{19}$$

Solving the recurrence (16) with respect to the right-hand side (19) gives

$$g_1 = \begin{pmatrix} c_{1,1}x^{-1} + c_{1,2} + O(x) \\ c_{1,3}x^{-1} + c_{1,3} + O(x) \end{pmatrix}.$$

and due to resolving appearing constraints changes preceding solution component

$$g_0 = \begin{pmatrix} c_{0,1}x^{-1} + O(x^2) \\ 0 + O(x^2) \end{pmatrix}$$

Compute the next operator needed for the right-hand side of the third system from the sequence (11):

$$L_2(y(x)) = \begin{pmatrix} 2y_1(x) + 3xDy_1(x) - y_2(x) \\ 0 \end{pmatrix} \tag{20}$$

Taking into account (17), (18), and (20) we compute that the initial terms both of g_0 and of g_1 should be up to x^1. g_0 is already in the needed form and extending g_1 gives

$$g_1 = \begin{pmatrix} c_{1,1}x^{-1} + c_{1,2} + \frac{1}{4}c_{1,3}x + O(x^2) \\ c_{1,3}x^{-1} + c_{1,3} + \frac{1}{2}xc_{1,3} + O(x^2) \end{pmatrix}$$

That leads to the transformed right-hand side being equal to

$$\begin{pmatrix} -c_{1,3}x^{-1} - c_{1,2} + c_{1,3} - \frac{1}{2}c_{1,3}x + O(x^2) \\ -c_{1,3}x^{-1} - c_{1,3} - \frac{1}{2}c_{1,3}x + O(x^2) \end{pmatrix} \tag{21}$$

Solving the recurrence (16) with respect to the right-hand side (21) changes preceding component g_0 to be zero due to resolving appearing constraints. It means that all solutions components are already found and no proper g_2 exists.

Combining all the above we find the solution of (15)

$$y = \begin{pmatrix} x^{-1/2}(\ln(x) * (c_1x^{-1} + O(x^2)) + c_4x^{-1} + c_2 + \frac{1}{4}c_3x + O(x^2)) \\ x^{-1/2}(c_3x^{-1} + c_3 + \frac{1}{2}xc_3 + O(x^2)) \end{pmatrix}$$

7 Implementation and experiments

The algorithm is implemented in Maple on top of the package `LinearFunctionalSystems`, which is implementing EG'-algorithm and some algorithms for finding closed-form solutions of linear functional systems with polynomial coefficients. The algorithm is implemented as the function that returns the regular solutions of the specified linear differential system of equations with polynomial coefficients with involved Laurent series represented as their initial segments. The number of the initial terms are determined automatically to ensure that the rest terms of the series can be directly computed (in turn) using associated recurrences (i.e. the leading matrix is invertible for all the rest terms). In order to extend initial segments of the Laurent series of the found regular solution the other function is provided, which returns the regular solution with the segments extended to the specified degree.

For experiments we use as well a Maple implementation of the algorithm from [5] (presented in the package `ISOLDE`).

We compared the two programs on two types of sets of generated systems.

For the first type of the sets we generated randomly the systems of the form $Y'(x) = A(x)Y(x)$, where $A(x)$ is the matrix of rational functions, and the entries on each row of the matrix have the same denominator. For each $n \in \{4, 7, 10\}$, three sets of 20 random $n\times n$ matrices were generated. For each of the sets, numerators and denominators of the entries of the generated matrices had degrees bounded by $d \in \{4, 8, 12\}$ respectively. More precisely the following Maple instruction was used as a generator:

```
randpoly(x, terms=rand(1..floor(d/2))(), expons=rand(0..d))
```

Additionally, the probability of non-zero entries in the matrix was set to 3/5. 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/regsol/comparisons.html`. The results for the first type of the sets are presented in Table 1, where the rows represent the degree bound on the coefficients, 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 the program from `ISOLDE` over the number of systems solved faster by EG'-based one, and the second is the total CPU time (in seconds) taken by the program from `ISOLDE` for the 20 systems over the CPU time taken by EG'-based one. Additionally two numbers are indicated: the first one shows the number of solutions of the systems in the set containing logarithms and the second one shows the number of trivial (zero) solutions of the systems in the set.

Table 1: Results for the first type of the sets

	4	**7**	**10**
4	3/17 13.642/11.279 8–0	1/19 747.451/187.077 10–0	1/19 1060.768/399.171 13–0
8	2/18 17.405/10.362 4–0	2/18 276.639/312.407 9–1	0/20 627.547/164.203 13–0
12	4/16 15.609/13.251 4–2	2/18 211.671/389.345 8–2	1/19 1371.342/184.999 10–0

For the second type of the sets we constructed the systems in the following way. First for each pair l and d, where $l \in \{2, 4, 6\}$ and $d \in \{3, 5, 7\}$, we constructed 20 random scalar recurrences of the order bounded by l and coefficients of the degrees bounded by d. More precisely the following Maple instruction was used as a generator:

```
    (n-rand(-5..5)())^2*E^l+randpoly(E, terms=rand(1..l)(), expons=rand(l), coeffs=
 (()->randpoly(n, coeffs=rand(-5..5), terms=rand(1..floor(d/3)+1)(), expons=rand(0..d))))
```

Each scalar recurrence can be treated as being induced by scalar differential equation. So, second, for the constructed scalar recurrences we constructed corresponding scalar differential equations. Third we constructed first order differential systems corresponding to these equations. And as the last step we transformed the systems in accordance with changing function variables induced by randomly generated transformation matrices with integer entries and the probability of non-zero entries in the matrices set to 1/2 (only invertible matrices were selected). The results for the second type of the sets are presented in Table 2, where the rows represent the order bound on the source scalar recurrence, and the columns represent the degree bound on its coefficients. Cells contain the same information as in the first type, except for the numbers of logarithmic and trivial solutions since all solutions for the second type are non-trivial and logarithmic.

Table 2: Results for the second type of the sets

	3	**5**	**7**
2	0/20 9.640/4.376	1/19 22.625/25.594	5/15 64.592/185.720
4	0/20 15.811/7.390	2/18 30.248/46.553	7/13 71.404/122.832
6	0/20 21.567/8.920	1/19 45.389/23.609	9/11 125.859/517.655

As it is mentioned above, since the programs use different approaches, their weak and strong features are displayed on different systems. As we can see for the first type of the sets most systems were solved faster by the EG'-based program, however for some sets the total CPU time was less for `ISOLDE` since a few systems in these sets were solved much faster by this program. For the second type of the sets we can see both the same effect and the growth of the number of the systems solved faster by `ISOLDE` with the growth of the degree bound of the coefficients of the source scalar recurrences. 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'-based program has improved its efficiency after the recent update of some modules, the package `ISOLDE` has not been updated for quite a long time, so we do not exclude the possibility that further improvements in the package could lead to some changes in the table. It nevertheless reflects accurately the current status of those programs.

References

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

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

[3] S. Abramov, M. Bronstein, D. Khmelnov. Regularization of linear recurrence systems. In *Transactions of French-Russian A.M.Lyapunov Institute, MSU*, 2003, Vol.4, 158 – 171.

[4] M. Barkatou. On rational solutions of systems of linear differential equations. *J. Symbolic Computation*, 28(4 and 5), 547–568, October/November 1999.

[5] M. Barkatou, E. Pfluegel. An algorithm computing the regular formal solutions of a system of linear differential equations. *J. Symbolic Computation*, 28(4 and 5), 569—587, October/November 1999.

[6] E.A. Coddington, N. Levinson. Theory of ordinary differential equations. McGraw-Hill book company Inc. 1955

[7] A. Hilali, W. Wazner. Formes super-irréducible de systèmes différentiels linéares. *Num. Math.* **50**, 1987, 429–449.