





# Symbolic and Numerical Methods for Searching Symmetries of Ordinary Differential Equations with a Small Parameter and Reducing Its Order

Alexey A. Kasatkin<sup>(✉)</sup>  and Aliya A. Gainetdinova 

Ufa State Aviation Technical University, Ufa, Russia  
kasatkin@ugatu.su

**Abstract.** Two programs for computer algebra systems are described that deal with Lie algebras of generators admitted by systems of ordinary differential equations (ODEs). The first one allows to find the generators of admitted transformations in the specified form. This program is written in Python and based on SciPy library. It does not require solving partial differential equations symbolically and can also analyze equations with Riemann–Liouville fractional derivatives and find approximate symmetries for systems of equations with a small parameter. The second program written as a package for Maple computes the operator of invariant differentiation in special form for given Lie algebra of generators. This operator is used for order reduction of given ODE systems.

**Keywords:** Lie algebras of generators ·  
Point symmetries of differential equation · Fractional derivatives ·  
Differential invariants · Operators of invariant differentiation ·  
Computer algebra

## 1 Introduction

Group analysis of differential equations (see, e.g., [1–4]) provides efficient analytical methods for studying and solving differential equations, which arise in different areas of science.

In this paper, we consider the systems of ordinary differential equations (ODEs), focusing on nonlinear ones. The Lie symmetry method for solving nonlinear ODE is one of the most universal methods because it can work for equations not matching any particular form. Each symmetry of equation (or system of equations) describes the one-parametric group of point transformations that leaves the equation (or system) invariant. When the Lie algebra of symmetry generators is found, it can be used to reduce the order of the given equation

---

Supported by the Ministry of Science and High Education of the Russian Federation (State task No. 1.3103.2017/4.6).

or system. The order reduction algorithm for single equation is well-known and Lie method is implemented and often used by default for solving ODEs in computer algebra software like Maple and SymPy Python library.

However, the modifications of order reduction algorithm for systems of ODEs (see, e.g., [5, 6]) are not universal. Recently, a new order reduction algorithm was proposed in [7, 8]. It uses the operator of invariant differentiation (OID) with multiplier of special type. In [9], a modification of this algorithm is proposed for application to systems of ODEs with a small parameter that admit approximate Lie algebras of generators (theory of approximate Lie groups is introduced in [4]). To simplify the calculations, a program for constructing an OID with a multiplier of a special type was developed in Maple computer algebra system (Maple was chosen because it has a powerful solver module for partial differential equations). In Sect. 3.2, we describe this program and illustrate its usage with some examples.

Methods based on symmetries have been recently applied to classes of equations with fractional derivatives [10–12]. Systems of ordinary fractional differential equations (FDEs) are often considered in modern control system theory and mathematical models of different processes. Such systems are also considered when finding particular solutions of fractional partial differential equations by invariant subspace method [13–15]. Known symmetries allow one to construct invariant solutions of the considered systems. However, there are only few works on symmetries of ordinary FDEs and systems, for example [16, 17], there are many technical difficulties and limitations because the fractional derivatives are non-local integro-differential operators. For FDEs, the current method of constructing symmetries is restricting the form of operators and the resulting Lie algebra is usually formed by combinations of some standard generators (usually corresponding to translation, scaling, rotation and projective transformations).

Constructing symmetries for the considered systems requires a lot of calculations. To find the symmetries of equation or a system, one needs to solve the so-called determining equations, which are the linear first-order partial differential equations (PDEs) [1–4, 11]. There are multiple existing packages for finding Lie symmetries of different types [18–20] (some of them support finding approximate symmetries), and one for equations with fractional derivatives [21]. They are usually based on powerful first-order PDE solvers but there is no guarantee that the symmetry generators can be found in closed form for arbitrary form of system, for example, including functions of derivatives.

The program described in Sect. 3.1 of this paper uses a semi-numerical approach for finding symmetries. Symmetry generators are found as linear combinations of given basis operators  $X_i$ . This approach is usually implemented for polynomial form of coefficients. For example, it is available inside Maple PDEtools package and in SymPy library for single equation. Recent application for dynamical systems is also described in [22]. However, in this work, we do not restrict coefficients to have polynomial form. The numerical algorithms like SVD decomposition are used to find the space of solutions for automatically constructed determining equations. This approach allows to work with very

complicated forms of nonlinear systems. Symbolic calculations are only used to differentiate equation one time, substitute the coefficients of given operators  $X_i$  into prolongation formula and collect the terms in linear expressions, the final steps of computation are numerical.

Another advantage of using the fixed-form representations of symmetry generator is the possibility to analyze symmetries of the first-order differential equations that are used in many mathematical models. Finding the symmetries analytically in this case is a complicated task because the space of symmetry generators is usually infinite-dimensional. Restricting the form of infinitesimal generator (for example, by assuming polynomial coefficients) allows one to construct solutions or conservation laws even without finding the complete set of symmetries.

## 2 Theoretical Section

### 2.1 Point Symmetries of Differential Equations

Let us consider the system of ordinary differential equations

$$u^{(p)}(t) = f\left(t, u(t), \dots, u^{(p-1)}(t)\right), \tag{1}$$

where  $u = u(t)$  is a vector of  $m$  unknown functions  $u_1(t), \dots, u_m(t)$ ,  $f$  is a vector function with components  $f_\mu$  and  $u^{(k)}$  is a vector of  $k$ th derivatives of  $u$  by  $t$ .

To define the symmetry of (1), the local Lie group of point transformations  $T_a$  with parameter  $a$  is also considered:

$$T_a : (t, u) \rightarrow (\tilde{t}, \tilde{u}), \quad \tilde{t} = \varphi(t, u; a), \quad \tilde{u} = \psi(t, u; a), \quad T_{a+b} = T_a T_b. \tag{2}$$

The group of transformations can be defined by its generator

$$X = \xi(t, u) \frac{\partial}{\partial t} + \eta(t, u) \frac{\partial}{\partial u}, \tag{3}$$

where functions  $\xi(t, u) = \left. \frac{\partial \varphi}{\partial a} \right|_{a=0}$  and  $\eta(t, u) = \left. \frac{\partial \psi}{\partial a} \right|_{a=0}$  are the coordinates of the tangent vector field. To study the symmetry properties of system (1), we find the prolonged group acting in space, where the coordinates of some point are  $t, u$  and all derivatives of  $u$  up to the  $p$ th order. This group is defined by prolonged generator which has the form

$$X^{(p)} = \xi(t, u) \frac{\partial}{\partial t} + \sum_{\mu=1}^m \eta^\mu(t, u) \frac{\partial}{\partial u_\mu} + \sum_{\mu=1}^m \sum_{k=1}^p \zeta_k^\mu(t, u, \dots, u^{(k)}) \frac{\partial}{\partial u_\mu^{(k)}}, \tag{4}$$

where the coefficients  $\zeta_k^\mu$  are given by the prolongation formula

$$\zeta_k^\mu = D_t^k (\eta - \xi u'_\mu) + \xi u_\mu^{(k+1)}. \tag{5}$$

The coefficients  $\zeta_k^\mu$  describe the infinitesimal transformations of derivatives:

$$\tilde{u}_\mu^{(k)}(\tilde{t}) = u_\mu^{(k)}(t) + a\zeta_k^\mu + o(a).$$

Here  $D_t$  is the total derivative operator:

$$D_t = \frac{\partial}{\partial t} + \sum_{k=0}^p \sum_{\mu=1}^m u_\mu^{(k+1)}(t) \frac{\partial}{\partial u_\mu^{(k)}}.$$

The transformation group (2) is called *point symmetry* of system (1) if the system is transformed to itself. The group generator is said to be *admitted* by system (1). It is known [1–3] that ODE (1) admits generator (3), if and only if

$$X^{(p)} \left( u_\mu^{(p)} - f_\mu \left( t, u, \dots, u^{(p-1)} \right) \right) \Big|_{(1)} = 0, \quad \mu = 1, \dots, m. \tag{6}$$

From equation (6), after splitting with respect to powers of the derivatives, one can obtain a system of *determining equations*. It is a linear overdetermined system of partial differential equations for  $\xi, \eta$  coefficients. The solution of this system gives a complete set of generators (3) admitted by equation (1).

The set of generators of the form (3) together with operation of commutation

$$[X_1, X_2] = X_1(X_2) - X_2(X_1) \tag{7}$$

forms a Lie algebra of generators if it is a vector space and the commutator of every two operators belongs to the same space. The conditions of bilinearity antisymmetry and Jacobi identity are satisfied (see e.g. [3]). It is known that generators admitted by an equation or a system always form a Lie algebra.

In this work we consider the ODE systems, which order is equal to dimension of admitting Lie algebra  $L_r$  of generators, i.e.,  $r = mp$ .

### 2.2 Invariant Representation of Differential Equations

Some function  $J^{(k)} = J(t, u, \dots, u^{(k)})$ ,  $J^{(k)} \neq \text{const}$ , is  $k$ th-order differential invariant of  $r$ -dimensional Lie algebra of generators  $L_r$ , if

$$X_j^{(k)} \left( J^{(k)} \right) = 0, \quad j = 1, \dots, r. \tag{8}$$

We can rewrite system (1) by differential invariants of admitted Lie algebra  $L_r$ , if and only if

$$\text{rank} \left( \xi_j \ \eta_j \ \dots \ \zeta_j^{(p)} \right) \Big|_{(1)} = r, \quad j = 1, \dots, r,$$

i.e., the general rank of the matrix formed by the coordinates of the prolonged generators does not change on the manifold defined by system (1) (see e.g. [2]). The system can then be rewritten so that every equation has the form

$$J^{(p)} = F \left( J^{(q)} \right) \tag{9}$$

with some function  $F$ , here  $J^{(p)}$  and  $J^{(q)}$  are some  $p$ th-order and  $q$ th-order differential invariants of  $L_r$ ,  $q < p$ .

### 2.3 Order Reduction for Differential Equations

For order reduction, in [7,8] authors suggested to seek auxiliary function  $\Phi \neq \text{const}$  and construct the operator of invariant differentiation (OID) (see [2]) in the form

$$\frac{1}{D_t \Phi} D_t. \tag{10}$$

The function  $\Phi$  is obtained as any particular solution of system

$$D_t \left( X_j^{(p-1)}(\Phi) \right) = 0, \tag{11}$$

or

$$X_j^{(p-1)}(\Phi) = C_j, \quad j = 1, \dots, r, \tag{12}$$

where constants  $C_j$  satisfy the relations

$$\sum_{l=1}^r a_{ij}^l C_l = 0,$$

and  $a_{ij}^l$  are structural constants of Lie algebra  $L_r$  (see [1–3]).

Then by using this OID, one obtains

$$\frac{1}{D_t \Phi} D_t \left( J^{(q)} \right) \Big|_{(9)} \equiv \Psi \left( J^{(q)}, J^{(p)} \right) \Big|_{(9)} = H \left( J^{(q)} \right).$$

The last expression can be rewritten as a first-order ODE

$$\frac{dJ^{(q)}}{d\Phi} = H \left( J^{(q)} \right).$$

General solution of this equation is also the first integral of ODE (1).

A similar method can also be used to reduce the order and integrate the systems of ODEs [7,8].

### 2.4 Symmetries, Invariants, and OID for Equations with a Small Parameter

Studying equations with a small parameter, i.e., ODEs of the form

$$u^{(p)} = f_0(t, u, \dots, u^{(p-1)}) + \varepsilon f_1(t, u, \dots, u^{(p-1)}), \tag{13}$$

we can use the theory of approximate transformation groups (see e.g. [4]). We consider only the case of linearity on  $\varepsilon$ .

Such equations admit the generators of two types:

$$X_{(0)} + \varepsilon X_{(1)}, \quad \varepsilon Y_{(0)}.$$

They can be obtained by solving Eq. (6) after neglecting  $\varepsilon^2$  terms and splitting by  $\varepsilon$  together with derivatives. The generators form approximate Lie algebra

[4]. We assume that system (13) has  $r_0$  operators of the first type, and  $r - r_0$  operators of the second type.

Invariants of the approximate Lie algebra can also have two forms [24]:

$$J_{(0)}^{(k)} + \varepsilon J_{(1)}^{(k)}, \quad \varepsilon J_{(0)}^{(n)},$$

where the term  $J_{(0)}^{(s)}$  satisfies the equations

$$\begin{aligned} X_{i,(0)}^{(s)} J_{(0)}^{(s)} &= 0, \quad i = 1, \dots, r_0, \\ Y_{j,(0)}^{(s)} J_{(0)}^{(s)} &= 0, \quad j = 1, \dots, r - r_0, \end{aligned}$$

and  $J_{(1)}^{(s)}$  satisfies equations

$$X_{i,(0)}^{(s)} J_{(1)}^{(s)} + X_{i,(1)}^{(s)} J_{(0)}^{(s)} \approx 0.$$

In the work [9], the OID for the approximate Lie algebra was introduced. It is shown that it can be obtained in the form

$$\frac{D_t \Phi_0 - \varepsilon D_t \Phi_1}{(D_t \Phi_0)^2} D_t,$$

where the functions  $\Phi_0$  and  $\Phi_1$  are particular solutions of systems

$$X_{i,(0)}^{(p-1)} \Phi_0 = C_{i,(0)}, \quad Y_{j,(0)}^{(p-1)} \Phi_0 = C_{j,(0)}$$

and

$$X_{i,(0)}^{(p-1)} \Phi_1 + X_{i,(1)}^{(p-1)} \Phi_0 \approx C_{i,(1)},$$

respectively.

### 2.5 Symmetries of Equations with Fractional Derivatives

In the last decade, Lie group analysis methods were applied to the specific case of integro-differential equations – equations with fractional derivatives. The survey of proposed techniques and results may be found, for example, in [11, 12].

The Riemann–Liouville fractional derivative is a nonlocal operator defined as

$$D_t^\alpha u(t, x) = \frac{1}{\Gamma(n - \alpha)} \left( \frac{\partial}{\partial t} \right)^n \int_0^t \frac{u(z, x)}{(t - z)^{\alpha - n + 1}} dz, \quad n - 1 \leq \alpha < n. \quad (14)$$

Here  $x$  can be a vector of all other independent variables,  $n$  is a natural number.

Let the generator of point transformation group have the form

$$X = \xi^0(t, x, u) \frac{\partial}{\partial t} + \sum_{i=1}^n \xi^i(t, x, u) \frac{\partial}{\partial x_i} + \sum_{\mu=1}^m \eta^\mu(t, x, u) \frac{\partial}{\partial u_\mu}. \quad (15)$$

The infinitesimal transformation of fractional derivative  $D_t^\alpha u$  can be written as

$$D_t^\alpha \tilde{u}_\mu = D_t^\alpha u_\mu + a \zeta_\mu^\alpha + o(a).$$

The prolongation formula for the fractional derivative has the general form

$$\zeta_\alpha^\mu = D_t^\alpha \left( \eta - \frac{\partial u_\mu}{\partial t} \xi^0 - \sum_{i=1}^n \frac{\partial u_\mu}{\partial x_i} \xi^i \right) + \xi^0 D_t^{\alpha+1} u_\mu + \sum_{i=1}^n \xi^i D_t^\alpha \frac{\partial u_\mu}{\partial x_i}. \quad (16)$$

with the restrictions  $\xi^0|_{t=0} = 0$ .

The main problem here is that  $\zeta_\mu^\alpha$  does not depend on finite set of differential variables like it was for operator (4). To propose constructive algorithm for finding symmetries, we have to restrict generator coefficients, considering the *linearly autonomous class of symmetries* with

$$\xi^0 = \xi^0(x), \quad \eta^\mu = \sum_{\nu=1}^m p^{\mu,\nu}(t, x) u_\nu + q^\mu(t, x). \quad (17)$$

Then the prolongation formula (16) can be written in the form

$$\begin{aligned} \zeta_\alpha^\mu = D_t^\alpha q^\mu + \sum_{\nu=1}^m \sum_{k=0}^\infty \binom{\alpha}{k} \left( D_t^k p^{\mu,\nu} + \delta_\nu^\mu \frac{k-\alpha}{k+1} D_t^{k+1} \xi^0 \right) D_t^{\alpha-k} u_\nu + \\ + \sum_{i=1}^n \sum_{k=1}^\infty \binom{\alpha}{k} (D_t^k \xi^i) D_t^{\alpha-k} \frac{\partial u_\mu}{\partial x_i}. \end{aligned} \quad (18)$$

When only one independent variable  $t$  is considered, the last term in prolongation formula vanishes.

After writing the invariance criterion like (6), we consider all integer derivatives  $u_\mu^{(k)}$  and fractional derivatives or integrals  $D_t^{(\alpha-k)} u_\mu$  to be independent variables,  $m$  of them are to be substituted from the given system. This is another assumption that allows constructive symmetry finding.

The resulting system of determining equations is now infinite-dimensional. Usually, but not always, it becomes finite after finding that  $\xi^0$  is a polynomial function of  $t$ .

### 3 Algorithms and Computer Algebra Modules

#### 3.1 Determining Lie Symmetries

To simplify the procedure of calculating symmetries, the computer algebra systems are often used.

There are many programs for calculating symmetry generators for equations and systems developed using Maple, Mathematica, Reduce, and Maxima computer algebra systems, see, for example, [18]. Modern examples of such packages

implemented in Maple are GeM [19] and DESOLVEII [20]. The package FracSym [21] also supports fractional differential equations.

Here we describe a new package for finding symmetries based on free and open source Python SymPy library [25]. It is not based on any of the existing modules, only SymPy differentiation, substitution, and singular value decomposition from Numpy are used.

The program part that constructs the determining equation is analogous to other packages but includes specific modifications for using prolongation formulas suitable for fractional derivatives (18). Working with approximate symmetries [4] is also supported by automatic collecting powers of  $\varepsilon$  and eliminating terms with  $\varepsilon^2$ .

Note that SymPy does not seem to have built-in “jet notation” for derivatives, so it was implemented manually. For example, the symbol `u_tttt` describes the derivative  $u^{(4)}(t)$ . The notation for fractional differential variable was also introduced. SymPy symbols and expressions can be used in given system.

While constructing the determining equations, the program also detects fractional derivatives in original system and restricts generators to be linearly autonomous. For each fractional derivative, some unknown coefficient  $\eta^\mu$  is replaced by its form (17) and the number of independent variables in corresponding coefficient  $\xi^i$  is reduced.

The determining equations for system  $F = 0$  are constructed in general form

$$\left( \tilde{X}[\xi, \eta, t, u] F_\mu \right) \Big|_{F=0} = 0, \quad \mu = 1, \dots, m, \tag{19}$$

Here  $\tilde{X}[\xi, \eta, t, u]$  is the prolonged generator given by (4) including prolongations to derivatives of fractional order. It depends on functions  $\xi, \eta$  linearly but contains the arbitrary combinations of  $u$  and its derivatives with respect to independent variables  $t$ . SymPy built-in PDE solver is not powerful enough to solve this overdetermined linear system automatically, but this is not a problem for proposed approach. Instead of splitting the determining equations and solving them symbolically, we use semi-numerical method described below.

We search for the generator as a combination of given basis operators:

$$X = \sum_{i=1}^N C_i X_i.$$

This idea is commonly used for polynomial coefficients of symmetry generators implemented for single ODE in *sympy.solvers.ode.lie\_heuristic\_bivariate*, for example. Maple package PDETools also contains an option for polynomial symmetries finding. Similar method is described in the work [22]. However, here we do not require coefficients  $\xi_i, \eta_i$  of  $X_i$  to be polynomial. Arbitrary known terms can be included in the form of generator, which is a more universal way.

Introducing the notation  $\Psi_\mu[\xi, \eta, t, u] = (\tilde{X} F_\mu)|_{F=0}$ , one can rewrite Eq. (19) as a linear equation for  $C_1, \dots, C_N$ :

$$\sum_{i=1}^N C_i \Psi_\mu[\xi_i, \eta_i, t, u] = 0. \tag{20}$$



For each  $\mu$  and  $i$ ,  $\Psi_{\mu,i}[t, u] = \Psi_{\mu}[\xi_i, \eta_i, t, u]$  is a function of independent variables  $t$ , dependent variables  $u$  and their derivatives (integer or fractional).

The program substitutes each  $\xi_i, \eta_i$  (or  $\xi_i, p_i$ , and  $q_i$  for linearly autonomous symmetry) into the determining equation symbolically and builds the list of functions  $\Psi_{\mu,i}$  for each number of dependent variable  $\mu$ .

The most general way of reducing (20) to algebraic system is using multivariate Taylor series expansion of  $\Psi_{\mu,i}[t, u]$  with respect to all of its independent variables. However, to speed up calculations, the program can also substitute random arguments into (20) enough times to get overdetermined linear system for  $C_i$ . This substitution can be done numerically by using the fast NumPy library, all constants and arbitrary functions should be specified exactly at this stage. As a result, rectangular matrix  $M$  with  $R$  rows and  $N$  columns is obtained.

To solve the overdetermined system  $MC = 0$  numerically, it is convenient to use singular value decomposition procedure. After using `np.linalg.svd` function to compute the decomposition of the form

$$M = U\Sigma V, \quad \Sigma = \text{diag}\{\sigma_1, \dots, \sigma_r, 0, \dots, 0\}$$

with orthogonal matrices  $U, V$  and  $r$  non-zero singular values  $\sigma_1, \dots, \sigma_r$ , the fundamental set of solutions  $C$  is given by last rows of matrix  $V$  (with numbers starting from  $r + 1$ ), see [23].

Every found vector  $C$  describes the symmetry generator, but to present the symmetries in convenient form, the matrix formed by coefficients  $C_i$  is transformed to reduced row echelon form. The basis columns are chosen automatically to maximize the number of coefficients close to zero. SymPy function `nsimplify` is then called to guess rational fractions like  $1/7$  and roots like  $\sqrt{2}$  from the corresponding numerical values.

Hereafter we assume that our module and SymPy are imported as follows:

```
> import symmetries as s
> from sympy import *
```

All commands entered by user are marked by starting `>` sign, lines after code without the sign shows its output converted to  $\text{\LaTeX}$ . The superscript with dependent variable symbol is used in the program instead of dependent variable number, for example,  $\xi^t = \xi^0$ ,  $\eta^u = \eta^1$ ,  $p^{uv} = p^{1,2}$  in (15) and (17).

The process of calculating symmetries contains several steps described below.

1. Procedure `LieSymmSetup(indepsList, depsList)` is used to define the lists of independent and dependent variables. For example, to study the ODE systems with  $u(t)$  and  $v(t)$ , one should call

```
> s.LieSymmSetup(indepsList=['t'], depsList=['u', 'v'])
```

2. Then the user defines the form of the system. The system can be presented directly as Python dictionary with strings or SymPy objects for left-hand side and right-hand side of equations, for example,

```
> sys={'u_t': 'v_xx-u', 'v_t': 'u_xx-v'}
```

- The function `detPDES` is used to compose the determining Eq. (19). For example, determining equation for  $u''(t) = u'(t)$  is calculated as  $(\zeta_2 - \zeta_1)|_{u''=u'}$ :

```
> s.LieSymmSetup(indepsList=['t'], depsList=['u'])
> XFS=s.detPDES({'u_tt': 'u_t'}, UseJetVars2=True)
```

$$\eta_t^u - \eta_{tt}^u - 2\eta_{tu}^u u_t - \eta_{uu}^u u_t^2 + u_t^3 \zeta_{uu}^t + 2u_t^2 \zeta_{tu}^t + 2u_t^2 \zeta_u^t + u_t \zeta_t^t + u_t \zeta_{tt}^t$$

To find approximate symmetries, one should call the function `detPDE_Approx`. When the equation contains fractional derivatives, the function `prolongUn` that implements prolongation formula (18) is used internally. The named parameter `UseJetVars2=True` is used here to show result in the short form and should not be used for further calculations.

- The list `BaseOps` of basis operators  $X_i$  is constructed to specify the chosen form of symmetry generator. Operators can be manually added to `BaseOps` list or chosen in multivariate polynomial form by `AddPolynomialBasis` function as shown in the examples.
- The `build_C_Coeffs(XFS)` substitutes all basis operators into constructed determining Eq. (19) and calculates coefficients `XFS_C_coeffs` of system (20) in symbolic form.
- Using the procedure `solveNumeric`, overdetermined algebraic system for coefficients  $C_i$  is constructed from (20) and solved numerically. The best combinations of basis operators are chosen and stored as `BestForm` matrix. To view the symmetries in convenient form looking like differential operators, one calls the function `getAdmittedOperators`.

*Example 1.* Let us consider the system of fractional differential equations

$$D_t^\alpha u = uv, \quad D_t^\alpha v = v^2.$$

According to [17], any symmetry of (1) is a combination of basis operators

$$X_1 = u\partial_u, \quad X_2 = v\partial_v, \quad X_3 = u\partial_v, \quad X_4 = v\partial_u, \quad X_5 = t\partial_t, \quad X_6 = t^2\partial_t.$$

To construct the determining equations (steps 1–3), we use the code

```
> s.LieSymmSetup(indepsList=['t'], depsList=['u', 'v'])
> alpha=sympify('1/5');
> dau = s.fracD('u', 't', alpha); dav = s.fracD('v', 't', alpha)
> XFS=s.detPDES({'dau': 'u*v', 'dav': 'v*v'}, sumN=2)
```

The function `s.fracD('u', 't', alpha)` gives  $Da_t[u](1/5)$ , which is the representation of  $D_t^{1/5}u$  (differential variable in the code). The determining equations stored in `XFS` contain multiple terms, one of them is calculated internally as fractional prolongation by calling `s.prolongUn('u', 't', alpha, sumN=2)`

$$p^{uu} Da_t[u] \left(\frac{1}{5}\right) + \frac{p_t^{uu}}{5} Da_t[u] \left(-\frac{4}{5}\right) - \frac{2p_{tt}^{uu}}{25} Da_t[u] \left(-\frac{9}{5}\right) +$$

$$p^{uv} Da_t[v] \left(\frac{1}{5}\right) + \frac{p_t^{uv}}{5} Da_t[v] \left(-\frac{4}{5}\right) - \frac{2p_{tt}^{uv}}{25} Da_t[v] \left(-\frac{9}{5}\right) - \frac{\xi_t^t}{5} Da_t[u] \left(\frac{1}{5}\right) + \frac{2\xi_{tt}^t}{25} Da_t[u] \left(-\frac{4}{5}\right) - \frac{6\xi_{ttt}^t}{125} Da_t[u] \left(-\frac{9}{5}\right) + Da_t[q^u] \left(\frac{1}{5}\right)$$

with substituted  $Da_t[u] = uv, Da_t[v] = v^2$ . The parameter `sumN` restricts the number of considered terms in prolongation formula (18). Note that no simplifications are done automatically to speed up calculations. During the following steps,  $t, u$ , and  $v$  and all other remaining fractional derivatives and integrals of  $u$  and  $v$  are considered as independent variables, see [11].

Using the code with provided basis operators

```
> s.BaseOps=['u*D_u', 'v*D_u', 'u*D_v', 'v*D_v', 't*D_t', 't**2*D_t']
> s.build_C_Coeffs(XFS)
> display(s.XFS_C_coeffs)
```

one gets the lists of  $\Psi_{\mu,i}$  stored as `XFS_C_coeffs` that define pairs of equations  $\sum C_i \Psi_{1,i} = 0, \sum C_i \Psi_{2,i} = 0$  for  $C_i$ :

$$\left[ \left[ 0, 0, u^2, uv, \frac{uv}{5}, \frac{2t}{5}uv - \frac{4}{25}Da_t[u] \left(-\frac{4}{5}\right) \right], \right. \\ \left. \left[ 0, 0, uv, v^2, \frac{v^2}{5}, \frac{2t}{5}v^2 - \frac{4}{25}Da_t[v] \left(-\frac{4}{5}\right) \right] \right].$$

Zeros in the first and second columns mean that  $C_1$  and  $C_2$  can be arbitrary constants and  $X_1, X_2$  are symmetry generators. All combinations of basis operators that form symmetries are found by constructing and solving the overdetermined algebraic system (step 6):

```
> s.solveNumeric()
> display(s.getAdmittedOperators())
9.977099577364706
Shape of matrix of overdetermined system for finding C_k:
(22, 6), doing SVD decomposition
Singular values:
[ 3.36e+00 1.07e+00 3.50e-01 3.41e-17 9.30e-18 -0.00e+00]
Finding best combinations of 3 basis operators...OK
```

The resulting 3 operators are found correctly, the code prints

$$[uD_u, vD_u, -5tD_t + vD_v].$$

*Example 2.* Let us consider equation from [16] that contains fractional derivatives of different orders:

$$D_x^{\alpha+1}y = -\frac{\alpha+1}{x}D_x^\alpha y + y \left(\frac{D_x^\alpha y}{y}\right)^{\frac{\alpha+1}{\alpha}}.$$

```

> s.LieSymmSetup(indepsList=['x'], depsList=['y'])
> x,y = symbols('x y'); alpha = sympify('1/3')
> s.BaseOps=[]; s.AddPolynomialBasis(3)
> DAY = s.fracD('y','x',alpha); DAY1 = s.fracD('y','x',alpha+1)
> XFS = s.detPDES({ DAY1 : -(alpha+1)*DAY/x +
    y*(DAY/y)**((alpha+1)/alpha)})
> s.build_C_Coeffs(XFS);s.solveNumeric();s.getAdmittedOperators()

```

All 3 admitted operators are found correctly:

$$\left[ xD_x, \quad x^2D_x - \frac{2x}{3}yD_y, \quad yD_y \right].$$

Note that although the right-hand side depends on fractional derivative, it causes no problems for the code because we do not solve partial differential equations analytically.  $\square$

*Example 3.* The method also works when operators contain non-polynomial terms. For equation

$$D_x^\alpha y = x^{-1-\alpha} e^{yx^{1-\alpha}}$$

the symmetries are found correctly and include the term  $x^{\alpha-1}\partial_y$ :

```

> s.LieSymmSetup(indepsList=['x'], depsList=['y'])
> x,y = symbols('x y'); alpha = sympify('1/3')
> s.BaseOps=[]; s.AddPolynomialBasis(2)
> s.BaseOps.append('x**(1/3-1)*D_y');
> s.BaseOps.append('x**(1/3)*D_y'); DAY = s.fracD('y','x',alpha)
> XFS = s.detPDES({DAY: x**(-1-alpha)* exp(y*x**(1-alpha))})
> s.build_C_Coeffs(XFS);s.solveNumeric();s.getAdmittedOperators()

```

$$\left[ x^2D_x - \frac{2x}{3}yD_y, \quad 3xD_x + \left( -2y + \frac{1}{x^{\frac{2}{3}}} \right) D_y \right].$$

Note that the program should be able to automatically calculate  $D^\alpha q$  for given functions  $q(t)$  by using its internal `fractional.D_RL` function.  $\square$

*Example 4.* In the work [15], the system of fractional differential equations

$$D_t^\alpha a = \mu b^2/2, \quad D_t^\alpha b = 2\mu bc, \quad D_t^\alpha c = 2\mu c^2$$

is considered. It is obtained when searching for exact solutions of the time fractional Korteweg-de Vries equation

$$D_t^\alpha u = \frac{\mu}{2} \left( \frac{\partial u}{\partial x} \right)^2 + \frac{\partial^3 u}{\partial x^3}$$

by invariant subspace method [13], using the form of solution

$$u(t, x) = a(t) + b(t)x + c(t)x^2.$$

The program successfully finds 3 symmetries for arbitrary order  $\alpha$  and 4 symmetries for order  $\alpha = 1/3$ :

```

> t,a,b,c = symbols('t a b c')
> alpha = sympify('1/3'); mu = sympify('7/8');
> s.LieSymmSetup(indepsList=['t'], depsList=['a','b','c'])
> dax = s.fracD('a','t',alpha); day = s.fracD('b','t',alpha);
> daz = s.fracD('c','t',alpha);
> XFS = s.detPDES({dax: mu*b*b/2, day: mu*2*b*c, daz: mu*2*c**2 })
> s.BaseOps = []; s.AddPolynomialBasis(2); s.build_C_Coeffs(XFS)
> s.solveNumeric(); display(s.getAdmittedOperators())

```

The displayed symmetries are

$$\left[ bD_a + 2cD_b, \quad aD_a - cD_c + 3tD_t, \right. \\ \left. atD_a + btD_b + ctD_c - \frac{3t^2}{2}D_t, bD_b + 2cD_c - 6tD_t \right].$$

It can be shown similarly to [17] that all of the linearly autonomous symmetry generators have polynomial coefficients, so the complete Lie algebra is obtained. These symmetries are found for the first time and allow one to construct more invariant solutions of the considered system.  $\square$

*Example 5.* To calculate approximate symmetries for ODEs with a small parameter  $\varepsilon$ , one needs to add the terms with  $\varepsilon$  into basis operators  $X_i$ . The determining equations are automatically split with respect to  $\varepsilon$  and the number of equations doubles. For example, consider equation from [9]

$$x'' = F_0(x') + \varepsilon (F_1(x') - F_0(x')(3xx' - t) + F_0'(x')x'(xx' - t))$$

constructed from approximate differential invariants. It has four approximate symmetries for arbitrary functions  $F_0(x')$ ,  $F_1(x')$ . For example, let us test the procedure for specific functions  $F_0 = \sin x'$ ,  $F_1 = (x')^2$ :

```

> s.LieSymmSetup(indepsList=['t'], depsList=['x'])
> dx = Symbol('x_t'); d2x = Symbol('x_tt')
> f0 = sin(dx); f01 = cos(dx); f1 = dx**2
> sys = {d2x: f0+ s.e*(f1-f0*(3*x*dx-t)+f01*dx*(x*dx-t))}
> XFS = s.detPDE_Approx(sys)
> s.BaseOps=[]; s.AddPolynomialBasis(3,approx=True)
> s.build_C_Coeffs_Approx(XFS); s.solveNumeric()
> display(s.getAdmittedOperators())

```

$$[\varepsilon xD_x + D_t, \quad \varepsilon xD_t + D_x, \quad \varepsilon D_t, \quad \varepsilon D_x]$$

The found operators are correct. The procedure gives the same result for any other forms of  $F_0$ ,  $F_1$ , even very complicated.

Here `BaseOps` includes operators like  $\varepsilon x^2 t^3 \partial_t$  and `XFS` contains  $\varepsilon^0$  and  $\varepsilon^1$  terms ( $\varepsilon^2$  are treated like zeros in this theory [4]).  $\square$

*Example 6.* Let us check the program on a more complex system composed of some differential invariants (the functions  $F_0, G_0, F_1$ , and  $G_1$  are arbitrary):

$$\begin{cases} \ddot{x} = \frac{\dot{x}^2}{x} F_0(\dot{y}) + \varepsilon \left( \frac{\dot{x}^2(t\dot{y}-y)}{x} F'_0(\dot{y}) + \frac{\dot{x}^2}{x} F_1(\dot{y}, x) - \dot{x}^3 \right), \\ \ddot{y} = \frac{\dot{y}}{x} G_0(\dot{y}) - \varepsilon \left( \frac{t\dot{x}}{x} G_0(\dot{y}) - \frac{\dot{x}(t\dot{y}-y)}{x} G'_0(\dot{y}) + \frac{\dot{y}}{x} G_1(\dot{y}, x) \right), \end{cases} \quad (21)$$

```
> s.LieSymmSetup(indepsList=['t'], depsList=['x','y'])
> F0=sympify('z'); G0=sympify('z**3')
> F0y=F0.subs({'z':'y_t'}); DF0y=F0.diff('z').subs({'z':'y_t'})
> G0y=G0.subs({'z':'y_t'}); DG0y=G0.diff('z').subs({'z':'y_t'})
> dx = Symbol('x_t'); dy = Symbol('y_t')
> F1=sympify('sin(x*y_t)'); G1=sympify('y_t**3-x')
> RS1=dx**2/x*F0y + s.e*(dx*dx*(t*dy-y)/x*DF0y+ dx*dx/x * F1-2*dx)
> RS2=dx/x*G0y - s.e*(t*dx/x * G0y- dx*(t*dy-y)/x*DG0y+dx/x*G1)
> XFS=s.detPDE_Approx({'x_tt':RS1,'y_tt':RS2})
> s.BaseOps=[]; s.AddPolynomialBasis(2, approx=True)
> s.build_C_Coeffs_Approx(XFS); s.solveNumeric()
> display(s.getAdmittedOperators())
```

The following admitted operators are found:

$$\begin{aligned} &[-\varepsilon y D_y + D_t, \quad (t^2\varepsilon + t) D_t + (t\varepsilon y + y) D_y, \\ &(t\varepsilon + 1) D_y, \quad \varepsilon D_t, \quad t\varepsilon D_t + \varepsilon y D_y, \quad \varepsilon x D_x, \quad \varepsilon D_y]. \end{aligned}$$

Combinations of 162 basis operators were considered. The calculations took about 15s on a laptop with Intel Core i7-4500U. □

The obvious limitation of the method is in suggesting the fixed form of operator. It is not possible to include all terms like  $x^\beta \partial_u$  or  $\sin(\omega t) \partial_u$  with unknown  $\beta$  or  $\omega$ . Therefore, if the equations admit such specific form of generators, this approach can be used only to check the symmetries (helping to avoid mistakes in calculations). However, most of nonlinear, approximate, and fractional differential equations have rather simple form of symmetries. For fractional differential equations, the fixed form of symmetry generator is the most common case [11].

The main advantage of the approach is that it works for a very wide class of equations including ones with nonlinear functions of derivatives. The program can easily be modified to compute different kinds of symmetries. It makes the described package a suitable tool for express-analysis. It also works when the group is infinite-dimensional, for example, if the first-order equations are considered, when constructing full symmetry algebra analytically is a very complicated problem.

### 3.2 Computing the Operator of Invariant Differentiation in the Specified Form

The algorithm for constructing the differential invariants and the OID for an approximate Lie algebra of operators is realized in Maple system as the program

“PR-OID: construction of differential invariants and an operator of invariant differentiation for an approximate Lie algebra of operators”, which is registered by Rospatent ([26]).

The program is a set of procedures. In this work, we present it with some modifications.

The procedure `commut(S1,S2,DepVars)` takes two vectors with coordinates of the corresponding infinitesimal generators (for example, the vectors  $[\tau_1(t, x, y), \xi_1(t, x, y), \eta_1(t, x, y)]$  and  $[\tau_2(t, x, y), \xi_2(t, x, y), \eta_2(t, x, y)]$  and a list of dependent variables (for example,  $[x(t), y(t)]$ ). The procedure calculates the commutator of given generators by formula (7) and returns the coordinate vector of the resulting generator.

The procedure `is_algebra(S,DepVars)` takes a set of coordinate vectors of infinitesimal generators and a list of dependent variables. The procedure checks that every commutator of given generators belongs to the same vector space. After calculations, the procedure returns the string “It is a Lie algebra”, if the operators generate an Lie algebra, and “It is not a Lie algebra” otherwise.

The procedure `prolong(S,DepVars,n)` takes the vector of coordinates of the infinitesimal generator, the list of dependent variables and the order of derivatives, to which the generator must be continued. The procedure calculates the coordinates of the prolonged generator using standard prolongation formulas (4) and returns the vector of coordinates for the prolonged generator.

The procedure `acting(S,f,DepVars,n)` is auxiliary. It takes the vector of coordinates of the infinitesimal operator, the tested function, the list of dependent variables and the order of the highest derivative in  $f$ . The procedure returns the result of the action of the prolonged generator on the given function.

The procedure `approx_invariants(S,DepVars,n,eps)` takes a set of coordinate vectors of generators, a list of dependent variables, the required order of differential invariants and indicator of case (exact or approximate). The procedure returns a set of all independent invariants up to the  $n$ -th order.

The procedure `approx_OID(S,DepVars,eps)` takes a set of coordinate vectors for infinitesimal generators, a list of dependent variables and indicator of case (exact or approximate). The procedure returns the multiplier for the OID, the function  $\Phi$  used to construct this OID, as well as the result of the action of given generators on the obtained function  $\Phi$ .

*Example 7.* Let us consider the generators

$$\begin{aligned} X_1 &= (1 + \varepsilon t) \frac{\partial}{\partial t}, \quad X_2 = \varepsilon x \frac{\partial}{\partial x}, \quad X_3 = (1 + \varepsilon t) \frac{\partial}{\partial y}, \\ X_4 &= (t + \varepsilon t^2) \frac{\partial}{\partial t} + (y + \varepsilon ty) \frac{\partial}{\partial y} \end{aligned} \tag{22}$$

and write them in a Maple worksheet:

```
> S:=[[1+epsilon*t,0,0],[0,epsilon*x,0],[0,0,1+epsilon*t],
      [t+epsilon*t^2,0,y+epsilon*t*y]];
```

$$S := [[\varepsilon t + 1, 0, 0], [0, \varepsilon x, 0], [0, 0, \varepsilon t + 1], [\varepsilon t + t, 0, \varepsilon t y + y]]$$

If we check them with procedure

```
> is_algebra(S, [x,y](t));
```

we obtain the following structural constants of a certain Lie algebra:

$$\begin{aligned} ["It is a Lie algebra"], & \{a_{1,2,1} = 0, a_{1,2,2} = 0, a_{1,2,3} = 0, a_{1,2,4} = 0\}, \\ & \{a_{1,3,1} = 0, a_{1,3,2} = 0, a_{1,3,3} = \varepsilon, a_{1,3,4} = 0\}, \\ & \{a_{1,4,1} = 1, a_{1,4,2} = 0, a_{1,4,3} = 0, a_{1,4,4} = \varepsilon\}, \\ & \{a_{2,3,1} = 0, a_{2,3,2} = 0, a_{2,3,3} = 0, a_{2,3,4} = 0\}, \\ & \{a_{2,4,1} = 0, a_{2,4,2} = 0, a_{2,4,3} = 0, a_{2,4,4} = 0\}, \\ & \{a_{3,4,1} = 0, a_{3,4,2} = 0, a_{3,4,3} = 1, a_{3,4,4} = 0\} \end{aligned}$$

Note that each constant  $a_{i,j,k}$  is a sum  $a_{i,j,k,(0)} + \varepsilon a_{i,j,k,(1)}$ , i.e., if we obtain  $a_{1,3,3} = \varepsilon$ , it means that  $a_{1,3,3,(0)} = 0$ ,  $a_{1,3,3,(1)} = 1$ , and if we obtain  $a_{3,4,3} = 1$ , it means  $a_{3,4,3,(0)} = 1$ ,  $a_{3,4,3,(1)} = 0$ . So, the basis of Lie algebra is six-dimensional and consists of the following generators:

$$X_1, X_2, X_3, X_4, \varepsilon X_3, \varepsilon X_4.$$

For obtaining the differential invariants up to the second order, we use

```
> approx_invariants(S, [x,y](t), 2, 1);
```

Its result is four invariants

$$y_t + \varepsilon (t y_t - y), \quad \frac{x_{t,t} x}{x_t^2} + \frac{2\varepsilon x}{x_t}, \quad \frac{y_{t,t} x}{x_t} + \frac{\varepsilon y_{t,t} t x}{x_t}, \quad \varepsilon x.$$

The OID for approximate Lie algebra generated by six operators can be obtained by the expression

```
> approx_OID(S, [x,y](t), 1);
```

which returns the multiplier for OID, the function  $\Phi$  and the result of acting generators on  $\Phi$ :

$$\left[ \frac{x}{x_t} \right], [\ln(x)], [0, \varepsilon, 0, 0].$$

Let us consider the system of ODEs (21), which admits the given generators. Using obtained invariants, we can rewrite this system in the form

$$J_1^{(2)} \approx F_0(J^{(1)}) + \varepsilon F_1(J_{(0)}^{(1)}, J_{(0)}^{(0)}), \quad J_1^{(2)} \approx G_0(J^{(1)}) + \varepsilon G_1(J_{(0)}^{(1)}, J_{(0)}^{(0)}).$$



Applying OID to invariants  $J^{(1)}$  and  $J^{(0)}$ , we get

$$\begin{cases} \frac{x}{\dot{x}}D_t(\dot{y}) = H_0(\dot{y}), \\ \frac{x}{\dot{x}}D_t(t\dot{y} - y) = H_0(\dot{y})\frac{x}{t\dot{x}} + H'_0(\dot{y}) \cdot (t\dot{y} - y) + H_1(\dot{y}, x), \\ \frac{x}{\dot{x}}D_t(x) = x. \end{cases}$$

General solution of these equations is the approximate first integral of considered system. Thus, we obtain the reducing system

$$J^{(1)} \approx W_0(\Phi) + \varepsilon W_1(\Phi_0), \quad J^{(2)} \approx \hat{F}_0(\Phi) + \varepsilon \hat{F}_1(\Phi_0),$$

which admits generators  $X_1, X_3$ , and  $X_4$ . Repeating the procedure, we can obtain the approximate solution of given system.  $\square$

The program can also be used to study the exact Lie algebra and construct its differential invariants and the invariant differentiation operator.

*Example 8.* Let us consider the generators

$$X_1 = x \frac{\partial}{\partial t}, \quad X_2 = x \frac{\partial}{\partial x}, \quad X_3 = y \frac{\partial}{\partial t}, \quad X_4 = y \frac{\partial}{\partial y} \tag{23}$$

and write them in Maple worksheet:

```
> S:=[ [x,0,0], [0,x,0], [y,0,0], [0,0,y] ];
```

If we check them with procedure

```
> is_algebra(S, [x,y] (t));
```

we obtain the following structural constants of a certain Lie algebra:

$$\begin{aligned} & \text{[“It is a Lie algebra”], } \{ \{ a_{1,2,1} = -1, a_{1,2,2} = 0, a_{1,2,3} = 0, a_{1,2,4} = 0 \}, \\ & \quad \{ a_{1,3,1} = 0, a_{1,3,2} = 0, a_{1,3,3} = 0, a_{1,3,4} = 0 \}, \\ & \quad \{ a_{1,4,1} = 0, a_{1,4,2} = 0, a_{1,4,3} = 0, a_{1,4,4} = 0 \}, \\ & \quad \{ a_{2,3,1} = 0, a_{2,3,2} = 0, a_{2,3,3} = 0, a_{2,3,4} = 0 \}, \\ & \quad \{ a_{2,4,1} = 0, a_{2,4,2} = 0, a_{2,4,3} = 0, a_{2,4,4} = 0 \}, \\ & \quad \{ a_{3,4,1} = 0, a_{3,4,2} = 0, a_{3,4,3} = -1, a_{3,4,4} = 0 \} \} \end{aligned}$$

The first- and second-order differential invariants of this Lie algebra are obtained by

```
> approx_invariants(S, [x,y] (t), 2, 0);
```

and they have the form

$$\frac{xy_t}{x_t y}, \quad \frac{(x_t y_{t,t} - x_{t,t} y_t) x^2}{x_t^3 y}, \quad \frac{x_{t,t} (-t y_t + y) + y_{t,t} (t x_t - x)}{x_t y_{t,t} - x_{t,t} y_t}.$$

The OID for Lie algebra generated by given operators can be found by

`>approx_OID(S, [x, y] (t), 0);`

which returns the multiplier for OID, the function  $\Phi$ , and the result of acting generators on  $\Phi$ :

$$\left[ \frac{x}{C_2 x_t} \right], [C_2 \ln(x)], [0, C_2, 0, 0]$$

with arbitrary nonzero constant  $C_2$ . Let  $C_2 = 1$ .

Applying OID to the first-order invariant, we obtain

$$\frac{x}{\dot{x}} D_t \left( \frac{x\dot{y}}{\dot{x}y} \right) = \frac{x\dot{y}}{\dot{x}y} - \left( \frac{x\dot{y}}{\dot{x}y} \right)^2 + \frac{(\dot{x}\ddot{y} - \dot{y}\ddot{x})x^2}{\dot{x}^3 y}.$$

So, for the system of ODEs

$$\begin{cases} \ddot{x} = \frac{x\dot{y} + t\dot{x}^2\dot{y}^2 - x\dot{x}\dot{y}^2}{x(\dot{y}(y - t\dot{y}) + \dot{x}(x - t\dot{x}))}, \\ \ddot{y} = \frac{y\dot{x}\dot{y}^2 - t\dot{x}\dot{y}^3 - x\dot{x}}{x(\dot{y}(y - t\dot{y}) + \dot{x}(x - t\dot{x}))}, \end{cases}$$

which admits these generators, one obtains

$$\frac{dJ^{(1)}}{d\Phi} = 2J^{(1)} - (J^{(1)})^2, \quad J^{(1)} = \frac{x\dot{y}}{\dot{x}y}.$$

And the first integral of given system is

$$\Phi = \frac{1}{2} \ln \left| \frac{J^{(1)}}{J^{(1)} - 2} \right| - \frac{1}{2} \ln C_1.$$

The reduced system is

$$\begin{cases} \dot{x} = \frac{C_1 x^2 \dot{y} - y}{2C_1 xy}, \\ \dot{y} = \frac{y\dot{x}\dot{y}^2 - t\dot{x}\dot{y}^3 - x\dot{x}}{x(\dot{y}(y - t\dot{y}) + \dot{x}(x - t\dot{x}))}. \end{cases}$$

These equations admit 3 generators:  $x \frac{\partial}{\partial t}$ ,  $y \frac{\partial}{\partial t}$ ,  $y \frac{\partial}{\partial y}$ . The order reduction procedure can be repeated. □

## 4 Conclusion

In this work, we have presented two programs for automatic symmetry calculation and constructing OID for ODE system order reduction by Lie group methods.

The procedure of finding symmetries is realized semi-numerically. It allows one to compute symmetries of equations of any complex form. Another feature

of the program is that it can be used for finding the symmetries of ODEs with small parameter and fractional ODEs and their systems. The form of symmetry generator is chosen by the user, it can contain non-polynomial terms. No analytical solving of PDE systems is required. The examples show the correct behavior of the program for FDEs and ODEs with a small parameter for equations with known symmetries. In example 4, new symmetries are found for FDE system which were not published before.

The second program implements the recently developed algorithm of OID construction. It is the first realization of this algorithm. The program can be used for constructing OIDs of Lie algebras of generators, which are admitted by standard ODE systems as well as systems of ODEs with a small parameter. In the second case, the OID is constructed for corresponding approximate Lie algebra of generators.

In the future, we plan to combine described programs on one platform (maybe using SAGE as it has interfaces to other systems) and automate the algorithm for order reduction of the ODE system with and without small parameter.

**Acknowledgments.** We are grateful to Prof. R.K. Gazizov and Prof. S.Yu. Lukashchuk for constructive discussion. Also we thank the referees whose comments helped us a lot to improve the early draft of this paper.

## References

1. Olver, P.J.: Applications of Lie Groups to Differential Equations, 1st edn. Springer-Verlag, New York (1986). <https://doi.org/10.1007/978-1-4684-0274-2>
2. Ovsyannikov, L.V.: Group Analysis of Differential Equations, 1st edn. Academic Press, New York (1982)
3. Ibragimov, N.H.: Elementary Lie Group Analysis and Ordinary Differential Equations. John Wiley and Sons, Chichester (1999)
4. Baikov, V.A., Gazizov, R.K., Ibragimov, N.H.: Approximate groups of transformations. *Differ. Uravn.* **29**(10), 1712–1732 (1993). (in Russian)
5. Ayub, M., Mahomed, F.M., Khan, M., Qureshi, M.N.: Symmetries of second-order systems of ODEs and integrability. *Nonlinear Dyn.* **74**, 969–989 (2013). <https://doi.org/10.1007/s11071-013-1016-3>
6. Wafo Soh, C., Mahomed, F.M.: Reduction of order for systems of ordinary differential equations. *J. Nonlinear Math. Phys.* **11**(1), 13–20 (2004). <https://doi.org/10.2991/jnmp.2004.11.1.3>
7. Gainetdinova, A.A., Gazizov, R.K.: Integrability of systems of two second-order ordinary differential equations admitting four-dimensional Lie algebras. *Proc. Roy. Soc. A: Math. Phys. Eng. Sci.* **473**(2197), 20160461 (2017). <https://doi.org/10.1098/rspa.2016.0461>. 13 pp
8. Gazizov, R.K., Gainetdinova, A.A.: Operator of invariant differentiation and its application for integrating systems of ordinary differential equations. *Ufa Math. J.* **9**(4), 12–21 (2017). <https://doi.org/10.13108/2017-9-4-12>
9. Gainetdinova, A.A.: Integration of systems of ordinary differential equations with a small parameter which admit approximate Lie algebras. *Vestnik Udmurtskogo Universiteta: Matematika, Mekhanika, Komp'yuternye Nauki* **28**(2), 143–160 (2018). <https://doi.org/10.20537/vm180202>. (in Russian)

10. Kilbas, A.A., Srivastava, H.M., Trujillo, J.J.: Theory and Applications of Fractional Differential Equations. Elsevier, The Netherlands (2006)
11. Gazizov, R.K., Kasatkin, A.A., Lukashchuk, S.Yu.: Symmetries and group invariant solutions of fractional ordinary differential equations. In: Kochubei, A., Luchko, Yu. (eds.) Fractional Differential Equations, pp. 65–90. De Gruyter, Berlin (2019). <https://doi.org/10.1515/9783110571660>
12. Gazizov, R.K., Kasatkin, A.A., Lukashchuk, S.Yu.: Symmetries, conservation laws and group invariant solutions of fractional PDEs. In: Kochubei, A., Luchko, Yu. (eds.) Fractional Differential Equations, pp. 353–382. De Gruyter, Berlin (2019). <https://doi.org/10.1515/9783110571660>
13. Galaktionov, V.A., Svirshchevskii, S.R.: Exact Solutions and Invariant Subspaces of Nonlinear Partial Differential Equations in Mechanics and Physics. Chapman and Hall/CRC (2006). <https://doi.org/10.1201/9781420011623>
14. Gazizov, R.K., Kasatkin, A.A.: Construction of exact solutions for fractional order differential equations by the invariant subspace method. *Comput. Math. Appl.* **66**(5), 576–584 (2013). <https://doi.org/10.1016/j.camwa.2013.05.006>
15. Sahadevan, R., Bakkyaraj, T.: Invariant subspace method and exact solutions of certain nonlinear time fractional partial differential equations. *Fractional Calc. Appl. Anal.* **18**(1), 146–162 (2015). <https://doi.org/10.1515/fca-2015-0010>
16. Gazizov, R.K., Kasatkin, A.A., Lukashchuk, S.Yu.: Linearly autonomous symmetries of the ordinary fractional differential equations. In: Proceedings of 2014 International Conference on Fractional Differentiation and Its Applications (ICFDA 2014), pp. 1–6. IEEE (2014). <https://doi.org/10.1109/ICFDA.2014.6967419>
17. Kasatkin, A.A.: Symmetry properties for systems of two ordinary fractional differential equations. *Ufa Math. J.* **4**(1), 71–81 (2012)
18. Hereman, W.: Review of symbolic software for Lie symmetry analysis. *Math. Comput. Model.* **25**(8–9), 115–132 (1997). [https://doi.org/10.1016/S0895-7177\(97\)00063-0](https://doi.org/10.1016/S0895-7177(97)00063-0)
19. Cheviakov, A.F.: Symbolic computation of local symmetries of nonlinear and linear partial and ordinary differential equations. *Math. Comput. Sci.* **4**(2–3), 203–222 (2010). <https://doi.org/10.1007/s11786-010-0051-4>
20. Vu, K.T., Jefferson, G.F., Carminati, J.: Finding higher symmetries of differential equations using the MAPLE package DESOLVII. *Comput. Phys. Commun.* **183**(4), 1044–1054 (2012). <https://doi.org/10.1016/j.cpc.2012.01.005>
21. Jefferson, G.F., Carminati, J.: FracSym: automated symbolic computation of Lie symmetries of fractional differential equations. *Comput. Phys. Commun.* **185**(1), 430–441 (2014). <https://doi.org/10.1016/j.cpc.2013.09.019>
22. Merkt, B., Timmer, J., Kaschek, D.: Higher-order Lie symmetries in identifiability and predictability analysis of dynamic models. *Phys. Rev. E, Stat. Nonlinear Soft Matter Phys.* **92**(1), 012920 (2015). <https://doi.org/10.1103/PhysRevE.92.012920>
23. Golub, G.H., Van Loan, C.F.: Matrix Computations. JHU Press, Baltimore (2012)
24. Bagderina, Y.Y., Gazizov, R.K.: Invariant representation and symmetry reduction for differential equations with a small parameter. *Commun. Nonlinear Sci. Num. Simul.* **9**(1), 3–11 (2004). [https://doi.org/10.1016/S1007-5704\(03\)00010-8](https://doi.org/10.1016/S1007-5704(03)00010-8)
25. Meurer, A., et al.: SymPy: symbolic computing in Python. *PeerJ Comput. Sci.* **3**, e103 (2017). <https://doi.org/10.7717/peerj-cs.103>
26. Gainetdinova, A.A.: Computer program registration certificate 2018618063. Federal Service for Intellectual Property (Rospatent). Registered on 07 September 2018