Solving large linear algebraic systems in the context of integrable non-abelian Laurent ODEs

Thomas Wolf, Brock University, Ontario, Canada
email: twolf@brocku.ca

Eberhard Schrüfer, Bonn, Germany
email: eberhard.schruefer@ca-musings.de

Kenneth Webster, Brock University, Ontario, Canada
email: kw07ty@brocku.ca

September 6, 2011

Abstract

The paper reports on a computer algebra program LSSS (Linear Selective Systems Solver) for solving linear algebraic systems with rational coefficients. The program is especially efficient for very large sparse systems that have a solution in which many variables take the value zero. The program is applied to the symmetry investigation of a non-abelian Laurent ODE introduced recently by M. Kontsevich. The computed symmetries confirmed that a Lax pair found for this system earlier generates all first integrals of degree at least up to 14.

1 Introduction

In mathematics, science and engineering many problems lead to sparse linear algebraic systems. For example, in a discretization of a smooth object the relations between variables defined in neighbouring points typically involve only a small number of variables that is dependent on the dimension of the object which is low.

For sparse linear systems there are computer algebra programs available, for example, the code of Roman Pearce [12] that has been incorporated into MAPLE 14: `SolveTools:-Linear`. It is automatically applied if a sparse system is to be solved.

The typical concern of solvers for sparse systems is to avoid a choice of pivot that increases the number of variables in equations and thus leads to non-sparse equations during the solution process. For many problems this can not be avoided, only delayed. For example, when solving the sparse linear system resulting from the discretization of the (partial differential) Laplace equation (i.e. heat equation in the stationary limit) it is clear that the temperature at each single point depends on the temperature at *all* boundary points and that the temperature inside is typically non-zero.[1]

[1] Even if the temperature on the whole boundary is zero except on an ε-sized part where it is positive, the temperature at all inner points will be positive.

type	"numerical" systems	"selection" systems
examples	systems resulting from a discretization of PDEs	systems resulting from a symmetry investigation of PDEs
value of free parameters when applying the solution of the linear system	any floating point numbers (boundary values of PDE)	0 or 1 (to isolate the individual symmetries)
number of zero-valued variables in solution	essentially none	most variables
sparsity	yes	yes
overdetermination	no	yes
usability of iteration schemes for large problems of that type	useful	not useful

Table 1: Characterization of two different types of sparse linear systems

The linear systems we studied are also sparse but different in nature. The value of most of their variables in the solution is exactly zero. Such systems play the role of selectors and are therefore called "selection systems" in this paper, because from a large number of monomials, all with a constant undetermined coefficient, some linear combinations of the monomials are selected which satisfy some condition and the other monomials are dropped by setting their coefficient to zero. This is very different from linear sparse systems like those resulting from the discretization of the Laplace equation. Table 1 compares both types of problems.

Selection systems have a number of useful properties.

- The existence of zero-valued variables allows an effective solution as performed by the program LSSS, described in this paper. LSSS is running under the computer algebra system Reduce ([7]).
- From the application it is clear whether a linear system belongs to this class, i.e. whether many variables take the value zero and thus a specialized computer program should be applied.
- The efficiency of solving selection systems increases with the complexity of the problem. For example, the higher the degree of the polynomial ansatz in a symmetry investigation - the larger is the size of the linear system for the unknown coefficients but the higher is also the percentage of zero-valued coefficients and the higher is the efficiency of the solution technique that uses zero-valued variables (see section 3).
- Not only can such linear systems be solved efficiently, they can also be formulated more economically. More precisely, it is possible to formulate much smaller equivalent systems as shown in section 3.7.

What justifies the creation of special purpose programs for selection systems is the fact that they occur frequently, especially in integrability investigations of differential equations when computing infinitesimal symmetries, first integrals, conservation laws, variational principles or other qualitative properties which may but need not exist and for which an ansatz can be formulated.

The following section describes an application that leads to a series of sparse systems with a high percentage of zero-valued variables. This application was the starting point for the development of LSSS. Readers interested essentially in the computational aspects of formulating and solving the systems may proceed directly to section 3 which describes methods for formulating and solving selection systems efficiently.

The times for solving the linear systems resulting in the applications are shown in section 4. The subsequent section 5 discusses aspects of the computer algebra system REDUCE which become important in large computations. Sections 6 and 7 report on tests of other computer algebra systems. The paper concludes with a list of available procedures in section 8 and a summary.

2 The application

2.1 A non-abelian ODE-system

In the programme of M. Kontsevich of creating a proper non-commutative algebraic geometry inspired by modern quantum field-theoretic requirements and challenges, he proposed a non-commutative version of symplectic geometry in which integrable non-abelian ODE-systems appear naturally ([1]).

More specifically, he considered the following non-abelian ODE system:

$$u_t = uv - uv^{-1} - v^{-1}, \qquad v_t = -vu + vu^{-1} + u^{-1} \tag{1}$$

where u, v are non-commutative variables (in particular, square matrices of arbitrary size). In comparison to non-abelian ODE systems with polynomial right hand sides investigated first in [2] and later in [3], [4] the system (1) involves Laurent polynomials, i.e. polynomials in u, v and inverses u^{-1}, v^{-1}.

Based on numerical experiments Kontsevich conjectured that (1) is integrable itself. This was demonstrated recently in [5] where a Lax pair has been found. In addition a pre-Hamiltonian operator is given in [5] that maps gradients of trace first integrals to infinitesimal symmetries. The existence of infinitely many symmetries and even better of a Lax pair is taken as an indicator of integrability of an ODE or PDE system ([6]). To verify that the Lax pair produces all Laurent polynomial first integrals our strategy is to compute by brute force all Laurent polynomial infinitesimal symmetries up to some degree and to compare them with the symmetries produced from the Lax pair. This paper reports on the computer algebra program LSSS that was developed to solve the linear algebraic systems resulting in the computation of all such symmetries up to degree 16.

2.2 Symmetries

For a given system of equations

$$u_t = P_1(u, v, u^{-1}, v^{-1}), \qquad v_t = P_2(u, v, u^{-1}, v^{-1}) \tag{2}$$

where P_1, P_2 are polynomials in u, v, u^{-1}, v^{-1} a Laurent polynomial (infinitesimal) symmetry is defined through two polynomials Q_1, Q_2 in u, v, u^{-1}, v^{-1} such that the flow

$$u_\tau = Q_1(u, v, u^{-1}, v^{-1}), \qquad v_\tau = Q_2(u, v, u^{-1}, v^{-1}) \tag{3}$$

commutes with the system, i.e.

$$D_\tau D_t\, u = D_t D_\tau\, u, \tag{4a}$$

$$D_\tau D_t\ v = D_t D_\tau\ v. \tag{4b}$$

The vector $(Q_1, Q_2)^t$ is called generator of the symmetry.

In this paper the system (2) is given through (1). The polynomials Q_1, Q_2 are generated by a special purpose procedure that creates the most general (inhomogeneous) polynomials up to some given degree in the non-abelian variables u, v, u^{-1}, v^{-1} with symbolic (abelian) coefficients c_i. The only condition satisfied by Q_1, Q_2 is that u, u^{-1} and v, v^{-1} are not standing next to each other anywhere in any term as they would cancel.

The symmetry conditions (4) have to be satisfied identically for any u, v. That means, after (4) is formulated, the coefficients of all different products of powers of u, v, u^{-1}, v^{-1} have to be set to zero generating a linear system for the unknown coefficients in the symmetry generators(3). This process is called 'splitting' in the remainder of the paper.

The number $\hat{t}_n$ of terms of each Q_i of degree n (which is one half of the number of unknown coefficients) satisfies the recursive relation $\hat{t}_n = 3\hat{t}_{n-1} + 2$ with $\hat{t}_0 = 1$ because apart from the coefficients c_i the polynomial of degree 0 has only the term 1 and the most general polynomial of degree n is obtained by multiplying each term of the most general polynomial of degree $n-1$ from one side, say from the left, with all possible three of the four factors u, v, u^{-1}, v^{-1} which do not give a cancellation. One exception is the term 1 that is multiplied with each one of the 4 factors i.e. from this term results one extra term and the extra term 1 is added giving $\hat{t}_n = 3\hat{t}_{n-1} + 1 + 1 = 3\hat{t}_{n-1} + 2$. Thus the total number of terms t_n occuring in Q_1 and Q_2 of degree n satisfies $t_n = 3t_{n-1} + 4$.

2.3 Necessary symmetry conditions

Additional information available for our symmetry computation results from a separate computation of full first integrals I satisfying $D_t I = 0$ where D_t is the total time-derivative. The condition that a Laurent polynomial in u, v is a first integral is very restrictive because it implies that each of the $m \times m$ components of the matrix first integral is an integral of its own. Restrictive conditions lead to very overdetermined systems (here linear) which are easier to solve. Therefore it was possible to compute all first integrals up to degree 14 with the result that

$$I = uvu^{-1}v^{-1}, \quad I^{-1} = vuv^{-1}u^{-1} \tag{5}$$

and their integer powers $I^k,\ \ k = -3, .., 3$ are the only first integrals up to degree 14.

From the symmetry conditions (4) and $D_t I = 0$ follows

$$D_t D_\tau I = D_\tau D_t I = 0$$

and further from I^k being the only first integrals (up to degree 14)

$$D_\tau I = \sum_{k=-k_0}^{k_0} a_k I^k, \tag{6a}$$

and similarly

$$D_\tau I^{-1} = \sum_{k=-k_0}^{k_0} b_k I^k, \tag{6b}$$

for sufficiently high k_0, in our case $k_0 = 3$ because I is of degree 4. These conditions involve a few more extra unknown constants a_k, b_k but adding (6b) is beneficial because these are first order conditions involving fewer terms than the full symmetry conditions (4) which are of second order.

n	k	e_1	t_1	e_2	t_2	p
3	106	142	192	448	1,034	1
4	322	430	616	1,412	3,706	2
5	970	1,294	1,904	4,448	12,914	4
6	2,914	3,886	5,784	13,878	44,098	5
7	8,746	11,662	17,440	43,052	148,346	7
8	26,242	34,990	52,424	132,954	493,162	8
9	78,730	104,974	157,392	409,470	1,623,842	12
10	236,194	314,926	472,312	1,258,526	5,304,562	13
11	708,586	944,782	1,417,088	3,862,086	17,212,778	17
12	2,125,762	2,834,350	4,251,432	11,835,758	55,535,578	18
13	6,377,290	8,503,054	12,754,480	36,228,892	178,298,450	24
14	19,131,874	25,509,166	38,263,640	110,777,292	569,970,466	25
15	57,395,626	76,527,502	$> 1.1\times10^8$	$> 3.3 \times 10^8$	$> 1.7 \times 10^9$	31
16	$> 172\times10^6$	$> 229\times10^6$	$> 3.4\times10^8$	$> 1\times10^9$	$> 5.7\times10^9$	32

Table 2: For each symmetry ansatz of degree $n = 3..16$ are listed the numbers: k of variables, e_1 of equations and t_1 of terms of system $D_\tau I = 0$, e_2 of equations and t_2 of terms of system $[D_t, D_\tau](u, v) = 0$ and p of free parameters of the solution.

3 The solution of selection systems

The linear systems resulting from splitting the necessary and sufficient conditions (4) and the additional necessary conditions (6) cover a wide range of sizes as shown in table 2. As discussed in section 2.2 the numbers k_n of unknowns for a symmetry ansatz of degree n grow like $k_{n+1} = 3k_n+4$ which is also the growth rate of the number e_1 of equations in the necessary condition $D_\tau I = 0$.[2]

As the table indicates, the linear systems have a few characteristic properties that need to be exploited in order to compute high degree symmetries. These properties are:

- *Overdetermination:* There are 4.5 to 5.5 times as many equations as unknowns.
- *Sparsity:* Equations have on average 4 to 5 terms.
- *Zero-valued Variables:* Most of the variables take an exact value of zero in the solution. Even more importantly, the percentage of zero-valued variables is increasing as the degree of the ansatz increases, i.e. as the system to be solved increases in size. For systems resulting from symmetries of degree 4 this percentage is 93.2% and for symmetries of degree 16 the percentage is 99.966% . Nevertheless, the solution for degree 16 is not trivial. It has 58118 non-vanishing variables and 32 free parameters.

3.1 Ideas for a solver of large sparse linear algebraic systems

The following ideas take advantage of the features of selection systems as listed above and describe the stages of development of the computer algebra program LSSS. These steps have been to:

- start with a program for solving a stream of equations,

[2]The vanishing of constants a_k in (6a) and b_k in (6b) follows when formulating these conditions.

- sort equations initially by size, shortest first,
- apply 1-term equations very efficiently and apply them before any other equation,
- be able to generate only 1-term equations before formulating the whole system,
- iterate between the exclusive formulation and the application of 1-term equations,
- be able to generate and take advantage of extra necessary conditions before working on the original system, and
- choose from possibly different options to generate 1-term equations the most efficient one.

Any extra low level procedures that were written to implement these concepts assume that systems are linear in the unknowns. This allows them to be more efficient than universal routines.

In the following subsections more detailed comments are made to each of the measures.

3.2 A stream of equations

In earlier work [11] large and very overdetermined polynomial systems were solved which had too many equations (in the order of 10^{14}) to be even formulated initially. A linear algebraic system solver was developed at the time for related problems dealing with the system of equations as an incoming stream of data not to be stored in RAM memory, only to be processed equation by equation. A strength of this algorithm is that any limitation on available RAM memory poses only restrictions on the size of the preliminary solution of the system and thus only on the number of variables, not on the number of equations.

At the start of this program a preliminary solution is initialized as an empty list. Then with each incoming equation the following steps are performed.

- The equation is simplified modulo a preliminary solution stored in the form of a substitution list $g_i = f_i(g_j)$ where g_k are the unknowns and f_i are linear expressions in the unknowns except g_k from any of the left hand sides.
- If the resulting simplified equation is not an identity then it is solved for one of the g_m,
- substitutions $g_m = f_m(g_j)$ are performed in all f_i of the preliminary solution, and
- $g_m = f_m(g_j)$ is appended to the substitution list.

In the computation of high order symmetries this procedure is applied to the remaining system after all 1-term equations have been determined and used. The remaining system is small enough that it does not have to be stored on hard disk and then read again from disk.

3.3 Sorting equations

A first speedup is obtained by sorting equations according to size, shortest first, before processing them as described above leading to the times shown in column (B) of table 4. For example, for symmetries of degree 9 the speed up is a factor of 4.

Processing shorter equations earlier means that the substitution list involves shorter right hand sides and reduces incoming equations to shorter size earlier which adds shorter new substitution rules to the preliminary solution, i.e. it is a self-amplifying increase of efficiency.

In symbolic mode of REDUCE the sorting of equations can be done very effectively, for example, by establishing a list $L = \{l_1, l_2, l_3, ...\}$ where l_i is a list of (pointers to) all equations with i terms and then linking these lists. All steps are done in place by handling pointers to equations without the need to copy or rewrite equations. Assigning an equation to a list l_i is done efficiently by having 2 pointers, one stepping from one term to the next in the equation and the other at the same time stepping from l_i to l_{i+1}. When the first pointer reached the end of the equation, the other pointer gives the list l_i under which the equation is then listed.

3.4 Applying 1-term equations

Sorting equations is beneficial if equations vary much in size. This is especially the case for selection systems with many equations having only one term.

But not all the variables that take zero value in the solution need to appear in 1-term equations in the original system. Many 1-term equations may result only after the first 1-term equations are applied. Still, during the solution process many 1-term equations are generated which justifies their special treatment.

What makes 1-term equations special is the fact that replacing a variable by zero in a polynomial (or linear) expression can be done very fast without re-writing the expression, at least in a lisp-representation: the pointer from the previous term to the term that vanishes is simply re-directed to the next term. Also, no re-simplification is necessary. In contrast, already the application of 2-term equations requires afterwards simplifications as the term resulting from substitution may combine with other terms.

To set many variables in an expression to zero in a most efficient way, the value cell of the variable was set to nil. Testing a variable for a zero value can then be done by just testing whether the variable is bound. After this is done for all known vanishing variables, the (large linear) expression in these variables is pruned only once by the special low level routine `PruneZeros`. The pruning is done 'in place' avoiding copying of the whole expression and also reducing future garbage collections.

All efficiency improving measures introduced in the sections 3.2, 3.3, 3.4 are applied in the procedure LSSS [9] but can also be performed alone on given expressions.

In the following subsections techniques are described that make the formulation of the linear system more efficient, or avoid the formulation of large selection systems.

3.5 Selective splitting of equations

The following steps are all computationally expensive:

- the formulation of two large symmetry conditions (4),
- their separation (called 'splitting' in the following) into two large linear systems with many redundant equations by setting coefficients of different products of powers of u, v separately to zero, and
- millions of simplifications of the large linear systems due to millions of vanishing variables.

The idea is to avoid most of these computations by

- extracting selectively only 1-term equations from the symmetry conditions (4), and
- using them to simplify the symmetry conditions themselves, and

- to repeat this procedure as long as 1-term equations can be found (see section 3.7).

Finally, the much smaller symmetry conditions are completely split and the resulting linear system is solved using LSSS as described in the previous three subsections.

The only remaining large step is the initial formulation of the symmetry conditions. One can save half of that computation by formulating only one symmetry condition, extracting from it and applying to it repeatedly 1-term equations, then pruning the ansatz for the symmetry (3) before computing the second symmetry condition.

Furthermore, even the formulation of the first symmetry condition can be postponed if short additional necessary conditions are known as described in the next subsection. From them 1-term equations, i.e. vanishing variables can be extracted and the ansatz for the symmetry (3) be pruned before any new symmetry condition is formulated.

The interplay between formulating and solving equations may be the only way to approach a large problem which otherwise could even not be formulated. For example, this iterative approach was necessary to compute symmetries of degree 14 on the available computer hardware with 128 GB memory under PSL REDUCE. The same approach was also applied in [11] to solve a non-linear system which was too large to be formulated at once.

The key to be able to selectively split up 1-term equations is a low level procedure written in SYMBOLIC MODE OF REDUCE that takes only two lines. It recursively steps through an expression and identifies 1-term coefficients of any monomial in u, v, u^{-1}, v^{-1}. If such a coefficient is found, it directly sets the value cell of the variable to nil (see section 3.4 for the case that a system of linear equations is given from which 1-term equations are picked).

3.6 Utilizing additional necessary conditions

Apart from a system of equations to be solved sometimes additional necessary conditions are available. The question is whether these conditions can be utilized to speed up the solution of the original system. If a program is performing its computation with the whole system that is to be solved at once then solving in addition extra conditions results in an increase of the total computation time. On the other hand, if the program is able to extract information from the system selectively then having extra necessary equations available means that the program has more options and may be able to solve the combined system faster than the original system alone.

If a program is able to generate repeatedly 1-term equations and to apply them then it is beneficial for the program to have extra necessary conditions (6a). Even if such opportunities do not exit, extra necessary conditions can already be useful if the length of equations varies much and if all equations are sorted by size and if system plus extra necessary conditions are very overdetermined.

3.7 Optimizing iterations

An optimal strategy to speed up computation is not to find and use as many as possible vanishing variables but to find and use as many as possible *per time*. This means one wants to find an optimum between formulating new conditions (6), (4) which each provide many vanishing variables but which take considerable time to formulate, or to utilize already formulated conditions by extracting again vanishing variables, utilizing them, extracting more, and so on. The second process is faster than formulating new conditions but the number of vanishing variables

runs	1	2	3	4	5	6	7	8
t_1	116	37.3	14.6	12.2	11.7	11.7	11.7	11.7
t_2	1.8	.49	.32	.26	.26	.25	.25	.25
new	5314423	1535271	288684	32076	3564	396	44	0

Table 3: Times and results for extracting 1-term equations repeatedly in the computation of symmetries of degree 13. t_1: time to formulate (6a) in run 1 and to prune it in later runs, t_2: time to 1-term-split (6a), new: number of new vanishing variables found in each run.

that are found is gradually decreasing as shown in table 3. The following comments refer to the computation of symmetries of degree 13.[3]

After extracting vanishing variables from the necessary condition (6a) one has the options to

- prune $D_\tau(u, v)$ (8 sec) and re-formulate (6a) (39 sec), or
- prune $D_\tau(u, v)$ (8 sec) and formulate the other necessary condition (6b) (39 sec)
- prune the already formulated condition (6a) (37.3 sec)

before extracting vanishing variables from that equation which was just formulated or pruned. Because in all 3 cases about 1,535,270 new vanishing variables are found, the third option is the fastest and most efficient one which is therefore repeated several times. As shown on table 3 the number of vanishing variables that are found, shrinks and the time to scan the condition stays constant, thus it pays off after three runs to invest in the formulation of a new condition

$$u_{\tau t} = u_{t\tau}. \tag{7}$$

After a first selective split of (7) it is most effective to prune and selectively split (6a) again, even twice before continuing to prune and split selectively (7). It turns out that formulating in addition the second symmetry condition

$$v_{\tau t} = v_{t\tau} \tag{8}$$

only for the purpose of selective splitting does not result in additional 1-term equations and is therefore not beneficial.

To summarize, denoting the pruning and selective splitting of necessary condition (6a) by n and the pruning and selective splitting of symmetry condition (7) by s, the sequence $n^3(snn)^4(sn)^4$ followed by a formulation of the symmetry condition (7), a complete splitting of (8) and complete splitting of what is left of (7) and a call of LSSS to solve the linear system brings down the total time for formulating and solving symmetry conditions for degree 13 to 467 sec, compared to 3615 sec when straightforwardly formulating and splitting (7), (8) and solving them with LSSS.

An additional benefit of utilizing 1-term equations consists in a drastic reduction of memory needs to only 13 GB for PSL Reduce or 7 GB for CSL Reduce (see section 5 about the differences between PSL and CSL Reduce) whereas the ad hoc formulation of the large symmetry conditions (7), (8) requires 120 GB in PSL Reduce or 60 GB in CSL Reduce for degree 13 symmetries. Without selective splitting of 1-term equations, i.e. determination of vanishing variables, it would not have been possible to determine symmetries of degree 15 on the computer with 128 GB and of degree 16 on a computer with 256 GB.

[3]Given times refer to the computation of degree 13 symmetries on a single CPU of a 8 core node (2 sockets x 4 cores per socket) Xeon @ 2.93 GHz using 48 GB memory of the node.

n	(A): stream solve		(B): sorting by size, stream solve		(C): 1-term equ., sorting by size, stream solve		(D): 1-term equ., sorting by size, stream solve	
	solve	subst.	solve	subst.	solve	subst.	solve	subst.
3	.02	.01	.01	.00	.00	.00	.01	0
4	.14	.01	.15	.02	.02	.02	.03	0
5	1.6	.17	1.3	.23	.20	.16	.08	0
6	20.8	1.46	15.3	2.4	1.7	1.3	.26	0
7	206	9.0	85	9.0	16.3	15.4	.77	0
8	2,740	82.5	808	92.8	241	223	2.7	.01
9	53,940	1,190	13,335	1,482	4,645	2,675	9.1	.01
10					60,140	20,360	30	.03
11							96	.07
12							302	.13
13							927	.34
14							2284	.42
15							7587	1.58
16							27970	3.16

Table 4: Times in sec for solving the symmetry conditions (4) for each degree n by different methods and for substituting the computed values into the symmetry ansatz (3).

3.8 Applying Solutions of large linear Systems

In applications the solution of a system of equations has usually to be substituted in expressions that are the main interest of the application. But millions of variables to be substituted in expressions with millions of terms is expensive. In such situations the labelling of zero-valued variables and the pruning of large expressions containing them becomes useful again.

In table 4, column (D) subst. shows drastically reduced substitution times of the solution into the symmetry ansatz (3) compared to column (C) subst. After the computation in column (D) no explicit substitutions are necessary. Any expression containing the unknowns needs only a) to be pruned to drop terms involving zero-valued variables and b) be simplified where non-zero variables get replaced by their computed value.

4 Results

The impact of measures described in the previous section is shown in table 4. The entries in columns with the header 'solve' give the time in sec to solve the linear algebraic system resulting from splitting the symmetry condition (4) (denoted in the following by $D_{[t,\tau]}(u,v) = 0$) wrt. monomials in u, v, u^{-1}, v^{-1}. Columns with the header 'subst.' show the time to substitute the solution into the symmetry ansatz (3).[4]

[4]Columns (A)-(C) have been run on one Opteron 2.2 GHz core of a 32 core node (8 sockets x 4 cores per socket) with 128 GB memory. In column (D) $n = 3..15$ were run on one Xeon 2.4 GHz core of a 16 core node with 128 GB memory and $n = 16$ was run on a single core of a machine equipped with 4 AMD Opteron processors each having 12 cores running at 2.2.GHz and with 256 GB of main memory. Computation times turned out to be strongly dependent on the load on the remaining cores of the node and the other nodes of the cluster. The

	(E) $D_{[t,\tau]}(u,v)=0$ at once	(F) $D_{[t,\tau]}(u,v)=0$ in two stages	(G) $D_\tau I=0$ first	(H) Iteration first
formulation of ansatz for D_τ	47.7	47.7	47.7	47.7
multiple runs to find vanishing variables (see section 3.7)	–	–	–	777
formulation and extraction of vanishing variables once from necessary condition (6a)	–	–	238.7	–
formulation and complete splitting of $D_{[t,\tau]}u=0$	1624	1624	621.3	4.5
extraction of vanishing variables from complete set of conditions $D_{[t,\tau]}u=0$ once	–	101.7	64.5	0.28
formulation and complete splitting of $D_{[t,\tau]}v=0$	1274	46.9	46.7	1.77
complete solution of all remaining equations	365	125	142	92.7
substitution of solution in $D_\tau(u,v)$	2.95	0.39	0.35	0.18
total time (rounded)	3314	1946	1161	924

Table 5: Times in sec of the whole symmetry computation for $n=13$

As described in the previous section, column (A) was produced with the original stream solver [8]. Sorting equations by size results in a speedup of a factor up to 4 shown in column (B) and column (C) shows the times when 1-term equations are applied first as described in section 3.4. Computations in columns (A)-(C) use standard substitutions and in column (C) the explicit formulation of 1-term equations and complete splittings of equations. The drastic improvement shown in column (D) became possible through direct detection of vanishing variables (specialized partial splitting), repeated partial splitting, assigning nil to the value cell of a variable and thus avoiding substitutions and enabling fast pruning of expressions from zero variables.

Table 5 compares four runs of the $n=13$ case.[5] In all these computations the program LSSS is used to solve the linear algebraic systems (the row 'complete solution of all remaining equations'). LSSS repeatedly applies 1-term equations and sorts the remaining equations by size before starting the stream-solver. What table 5 shows is the benefit of generating and solving the linear algebraic system in stages. With the availability of LSSS as an effective solver of large selection systems, the main cost shifted to the formulation of the linear system.

In the first column the whole system is formulated and solved at once. In the second column at first $D_{[t,\tau]}u=0$ is formulated, then 1-term equations are extracted and utilized to simplify the ansatz for $D_\tau u$ and $D_\tau v$ before $D_{[t,\tau]}v=0$ is formulated and the complete system is solved. This provides a speedup of nearly 1.7 for $n=13$.

times in column (D) are conservative times which are always reproducible. Sometimes computations have been up to 30% faster than shown in column (D).

[5]All times in this table include CPU time and garbage collection time of CSL REDUCE running on one of 16 cores (4 sockets x 4 cores per socket) Xeon 2.4 GHz node with 128 GB memory.

The third column gives the times when the whole computation is done in 3 stages. In the first step the necessary condition (6a): $D_\tau I = \sum_{k=-k_0}^{k_0} a_k I^k$ is formulated, and once vanishing variables are extracted which are used to simplify the symmetry ansatz (3) and speed up the formulation of the system. Although that first step is an additional computation not performed in the first two runs costing an extra 238 sec, this is more than compensated afterwards by the speedup of computing and solving $D_{[t,\tau]}(u, v) = 0$ (also in two stages) leading to another overall speedup factor of about 1.7.

The fourth column reports the times when at first a sequence of runs is performed where vanishing variables are extracted and set to nil as explained in section 3.7. This leads to another modest overall speedup of 1.25 but the main advantage of the computation in the fourth column is to lower the maximum memory requirements that occur when the first half of symmetry conditions is formulated.

The next section describes how differences between two versions of the computer algebra system REDUCE become important for large computations.

5 REDUCE issues

The open-source computer algebra system REDUCE can be run on two different LISP implementations. They are PSL (Portable Standard Lisp) and CSL (Codemist Standard Lisp).

The computations of this article require a very large number of identifiers. CSL has no restriction on the number of identifiers that can be used, whereas PSL is usually limited to 65,000 identifiers. To run the symmetry computations reported in this paper in PSL, an extended version of PSL REDUCE had been developed by Winfried Neun that allows for 20 Mio identifiers. This extended version of PSL can be downloaded for free [10].

When applied straightforwardly, PSL REDUCE typically runs twice as fast as CSL REDUCE. However, CSL allows for compilation of critical routines into C. When this is done, CSL runs in most cases as fast as PSL or even faster. Compilation into C can be achieved by creating a package and making a profiling run followed by a rebuild of the CSL-Reduce system.

A big advantage of CSL is its garbage collector. CSL uses both a copying and a mark-sweep collector. Copying garbage collectors have little overhead but can give only half of the memory to the application as the other half is needed for copying. A mark-sweep garbage collector allows the use of all of the memory by the application. In CSL the copying collector is used as long as the memory pressure is low, but switches to mark-sweep when the requirement for more memory rises. This way, CSL leverages performance and available memory. This feature of CSL made it possible to ultimately compute symmetries of degree 15 with 115 GB of memory and symmetries of degree 16 on a different machine with 256 GB.

6 Comparison with MAPLE

In this section we report on applying the computer algebra system MAPLE 14 on solving our linear systems. Table 6 shows times[6] for solving the symmetry conditions (4) with the MAPLE standard `solve` command in column (I) and with the `SolveTools:-Linear` command in column (J) and the times for solving the combined systems of additional conditions (6) and symmetry conditions (4) in columns (K) and (L).

[6]Computations were run on one Xeon 2.4 GHz core of a 16 core node with 128 GB memory

n	(I) s solve	(J) s tools	(K) ns solve	(L) ns tools
3	.0036	.024	.035	.027
4	.11	.089	.11	.085
5	.38	.20	.3	.23
6	1.1	.86	1.2	.80
7	4.3	2.97	4.9	3.75
8	20.4	13	24	16.9
9	128	73	162	81.3
10	792	262	929	294
11	7560	871	8615	972
12	DNF	3527	DNF	3232
13		11786		11840

Table 6: MAPLE 14 times

n	degree of the symmetry
s solve:	solving the system (4) with the `solve` command
s tools:	solving the system (4) with the `SolveTools:-Linear` command
ns solve:	solving both systems (6) and (4) with the `solve` command
ns tools:	solving both systems (6) and (4) with the `SolveTools:-Linear` command
DNF:	does not finish in one week

We see that for small systems, `solve` and `SolveTools:-Linear` take similar times. For larger systems ($N = 11$) `SolveTools:-Linear` is 9 times as fast and even bigger systems could not be solve with the command `solve` within a week.

Another observation is that neither `solve` nor `SolveTools:-Linear` can take advantage of additional necessary equations except for $n = 12$ in column (L) to a small extent.

The best times for MAPLE 14 in column (J) of table 6 and LSSS (REDUCE) in column (D) in table 4 have been measured on the same nodes of the same cluster and can be compared. To explain the 11786/927=12.7 times faster performance of LSSS for symmetries of degree 13 the following test has been made. Table 7 shows the times in sec for MAPLE 14 procedure `SolveTools:-Linear` to solve

- the unsimplified system (4) in column (M) (=column (J) of table 6),
- the same system simplified after 1-term equations have been solved repeatedly in column (N),
- and then in addition all equations being sorted by size, shortest first in column (O).

The cost of these simplifications using the REDUCE procedures FINDZEROS to find and utilize all 1-term equations repeatedly and LENGTHSORT to sort the remaining system by size is shown in column (Q). The rightmost column gives the factor of speedup of MAPLE if it would use at first the REDUCE procedures to simplify the system before solving it. The potential speedup of MAPLE is impressive and increasing with increasing size of the system.

n	(M) unsimp.	(N) unsorted	(O) sorted	(P) read simp.	(Q) simplify	(M/(O+P+Q))
3	.024	.0012	.0014	.010	0	2.1
4	.089	.0084	.0084	.012	.01	2.9
5	.20	.011	.0092	.012	.01	6.4
6	.86	.036	.037	.016	.03	10
7	3.0	.047	.047	.021	.09	19
8	13	.19	.22	.037	.31	23
9	73	.41	.38	.060	1.04	49
10	262	2.6	2.7	.118	3.44	42
11	871	4.7	4.9	.228	11.2	53
12	3527	32	31	.443	31.7	56
13	11786	78	71	.940	87.5	74
14		402	387	1.574		

Table 7: MAPLE 14 times for pre-simplified systems

7 Comparison with LINBOX

As described on its web page [13] the computer algebra system LINBOX is a C++ template library for exact, high-performance linear algebra computations with dense, sparse, and structured matrices over the integers and over finite fields.

Because the solution of our linear system (4) contains free parameters, LINBOX can not be applied straightforwardly to solve our system. However, what can be computed easily is the rank of these systems. Table 8 compares the best times of MAPLE solving the system (4) (column (J) in table 6) with the best times of REDUCE achieved when solving (4) together with the additional necessary conditions (6) and with LINBOX at least determining the dimension of the nullspace of (4). The timings of LINBOX include reading of the sparse coefficient matrix of the linear system which is the only way to enter data into LinBox.

8 The procedures

The following four procedures are available for the formulation and solution of linear algebraic system with an emphasis on selection systems which have many zero variables in their solution.

The procedure LSSS can solve linear systems with an arbitrary number of equations and unknowns, i.e. the linear system can be under- or overdetermined. In case the linear system is inconsistent a message will be printed. The unknowns to be solved for must be ordered ahead of the rest of the variables. The coefficients of the linear variables can be rational functions of parameters.

The procedure FINDZEROS repeatedly picks all 1-term equations from an input list of equations and sets the value cell of the vanishing variables to nil. It returns a list of vanishing variables and list of remaining equations.

The procedure LENGTHSORT effectively sorts a list of equations by their size.

The procedure PRUNEZEROS drops all terms of the input expression which should be zero due to an earlier run of LSSS or FINDZEROS.

n	MAPLE 14	LINBOX			REDUCE
	SolveTools	default	sparse		LSSS
	Sym	Sym	Sym+FI	Sym	Sym+FI
3	.024				.01
4	.09	.02	.02	.02	.03
5	.20		.12	.13	.08
6	.86	30.6	1.1	.90	.26
7	3		14.9	12.9	.77
8	13	3080	283.5	210	2.7
9	73		2318	1812	9.1
10	262		21610	21210	30
11	871				96
12	3527				302
13	11786				927
14					2284
15					7587
16					27970

Table 8: A comparison of times of MAPLE, LINBOX and REDUCE

The procedures LSSS, FINDZERO and LENGTHSORT are freely available from [9] where more details about syntax are given. These procedures are also included as module in the open source computer algebra system REDUCE. The server at [9] contains also files for all linear systems for symmetries of degree 3 to 15 in REDUCE format and in LINBOX format which can also be read by MAPLE.

9 Summary

A class of sparse linear systems that allows efficient solution strategies and that occurs frequently, for example, in integrability investigations, is characterized by the vanishing of many of the unknowns in its solution. In association with the purpose of such systems we call them selection systems. A series of such systems that we investigated results from symmetry investigations of a non-abelian ODE system of Kontsevich. It is demonstrated how the vanishing of many variables of a selection system can not only be used to speed up the solution of the system but also to avoid formulating most of the system and to apply the solution of the system to large expressions. It is demonstrated how a pre-processor step in which repeatedly 1-term equations are identified and applied and afterwards the remaining equations are sorted by size could speed up MAPLE routines that are specialized in sparse systems by a considerable factor.

Acknowledgement

The first author would like to thank Winfried Neun for providing a PSL version for large numbers of identifiers. The second author would like to thank Arthur C. Norman for very helpful discussions concerning CSL. Computations were run on computer hardware of the Sharcnet consortium (`www.sharcnet.ca`).

References

[1] Kontsevich, M., private communication.

[2] Olver, P.J. and Sokolov, V.V., Integrable evolution equations on associative algebras, *Comm. in Math. Phys.*, 1998, **193**, no.**2**, 245-268.

[3] Mikhailov, A.V. and Sokolov, V.V., Integrable ODEs on Associative Algebras, *Comm in Math Phys.*, **211**, 231-251, 2000.

[4] Efimovskaya, O.V., Integrable cubic ODEs on Associative Algebras, *Fundamentalnaya i Prikladnaya Matematika*, **8**, no. **3**, 705-720, 2002.

[5] Wolf, T. and Efimovskaya, O., On integrability of the Kontsevich non-abelian ODE system, accepted for publication in Lett in Math Phys, 9 pages, DOI: 10.1007/s11005-011-0527-4 and `http://lie.math.brocku.ca/twolf/papers/EfWNew11.pdf` (2011).

[6] Olver, P.J., Applications of Lie Groups to Differential Equations, Second Edition, Graduate Texts in Mathematics, **107**, *Springer–Verlag, New York*, 1993.

[7] REDUCE - A portable general-purpose computer algebra system, free download site: `http://reduce-algebra.sourceforge.net/`, 2009.

[8] The program `streamsolve.red` for solving linear algebraic systems in REDUCE (2007), `http://lie.math.brocku.ca/papers/TsWo2007/`.

[9] The program `LSSS.red` for solving linear algebraic selection systems in REDUCE (2011), `http://lie.math.brocku.ca/papers/LSSS/`.

[10] Neun W.: The computer algebra system REDUCE with an extension allowing 20M identifiers: `www.zib.de/Symbolik/reduce/twentyM.zip`

[11] Tsarev, S.P. and Wolf, T.: Classification of 3-dimensional integrable scalar discrete equations, Lett in Math Phys, DOI:10.1007/s11005-008-0230-2, also arXiv:0706.2464, (2008).

[12] Pearce, R.: Solving Sparse Linear Systems in Maple, `http://www.mapleprimes.com/posts/41191-Solving-Sparse-Linear-Systems-In-Maple`, source code at: `http://www.cecm.sfu.ca/~rpearcea/sge/sge.mpl` (2007).

[13] Project LinBox: Exact computational linear algebra, `http://www.linalg.org/`