Algorithms

for

Symmetry Analysis

by

Shih-lung Tracy Huang

A thesis submitted in partial fulfilment of the requirements of the degree of Masters of Information Sciences (Research), University of Canberra.

November 2009

© Shih-lung Tracy Huang, 2009

Abstract

We consider Lie point symmetry analysis of differential equations (DEs). For a family of DE systems containing arbitrary elements, the problem of symmetry classification can be solved algorithmically using a differential reduction & completion (DRC) algorithm applied to the determining equations of the symmetry vector fields. DRC algorithms such as Reid and Wittkopf's RIF split the determining equations into a number of cases. A family of DEs may additionally have some equivalence transformations which map DEs to other DEs within the same family. Case splittings of the symmetry classification should be invariant under the action of this equivalence group.

In this thesis, we give a new procedure for testing case splittings for invariance under the equivalence group action. The procedure uses the Lie infinitesimal technique and works on the level of determining equations. It is based on a method of computing prolongations of vector fields whose infinitesimals satisfy given determining equations. Our procedure does not need to know the equivalence group or the equivalence vector fields. The process is algorithmic and has been implemented as a package in the computer algebra system Maple. This package is to assist the existing DRC package `rifsimp` (which uses RIF algorithm) to improve classifying symmetries. We illustrate use of the package by applying it to symmetry classification of the 1+1 Richards equation and linear hyperbolic equations.

Certificate of Authorship of Thesis

Except where clearly acknowledged in footnotes, quotations and the bibliography, I certify that I am the sole author of the thesis submitted today entitled

Algorithms for Symmetry Analysis

I further certify that to the best of my knowledge the thesis contains no material previously published or written by another person except where due reference is made in the text of the thesis.

The material in the thesis has not been the basis of an award of any other degree or diploma except where due reference is made in the text of the thesis.

The thesis complies with University requirements for a thesis as set out in http://www.canberra.edu.eu/secretariat/goldbook/forms/thesisrqmt.pdf

Signature of Candidate: _____

Signature of chair of
the supervisory panel: _____

Date: _____

Acknowledgements

I am deeply indebted to my supervisor, Dr. Ian Lisle, for his guidance, encouragement and generous (unlimited) support throughout the years. His enthusiasm, his mathematical knowledge & proficiency and his patience guided me to the end of the thesis. More importantly, his fussiness has brought this research to a high standard – I thank him sincerely.

I am also deeply indebted to my co-supervisor, Dr. Peter Vassiliou, for giving excellent math advice & seminars and criticisms in the final stages of my thesis.

Special thanks to Dr. Gregory Reid and Dr. Allan Wittkopf for giving permission to modify the Maple `rifsimp` package, and helpful discussions on the modification of `rifsimp` and Maple implementation tips. Thanks also to Dr. Edgardo Cheb-Terrab for his insights into the Maple subpackage `PDEtools:-Symmetries`.

I also would like to thank many of my teachers over the years, in particular Mary Hewett, Dr. Peter Brown, Prof. Robert Bartnik, John Mathews, Dr. Judith Ascione, Dr. Shuangzhe Liu, Dr. Dat Tran, Robert Cox, A/Prof. John Campbell, and A/Prof. Craig McDonald, who have made me become a more skilled and knowledgeable person in Mathematics, Statistics, Software Engineering, and research.

I wish to acknowledge the travel and conference support provided by AMSI and PIMS.

On a personal note, I would like to thank my family for giving me the opportunity and financial support to continue my study.

Last, I wish to express my deepest appreciation to all staff in the Faculty of Information Sciences and Engineering at University of Canberra, for wonderful discussions and encouragement at Friday technical meetings.

Contents

1 Introduction **5**

2 Symmetry Analysis of DEs **11**

2.1 Symmetry Analysis 13

2.1.1 Lie Point Symmetry Analysis 18

2.2 Symmetry Classification 22

2.3 Equivalence Transformations 25

3 Computer Algebra in Symmetry Analysis **35**

3.1 Symmetry Analysis Packages 36

3.2 Differential Reduction & Completion Algorithms 37

3.2.1 The RIF Algorithm 41

3.3 Symmetry Classification Using RIF 51

3.3.1 Symmetry Classification Using `Rifsimp` 54

3.4 Results from Commutative Algebra 57

4 Invariance Checking from Determining Equations **61**

4.1 Projection of Equivalence Group Action 62

4.2 Issues with Symmetry Condition 67

4.3 Invariance Using Determining Equations 72

4.4 Invariance Checking in Symmetry Classification 81

4.4.1 Label pivots from classification tree 82
4.4.2 Guide RIF during classification 88

5 Implementation of Invariance Checking Method **91**
5.1 Required Implementations 92
5.2 Symmetry Classification Package 95
5.2.1 Storage Structure for DE System 97
5.2.2 Pre-step methods 99
5.2.3 `Rifsimp` with the ICM method 100
5.2.4 Display Procedure 103
5.2.5 Front-end procedure 103
5.3 Examples 104
5.3.1 1+1 Richards Equation 104
5.3.2 Linear Hyperbolic Equation with Laplace Invariants 110

6 Conclusion **115**

References **121**

A The `SymmetryClassification` Package **129**
`SymmetryClassification` Overview 130
`pdeRecord` 134
`newPDESys` 141
`detEqsForSymm` 144
`detEqsForEquiv` 147
`newProlongation` 150
`AddInvtInfo` 156
`SymmetricRifsimp` 159
`CasePlot` 162

`classifySymmetry` 168

B Published work arising from the thesis **173**

Chapter 1

Introduction

Differential equations (DEs) arise in many areas of science and technology such as physics, engineering and geometry. DEs are mathematically studied from several different perspectives, mostly are concerned with their solutions. Only rarely are we able to find solutions of a DE in terms of finite explicit formulas. But constructing such a solution can be a very important step in understanding, especially for difficult non-linear problems. One of the most powerful general approaches for constructing solutions for DEs is to study their Lie group of symmetries [70].

For a system of DEs, if there is a point transformation which maps each solution of DE to another solution then such a transformation is called a *symmetry*. Knowing the symmetries of DEs can help us to do various things such as studying the behaviour of the solution of DEs system, solving ordinary differential equation (ODE) system in formula [61], finding similarity solutions of partial differential equation (PDE) system [5, chap. 4], etc. Furthermore, we can even use their symmetries to extract information from DEs without having to solve them. Symmetry analysis for DEs was started in 1870 by Sophus Lie [69], and Lie symmetry methods have

since become a major tool for analysis of ODEs & PDEs [5, 47, 48]. There are several symmetry methods which are applied to different purposes. In this thesis, we will be dealing only with Lie point symmetries of a DE system. Other symmetry methods (contact [5, chap. 5.2], generalised [47, Chap. 5], Cartan, ...) are similar.

A main idea of the symmetry analysis is to work 'infinitesimally' at the level of 'group operators' or *vector fields*. Throughout this thesis, we will be working entirely on the level of infinitesimals, not on groups.

Due to the fact that methods of symmetry analysis involve systematic algebraic manipulation and tedious calculation, the method is well-suited to computer algebra. A number of computer algebra packages for symmetry analysis have been developed [13, 23, 63, 68]; and they have been used as an important basis for analysing and solving DEs.

Computer algebra implementation for symmetry analysis started in early 1980s, with early symmetry packages being specialised in symmetry analysis only, their goal being to find the symmetries of DEs. Then from about 1990, the development of symmetry analysis packages has separated the process into three parts: (1) finding determining equations of DEs, (2) reducing these determining equations, and (3) solving these reduced determining equations (so we get the symmetries). The purpose of Step 2 (reducing determining equations) is to simplify these determining equations so the chance of solving them is greatly increased. This is where *differential reduction and completion* (DRC) methods [7, 40, 54] are included in symmetry analysis. A DRC method is to help to reduce DEs into a form which contains some information about the solution of DEs (we call such a form 'reduced form'); also methods might have a better chance to solve DEs. (In symmetry analysis the DEs that the DRC method is ap-

plied to are the determining equations.) Several DRC methods including the RIF algorithm (reduced involutive form) [54], differential Gröbner basis [40] and Rosenfeld–Gröbner algorithm [7] have been implemented in computer algebra systems such as Maple. These computer algebra packages have been widely used by users. For example, the Maple package `rifsimp`, which uses RIF algorithm, has become the front-end procedure before solving DEs [66, help on `pdsolve,system`].

In symmetry analysis, a DEs system can also be a 'family' of DEs which involves arbitrary elements. These arbitrary elements can be constants or functions, often representing physical properties such as wave speed, diffusivity, etc. One is interested to see the behaviour for symmetries of DEs with different conditions on arbitrary elements applied. In this case, the problem of finding the symmetries of DEs system is called the *symmetry classification*. Many DE systems have been classified. People have been dealing with symmetry classification problem in several different ways [48, 4, 18] but there is no clear best (systematic) way to classify symmetries in all cases.

The symmetry classification problem can also been done by using DRC methods such as RIF [54, 67]. We choose RIF to classify symmetries in this thesis because it is robust and efficient, and this DRC method is implemented in Maple. To ask RIF to classify symmetries, one needs to make sure that all arbitrary elements are ranked lower than other variables [54], so the classification is forced to split on DEs that only involve arbitrary elements (these are the conditions on arbitrary elements mentioned above). In a complete classification produced by RIF, the reduced form of the determining equations in each case provides certain geometric information about the symmetry group action, and this information can help us to

analyse the symmetry group even before solving the determining equations [53, 54].

As well as symmetries, a family of DE systems usually has *equivalence transformations*. These are transformations which map DEs to DEs in the same family: only the arbitrary elements change 'value'. The equivalence transformations form a group called the *equivalence group* [48]. In the symmetry classification problem, the equivalence group can help to remove parameters after classification. Therefore, it is desirable to keep equivalent DEs in the same branch of classification tree. However, the DRC method RIF, which we use to classify symmetries in this thesis, does not do symmetry classification in a very intelligent way, for example, the classification tree does not respect the equivalence group, and the case splitting conditions are not guaranteed to be invariant under action of the equivalence group.

In this thesis, our objective is to help improve the way that the DRC algorithm RIF can be used to classify symmetries such that the classification respects the equivalence group. Our approach is to come up with a way to assist RIF to select a case splitting DE which is 'invariant' under the equivalence group action. We also challenge ourselves to make sure that the entire classification process is working on the level of determining equations – of symmetries and of equivalence. This means there is no need to know the explicit groups such as the equivalence group or symmetry group. Moreover, there is no need even to know the explicit vector fields for symmetry and equivalence. Because of this, the whole process is purely algorithmic, and we push ourselves a bit further to actually implement the method in computer algebra as a package for symmetry classification.

As a result in this thesis, we develop an invariance checking method for testing invariance of given DEs under the action of some group. Then we further apply this method to test invariance of case splitting DEs under the equivalence group action in the classification using RIF. In order to check invariance under the action of the equivalence group, we need the equivalence group. But in our case, we use the determining equations of the equivalence group instead of using the group itself. To do a better job, we only need the determining equations for the equivalence which is projected down to the space of arbitrary elements.

In order to show that we did achieve our objective, the whole process of classifying symmetries using RIF and the method is implemented in the computer algebra system Maple as a package. The idea of this implementation is to do the whole symmetry classification using the Maple function `rifsimp` (which uses RIF) from a DE system given by a user. The package includes some major implementations such as deriving the determining equations for the equivalence group in the space of arbitrary elements, providing the invariance checking method, and modifying `rifsimp` so it can select invariant case splitting DEs during classification.

The rest of thesis is organised as follows. We first provide some needed mathematical background and reviews in §2 and §3. In §2, we state some mathematical background in the areas of symmetry analysis (in particular in Lie point symmetry analysis) and symmetry classification for DEs. The steps of how to derive the determining equations for the equivalence group are in §2.3. Then in §3, we give a general review of computer algebra packages related to symmetry analysis and the DRC methods. In this chapter we show how RIF works and we illustrate how to use RIF in symmetry classification.

In Chapter 4 we present the main results of the thesis. We describe the development of the invariance checking method (i.e. how the method only needs the determining equations), and how we apply the method in symmetry classification. Some side discussion includes the method of projecting the equivalence group to the space of arbitrary elements, and why and how we develop an algebraic version of the symmetry condition (§4.2). We provide a complete example of using RIF with the invariance checking method to perform symmetry classification.

Chapter 5 describes the implementation of the method. This chapter includes a list of tasks required which are based on the previous chapter. We describe the design decisions of the package, and give a list of descriptions on each function in the package. We also demonstrate the use of the package by working some examples. The Maple help pages of the package are included in Appendix A.

Chapter 2

Symmetry Analysis of Differential Equations

Suppose we are considering a system E of DEs $\{f^1 = 0, f^2 = 0, \ldots, f^s = 0\}$ where $f^\nu(x, u, u_{(k)})$ involves n independent variables $x = (x^1, x^2, \ldots, x^n)$, m dependent variables $u = (u^1, u^2, \ldots, u^m)$, and $u_{(k)} = (u_1, u_2, \ldots, u_k)$ are the derivatives of u up to order k. Let $X = \mathbb{R}^n$, with coordinates x, be the space denoting the independent variables, and let $U = \mathbb{R}^m$, with coordinates u, denote the dependent variables. The space $X \times U$ is called 0-th order *jet space*, and is denoted by $J^0(X, U)$. The system E lives in the k-th order jet space $J^k(X, U)$ of the underlying space $J^0(X, U)$ [47, §2].

A multi-index $I = [i_1, i_2, \ldots, i_q]$ with $1 \leq i_q \leq n$, $1 \leq q \leq k$ is introduced to index partial derivatives. This multi-index I does not depend on the order of i's, and is used to define jet coordinates:

$$u^j_I = u^j_{i_1 i_2 \ldots i_q} \tag{2.1}$$

which can be thought of as values of the partial derivatives $\dfrac{\partial^q u^j}{\partial x^{i_1} \cdots \partial x^{i_q}}$.

We call q the *order* of I, denoted by $|I|$.

For example with $I = [1, 1, 2]$ then $|I| = 3$, and

$$u^1_{112} = u^1_{121} = u^1_{211} \left(= \text{“} \frac{\partial^3 u^1}{(\partial x^1)^2 \partial x^2} \text{”} \right)$$

The notation I, i from (2.1) denotes further differentiation:

$$u^j_{I,i} = \frac{\partial u^j_I}{\partial x^i} = \frac{\partial^{q+1} u^j}{\partial x^i \partial x^{i_1} \cdots \partial x^{i_q}} \tag{2.2}$$

In this thesis, our aim is algebraic manipulation of DEs, and the coordinates $x, u, u_{(k)}$ are treated as formal symbols which obey certain algebraic rules. However, we continue to use notations like $X \times U$ where there is no chance of confusion.

Definition 2.0.1 (Symmetry group)**.** [47, §2.2] Let E be a system of DEs. A *symmetry group* of the system E is a group of transformations G on the space $X \times U$ of independent and dependent variables for the system in such a way that G transforms every solution of E to another solution of E.

Some symmetries such as translation, scaling and rotation are found commonly in DEs. For example, we would expect the wave equation to have translation symmetries because the solution of the wave equation does not depend on where the wave is started. Also in fluid flow, the units of measurement should not change the behaviour of the solutions, it shows that scaling symmetries are involved [9]. Therefore, any symmetry can help to study the behaviour of solution of DEs.

Symmetries are widely used in

- solving ODE in formula [5, chap. 3]

- reducing PDE (e.g. to find similarity solutions) [5, chap. 4]
- mapping solutions to solutions (e.g. to solve certain boundary value problems numerically [37, 62])
- solving mapping problems between different equations [5, chap. 4]

2.1 Symmetry Analysis

The method for finding symmetries of DEs was introduced in about 1870 by Sophus Lie (1842–1899), who pioneered the study of continuous transformation groups that leave systems of DEs invariant [69]. Ever since, the Lie symmetry method has become one of the major ways to analyse ordinary and partial differential equations (ODEs/PDEs) [5, 47, 48]. The success of Lie's method is partly because it is able to find a symmetry of given DEs systematically.

Lie's method works 'infinitesimally', by considering tangent vector fields $\mathbb{X}$ – that is, it works with vector fields rather than transformation groups. Such an $\mathbb{X}$ assigns a tangent vector $\mathbb{X}|_x$ at each point $x \in \mathbb{R}^n$ with $\mathbb{X}|_x$ varying smoothly from point to point. In local coordinates $(x^1, \ldots, x^n)$, a vector field has the form

$$\mathbb{X}|_x = \xi^1(x)\frac{\partial}{\partial x^1} + \xi^2(x)\frac{\partial}{\partial x^2} + \cdots + \xi^n(x)\frac{\partial}{\partial x^n},$$

where each $\xi^i(x)$ is a smooth function of x. The coefficients $\xi^i(x)$ of the vector field $\mathbb{X}$ are called "infinitesimals" [28]. Let $\psi(t; x)$ be a one-parameter transformation group on $\mathbb{R}^n$, then its infinitesimal generator is obtained by

differentiating at $t = 0$:

$$\mathbb{X}|_x = \left.\frac{d}{dt}\right|_{t=0} \psi(t; x)$$

Moreover, $x' = \psi(t; x)$ is a solution of ODE initial value problem

$$\frac{dx^{i\prime}}{dt} = \xi^i(x'), \qquad x^{i\prime}(0) = x^i, \qquad i = 1, \ldots, n \tag{2.3}$$

Strictly speaking this is only a *local* Lie transformation group. That is, the group transformations are only defined in a neighbourhood of the identity and act on a open set of $X \times U$ (independent and dependent variables of the DE system) [47, §1.2]. In this thesis, we will be entirely working on the infinitesimals, so we don't need to find the groups anyway.

For DEs we start with a vector field $\mathbb{X} = \sum_{i=1}^{n} \xi^i \frac{\partial}{\partial x^i} + \sum_{j=1}^{m} \eta^j \frac{\partial}{\partial u^j}$ on $J^0(X, U)$. However we need to 'prolong' this vector field $\mathbb{X}$ to $J^k(X, U)$, that is to an action on the space of independent variables, dependent variables and derivatives up to order k.

Definition 2.1.1 (Total Derivative)**.** Let $P(x, u, u_{(k)})$ be a smooth function on $J^k(X, U)$. The i-th "total derivative" of P has the general form

$$D_i P = \frac{\partial P}{\partial x^i} + \sum_{j=1}^{m} \sum_{|I|=0}^{k} u^j_{I,i} \frac{\partial P}{\partial u^j_I}, \tag{2.4}$$

where $I = (i_1, i_2, \ldots, i_q)$.

Theorem 2.1.2 (Prolongation Formula)**.** *[47, §2.3] Suppose $\mathbb{X}$ is a vector field on $X \times U$ of the form*

$$\mathbb{X} = \sum_{i=1}^{n} \xi^i \frac{\partial}{\partial x^i} + \sum_{j=1}^{m} \eta^j \frac{\partial}{\partial u^j}$$

where ξ^i, η^j *depend on* (x, u)*. The k-th* prolongation *of* $\mathbb{X}$ *(denoted by* $\mathbb{X}^{(k)}$*) will be a vector field on the k-th order jet space* $J^k(X, U)$

$$\mathbb{X}^{(k)} = \mathbb{X} + \sum_{j=1}^{m} \sum_{|I|=1}^{k} \eta^j_{(I)} \frac{\partial}{\partial u^j_I} \tag{2.5}$$

where $\eta^j_{(I)}$ *depends on* $(x, u, u_{(q)})$ *with* $q = |I|$*. The coefficient function* $\eta^j_{(I)}$ *of* $\mathbb{X}^{(k)}$ *can be found by the following recurrence:*

$$\eta^j_{(I,i)} = D_i \eta^j_{(I)} - \sum_{l=1}^{n} \left(D_i \xi^l \right) u^j_{I,l}, \tag{2.6}$$

and in particular

$$\eta^j_{(i)} = D_i \eta^j - \sum_{l=1}^{n} \left(D_i \xi^l \right) u^j_l$$

The following example illustrates the use of the prolongation formula.

Example 1. Consider a vector field $\mathbb{X}$

$$\mathbb{X} = \xi \frac{\partial}{\partial x} + \eta \frac{\partial}{\partial y}.$$

on $J^0(\mathbb{R}, \mathbb{R})$ where the independent variable is x and dependent variable is y, and where ξ and η depend on (x, y). Second order jet space $J^2(\mathbb{R}, \mathbb{R})$ has coordinates (x, y, y_x, y_{xx}). From equation (2.5), the prolonged vector field up to order 2 has the form

$$\mathbb{X}^{(2)} = \mathbb{X} + \eta_{(x)} \frac{\partial}{\partial y_x} + \eta_{(xx)} \frac{\partial}{\partial y_{xx}}. \tag{2.7}$$

where $\eta_{(x)}$ depends on (x, y, y_x) and $\eta_{(xx)}$ on (x, y, y_x, y_{xx}). The coefficients

$\eta_{(x)}$ and $\eta_{(xx)}$ can be found from recurrence relation (2.6):

$$\begin{aligned}
\eta_{(x)} =& D_x\eta - y_x D_x\xi \\
=& \left(\eta_x + y_x\eta_y\right) - y_x\left(\xi_x + y_x\xi_y\right) \\
=& \eta_x + y_x\eta_y - y_x\xi_x - y_x^2\xi_y \\
\eta_{(xx)} =& D_x\eta_{(x)} - y_{xx}D_x\xi \\
=& \left(\frac{\partial}{\partial x} + y_x\frac{\partial}{\partial y} + y_{xx}\frac{\partial}{\partial y_x}\right)\left(\eta_x + y_x\eta_y - y_x\xi_x - y_x^2\xi_y\right) \\
& - y_{xx}\left(\xi_x + y_x\xi_y\right) \\
=& \eta_{xx} + 2y_x\eta_{xy} + y_x^2\eta_{yy} - y_x\xi_{xx} - 2y_x^2\xi_{xy} - y_x^3\xi_{yy} \\
& + y_{xx}\left(\eta_y - 2\xi_x - 3y_x\xi_y\right)
\end{aligned} \tag{2.8}$$

To describe the main theorem on the construction of symmetries of DEs (Theorem 2.1.4 below), we need to state a maximal rank condition for the system of differential equations.

Definition 2.1.3 (Maximal Rank Condition)**.** Let $E = \left\{f^1, \ldots, f^s\right\}$ be a system of differential equations

$$f^\nu(x, u, u_{(k)}) = 0, \qquad \nu = 1, \ldots, s \tag{2.9}$$

The system is said to be of *maximal rank* if the Jacobian matrix

$$\mathrm{Jac}_E(x, u, u_{(k)}) = \left(\frac{\partial f^\nu}{\partial u_I^j}\right)$$

of the system E with respect to all variables $(u, u_{(k)})$ is of rank s on the subset of $J^k(X, U)$ defined by (2.9).

Theorem 2.1.4 below connects symmetry groups of a DE system with

the infinitesimal criterion of invariance under the prolonged infinitesimal generator of the group.

Theorem 2.1.4 (Symmetry Sufficient Condition)**.** *[47, §2.3] Suppose*

$$f^\nu(x, u, u_{(k)}) = 0, \quad \nu = 1, \dots s$$

is a system of differential equations of maximal rank defined over $X \times U$. If G is a transformation group acting on $X \times U$, and

$$\mathbb{X}^{(k)}(f^\nu(x, u, u_{(k)})) = 0, \qquad \nu = 1, \dots, s \tag{2.10}$$

whenever

$$f^\nu(x, u, u_{(k)}) = 0, \tag{2.11}$$

for every infinitesimal generator $\mathbb{X}$ of G, then G is a symmetry group of the system.

Letting $E = \{f^1, \dots, f^s\}$, we will write more briefly

$$\mathbb{X}^{(k)} E = 0, \qquad \text{whenever } E = 0$$

Note that if the DE system is written in a solved form with respect to some subset of derivatives then the system automatically satisfies the maximal rank condition. However, this is not always enough for our purposes, and we will return to this issue in §4.2.

There are several concepts of symmetry which serve different purposes: classical Lie point symmetries, contact symmetries, and generalised ('Lie–Bäcklund') symmetries. In this study, we consider Lie point symmetries only; the techniques for other symmetry methods are similar.

2.1.1 Lie Point Symmetry Analysis

Consider a system of differential equations

$$f^{\nu}(x, u, u_{(k)}) = 0, \quad \nu = 1, \dots s \tag{2.12}$$

A point transformation

$$\tilde{x} = F(x, u), \qquad \tilde{u} = G(x, u)$$

is a symmetry of the DE system (2.12) if it maps each solution of (2.12) to another solution (see Definition 2.0.1). The set of symmetries forms a transformation group acting on $X \times U$, called the symmetry group of (2.12). Let the symmetry group have associated infinitesimal generator

$$\mathbb{X} = \sum_{i=1}^{n} \xi^i \frac{\partial}{\partial x^i} + \sum_{j=1}^{m} \eta^j \frac{\partial}{\partial u^j} \tag{2.13}$$

where ξ^i and η^j depend on x, u.

Our job is to determine the coefficients ξ^i and η^j (i.e. the infinitesimals) such that (2.13) is a symmetry vector field of (2.12). To satisfy the symmetry condition as stated in Theorem 2.1.4, the steps are as follows:

Step 1. Prolong the vector field $\mathbb{X}$ (2.13) up to order k. The prolonged vector field $\mathbb{X}^{(k)}$ has the form given in equation (2.5), and the coefficients are found by recurrence (2.6).

Step 2. Apply vector field $\mathbb{X}^{(k)}$ defined in (2.5) to the DEs (2.10) obtaining

$$\mathbb{X}^{(k)} E$$

Step 3. Restrict $\mathbb{X}^{(k)}E$ to the subset where $E = 0$ in $J^k(X, U)$ as stated in condition (2.11)

$$\mathbb{X}^{(k)}E = 0 \qquad \text{whenever } E = 0$$

Step 4. Set the resulting expression for $\mathbb{X}^{(k)}E$ to 0 and split equations by powers of derivatives $u_{(k)}$. As a result we have a list of linear homogeneous PDEs for ξ^i and η^j: we call them the *determining* or *defining* equations for the infinitesimal symmetries of the DE system [5, 9, 27].

Explicit forms for ξ^i and η^j can be found by solving the determining equations, either by hand or by using a symbolic computer algebra system such as Maple. Finally, we substitute the infinitesimals into the vector field $\mathbb{X}$ (2.13) to get the symmetry vector field of the DE system.

The following example demonstrates the Lie point symmetry method.

Example 2 (Second order ODE). Consider the simplest second order ODE

$$y_{xx} = 0 \tag{2.14}$$

where y is the dependent variable, and x the independent variable. Let the associated vector field $\mathbb{X}$ be

$$\mathbb{X} = \xi \frac{\partial}{\partial x} + \eta \frac{\partial}{\partial y} \tag{2.15}$$

where ξ and η depend on (x, y).

To meet the symmetry condition as described in Theorem 2.1.4, we

need to achieve

$$\mathbb{X}^{(2)}(y_{xx}) = 0, \qquad \text{whenever } y_{xx} = 0. \tag{2.16}$$

We now can apply the steps described above:

Step 1. Prolong the vector field (2.15) up to order 2:

$$\mathbb{X}^{(2)} = \xi\frac{\partial}{\partial x} + \eta\frac{\partial}{\partial y} + \eta_{(x)}\frac{\partial}{\partial y_x} + \eta_{(xx)}\frac{\partial}{\partial y_{xx}}$$

where $\eta_{(x)}$, $\eta_{(xx)}$ are given by (2.8)

Step 2. Apply the vector field $\mathbb{X}^{(2)}$ to the DE (2.14)

$$\mathbb{X}^{(2)} y_{xx} = \eta_{(xx)}$$

Step 3. Using the result of $\eta_{(xx)}$ from (2.8), restrict it to the surface $E = 0$ by substituting $y_{xx} = 0$ into $\eta_{(xx)}$:

$$\eta_{xx} + 2y_x\eta_{xy} + y_x^2\eta_{yy} - y_x\xi_{xx} - 2y_x^2\xi_{xy} - y_x^3\xi_{yy}$$

Step 4. Set this to 0 and split equations by powers of y_x. Now the symmetry condition for the DE system (2.16) is given by the list of determining equations

$$\xi_{xx} = 0, \qquad 2\eta_{xy} - \xi_{xx} = 0, \qquad \eta_{yy} - 2\xi_{xy} = 0, \qquad -\xi_{yy} = 0 \tag{2.17}$$

The determining equations for symmetries in this case are simple, so we

can easily solve for ξ and η by hand:

$$\xi(x,y) = a_1xy + a_2x^2 + a_3x + a_4y + a_0$$
$$\eta(x,y) = a_1y^2 + a_2xy + b_1x + b_2y + b_0$$

where $a_0, a_1, a_2, a_3, a_4, b_0, b_1, b_2$ are constants. Finally, the most general symmetry vector field of $y_{xx} = 0$ is

$$\mathbb{X} = (a_1xy + a_2x^2 + a_3x + a_4y + a_0)\frac{\partial}{\partial x} + (a_2xy + a_1y^2 + b_1x + b_2y + b_0)\frac{\partial}{\partial y}$$

Collecting vector fields with respect to the constants, defines a basis for the vector space of infinitesimal generator of the transformation group:

$$xy\frac{\partial}{\partial x} + y^2\frac{\partial}{\partial y}, \qquad x\frac{\partial}{\partial x}, \qquad y\frac{\partial}{\partial x}, \qquad \frac{\partial}{\partial x},$$
$$x^2\frac{\partial}{\partial x} + xy\frac{\partial}{\partial y}, \qquad x\frac{\partial}{\partial y}, \qquad y\frac{\partial}{\partial y}, \qquad \frac{\partial}{\partial y}$$

For these basis vector fields, we can find the corresponding symmetry transformations by solving the ODE initial value problem (2.3). For example,

$$\frac{\partial}{\partial x} \to \begin{cases} x' = x + c \\ y' = y \end{cases} \qquad \text{(Translation symmetry of } x\text{)}$$

$$x\frac{\partial}{\partial x} \to \begin{cases} x' = e^t x \\ y' = y \end{cases} \qquad \text{(Scaling symmetry of } x\text{)}$$

$$x^2\frac{\partial}{\partial x}+xy\frac{\partial}{\partial y}\rightarrow\begin{cases}x'=\dfrac{x}{1-sx}\\[2ex] y'=\dfrac{y}{1-sx}\end{cases}$$

where c, t, s are constants.

Note that the last one-parameter group here is an example of a local Lie group (see p.14) – the transformations are not defined for all s or for all x.

In general, the difficulty of finding and solving symmetry determining equations of DEs increases rapidly as the numbers of independent and dependent variables, and the order of DEs, increase. A differential reduction method (see §3.2) can help increase the chance of solving determining equations of DEs.

2.2 Symmetry Classification

So far we have shown how to construct point symmetries of a DEs system that only involves independent and dependent variables. However symmetry analysis can also apply to a *family* of DEs, where the DEs system contains "arbitrary elements". These arbitrary elements can be either functions or constants in the DEs system and usually represent physical properties such diffusivity, wave speed, etc. The choice of arbitrary element conditions determines a specific DE system within the family, and one can investigate their symmetries. The problem of finding the symmetries of all the DE systems in a family is called *'symmetry classification'* [48, §6].

Example 3. Consider the $1+1$ nonlinear heat equation [28, §10.2],[48, §6.7]

$$\begin{aligned} u_t + q_x &= 0, \\ q &= -K(u)u_x, \qquad \text{where } K \neq 0 \end{aligned} \tag{2.18}$$

Here the arbitrary element is the diffusivity function $K(u)$. The symmetry properties of (2.18) vary depending what form $K(u)$ has. For instance, if $K(u) = 1$ then we have $u_t - u_{xx} = 0$ (the linear heat equation), which has infinitely many symmetries. But if $K(u)$ is non-constant, the DE has only finitely many symmetries.

Examples of symmetry classification for DEs have been collected up to 1994 in the *CRC Handbook of Lie Group Analysis of DEs* [28]. The book includes examples from areas such as heat flow, fluid dynamics, wave propagation, diffusion, etc.: the 'Body of Results' in Part B of this book has some 160 pages. Subsequently many more DE systems have been classified [3, 4, 14, 21, 35, 38]. A survey of symmetry group classification of ODEs is given by Mahomed [39].

Example 4. Referring back to Example 3, take the corresponding vector field $\xi\frac{\partial}{\partial x} + \tau\frac{\partial}{\partial t} + \eta\frac{\partial}{\partial u} + \chi\frac{\partial}{\partial q}$ where ξ, τ, η, χ depend on (x, t, u, q). Following the steps as described in §2.1.1, we obtain the determining equations for point symmetries of DE (2.18),

$$\begin{gathered} \tau_q = 0, \qquad \tau_u + \xi_q = 0, \qquad -\xi_x + \tau_t - \eta_u + \chi_q = 0, \\ q\xi_q + K\eta_q = 0, \qquad q\tau_u - K\tau_x = 0, \qquad K\eta_t - q\chi_u + q\xi_t + K\chi_x = 0, \\ K\eta_x - \frac{q^2}{K}\xi_u - q\eta_u + q\xi_x + \chi - q\frac{K_u}{K}\eta = 0 \end{gathered} \tag{2.19}$$

The determining equations (2.19) contain the arbitrary element K (dif-

fusion coefficient from DE system (2.18)). As a result we expect the 'unclassified' symmetry determining equations (2.19) will split into cases (depending on K) as they are manipulated and solved.

The classification results are as follows:

- For arbitrary $K(u)$ there are three symmetries:

$$\frac{\partial}{\partial x}, \qquad \frac{\partial}{\partial t}, \qquad x\frac{\partial}{\partial x} + 2t\frac{\partial}{\partial t} - \frac{\partial}{\partial q}$$

In the following cases, additional symmetries are obtained:

- $K(u) = e^u$,

$$x\frac{\partial}{\partial x} + 2\frac{\partial}{\partial u} + q\frac{\partial}{\partial q}$$

- $K(u) = u^a$ with $a \neq 0, -\frac{4}{3}$,

$$\frac{a}{2}x\frac{\partial}{\partial x} + u\frac{\partial}{\partial u} + \left(1 + \frac{a}{2}\right)q\frac{\partial}{\partial q}$$

- $K(u) = u^{-4/3}$,

$$-\frac{2}{3}x\frac{\partial}{\partial x} + u\frac{\partial}{\partial u} + \frac{1}{3}q\frac{\partial}{\partial q}, \qquad -x^2\frac{\partial}{\partial x} + 3xu\frac{\partial}{\partial u} + \left(xq - 3u^{-1/3}\right)\frac{\partial}{\partial q}$$

There is no unique way for classifying symmetries, and several methods for symmetry classification have been developed. The first method started with Lie in 1881 [36]. Then Ovsyannikov [48] developed a modern formulation of the method, the so called *"Lie-Ovsyannikov method"* [35, 48]. Using this method, Ovsyannikov was able to complete a number of nontrivial symmetry classifications. However, for a DE system containing arbitrary functions of several variables, the Lie-Ovsyannikov method can

lead to explosive computational difficulties. That is why most symmetry classifications done by the Lie-Ovsyannikov method involve arbitrary functions of one variable only [48].

Another line of investigation (e.g. [4, 21]) follows a method of Gagnon and Winternitz [18], while the Cartan method of equivalence [20] gives a completely geometric approach to certain problems. A number of papers slightly modify the method, for instance by involving the equivalence group [8, 25].

Perhaps there is no single 'best' method for symmetry classification: the candidates suffer variously from not being geometric, or not algorithmic, or being algorithmic but overwhelmed by expression swell on difficult problems, or of producing only partial results. Many papers use a custom-built approach for the particular problem under consideration.

2.3 Equivalence Transformations

We now need to be more precise what we mean by a 'family of DEs'.

Definition 2.3.1 (Family of DEs)**.** Consider a system of DEs with arbitrary elements $A = (a^1, a^2, \ldots, a^r)$,

$$E = \left\{ f^1, \ldots, f^s \right\}, \quad \text{where } f^\nu(x, u, u_{(k)}, a) = 0, \quad \nu = 1, \ldots, s \tag{2.20}$$

where the arbitrary elements are $a = a(x, u)$. The arbitrary elements may be required to satisfy a constraint system,

$$C = \left\{ g^1, \ldots, g^t \right\}, \quad \text{where } g^\mu(x, u, a, a_{(h)}) = 0, \quad \mu = 1, \ldots, t \tag{2.21}$$

Such a system pair E and C is called a family of DEs.

Note carefully that systems (2.20) and (2.21) are of very different kinds. In equation (2.20), the independent variables are x and dependent are u. But in equation (2.21), the independent variables are x, u and dependent are a. In particular, u is a *dependent* variable in DEs system (2.20) and an *independent* variable in constraint system (2.21).

An example of a family of DEs has been given in Example 3.

In Definition 2.3.1 we have made two assumptions:

(i) Assume that the arbitrary elements a depend on (x, u) only.

(ii) Assume that the DEs depend on a only – not on derivatives of a.

Both these assumptions are easily relaxed, but this will be enough for many purposes. Note that assumption (ii) can be easily worked around by appending a constraint system.

Example 5. The nonlinear heat equation is $u_t = \left(K(u)u_x\right)_x$ or

$$u_t = K(u)u_{xx} + \frac{dK}{du}u_x^2$$

which apparently violates (ii). But it can be easily rewritten as

$$u_t = K(u)u_{xx} + L(u)u_x^2$$

with constraint system

$$\frac{dK}{du} = L(u)$$

Definition 2.3.2 (Equivalence Transformation)**.** For a given family of DEs (Definition 2.3.1), an equivalence transformation is a point transformation on $X \times U \times A$ (space of independent and dependent variables and arbitrary elements) for which

- Its prolongation (with respect to $u(x)$) leaves invariant the family of equations (2.20)
- Its prolongation (with respect to $a(x,u)$) leaves invariant the constraint equations (2.21)
- It projects to a point transformation on $X \times U$

This definition agrees with Ovsyannikov [48, §6.4] but we define it more precisely.

Under the equivalence transformation, the form of DEs does not change but the 'value' of the arbitrary elements may be transformed. Solutions of the DEs can be mapped (i.e. one-to-one correspondence) through the equivalence transformation.

Example 6. For the nonlinear heat equation (as shown in Example 3), we have

$$\begin{aligned} &X\ldots \text{space of independent variables, coordinates } (x,t)\\ &U\ldots \text{space of dependent variables, coordinates } (u,q)\\ &A\ldots \text{space of arbitrary elements, coordinates } (K) \end{aligned}$$

The following equivalence transformations are found:

$$x' = ax + e, \quad t' = bt + f, \quad u' = cu + g, \quad q' = \frac{ac}{b}q, \quad K' = \frac{a^2}{b}K \tag{2.22}$$

where a, b, c, e, f, g are arbitrary constants, and $abc \neq 0$. This tells us that the solution can be mapped by re-scaling or translating the variables t, x, u. Under transformations (2.22), the nonlinear heat equation (2.18) is mapped to an equation with the same form, but with the arbitrary element

$K'(u')$ where

$$K(u) = \frac{b}{a^2} K'(cu + g).$$

The set of all equivalence transformations, for a given class of DEs, forms a group known as the equivalence group [48, p.65]. Many equivalence groups of DEs have been collected in the CRC Handbook [28] and more have been found afterward [31, 19, 32]. The equivalence group can be generalised in various ways (see e.g. Meleshko [45]) but the above is enough for our purpose.

The idea for finding the equivalence transformations of DEs is as follows [28, 45].

We search for a one-parameter transformation group living in the space $X \times U \times A$. The vector field has the form:

$$\mathbb{Y} = \sum_{i=1}^{n} \xi^i \frac{\partial}{\partial x^i} + \sum_{j=1}^{m} \eta^j \frac{\partial}{\partial u^j} + \sum_{l=1}^{r} \alpha^l \frac{\partial}{\partial a^l} \tag{2.23}$$

where ξ^i, η^j depend on (x, u), and α^l depends on (x, u, a).

In order to find the equivalence transformation, both systems E (main system (2.20)) and C (constraint system (2.21)) must be invariant. Due to these systems depending on different variables, there are two different processes of prolongation:

- For the main system E, we prolong $\mathbb{Y}$ to order k where independent variables are x and dependent variables are u. The prolonged vector field is denoted by $\mathbb{Y}^{(k,0)}$:

$$\mathbb{Y}^{(k,0)} = \mathbb{Y} + \sum_{j=1}^{n} \sum_{|I|=1}^{k} \eta^j_{(I)} \frac{\partial}{\partial u^j_I}$$

- For the constraint system C, we prolong the same vector field $\mathbb{Y}$ to order h instead, with independent variables (x, u) and dependent variables a. The prolonged vector field is denoted by $\mathbb{Y}^{(0,h)}$:

$$\mathbb{Y}^{(0,h)} = \mathbb{Y} + \sum_{l=1}^{r} \sum_{|L|=1}^{h} \alpha^l_{(L)} \frac{\partial}{\partial a^l_L}$$

Using the infinitesimal techniques as stated in §2.1.1, we follow the steps to find the equivalence group:

Step 1. Find the determining equations for the constraint system C. This can be derived by using the steps from §2.1.1.

Step 2. In the main system E, apply the prolonged vector field $\mathbb{Y}^{(k,0)}$ to the DEs (2.20)

$$\mathbb{Y}^{(k,0)} E$$

Step 3. Reduce mod DE as stated in condition (2.11)

$$\mathbb{Y}^{(k,0)} E \qquad \text{on the surface } E = 0$$

Step 4. Set to 0 and split equations by powers of derivatives $u_{(k)}$. We now obtain a list of linear homogeneous equations. Note that this is not quite the determining equations for equivalence group yet.

Step 5. Append the determining equations for the constraint system (from Step 1) into the list of DEs. Further split by the power of a since ξ, η do not depend on a; only α depends on a. Now, we have a list of determining equations for the infinitesimals of the equivalence group.

Finally, by solving for the infinitesimals ξ, η and α we have the equivalence group for the DEs system E.

Example 7. Apply the above to the nonlinear heat equation again (denoted as NLH),

$$\begin{aligned} u_t + q_x &= 0, \\ q &= -K(u)u_x, \qquad \text{where } K \neq 0 \end{aligned} \tag{2.24}$$

with the vector field

$$\mathbb{Y} = \xi\frac{\partial}{\partial x} + \tau\frac{\partial}{\partial t} + \eta\frac{\partial}{\partial u} + \chi\frac{\partial}{\partial q} + \kappa\frac{\partial}{\partial K} \tag{2.25}$$

where ξ, τ, η, χ depend on (x, t, u, q) and κ depends on (x, t, u, q, K). Since K does not depend on x, t, q, we have the following constraint equations

$$K_x = 0, \qquad K_t = 0, \qquad K_q = 0 \tag{2.26}$$

We need two prolongations of the vector field $\mathbb{Y}$. First, prolong $\mathbb{Y}$ for the main system (2.24) up to order 1

$$\mathbb{Y}^{(1,0)} = \mathbb{Y} + \eta_{(x)}\frac{\partial}{\partial u_x} + \eta_{(t)}\frac{\partial}{\partial u_t} + \chi_{(x)}\frac{\partial}{\partial q_x} + \chi_{(t)}\frac{\partial}{\partial q_t}$$

where

$$\begin{aligned} \eta_{(x)} &= D_x\eta - u_x D_x\xi - u_t D_x\tau \\ \eta_{(t)} &= D_t\eta - u_x D_t\xi - u_t D_t\tau \\ \chi_{(x)} &= D_x\chi - q_x D_x\xi - q_t D_x\tau \\ \chi_{(t)} &= D_t\chi - q_x D_t\xi - q_t D_t\tau \end{aligned} \tag{2.27}$$

and D_x, D_t are total derivatives

$$D_x = \frac{\partial}{\partial x} + u_x \frac{\partial}{\partial u} + q_x \frac{\partial}{\partial q}$$
$$D_t = \frac{\partial}{\partial t} + u_t \frac{\partial}{\partial u} + q_t \frac{\partial}{\partial q}$$

Second, prolong $\mathbb{Y}$ for the constraint system (2.26) up to order 1

$$\mathbb{Y}^{(0,1)} = \mathbb{Y} + \alpha_{(x)} \frac{\partial}{\partial K_x} + \alpha_{(t)} \frac{\partial}{\partial K_t} + \alpha_{(u)} \frac{\partial}{\partial K_u} + \alpha_{(q)} \frac{\partial}{\partial K_q}$$

where

$$\begin{aligned}
\alpha_{(x)} &= \tilde{D}_x \alpha - K_x \tilde{D}_x \xi - K_t \tilde{D}_x \tau - K_u \tilde{D}_x \eta - K_q \tilde{D}_x \chi \\
\alpha_{(t)} &= \tilde{D}_t \alpha - K_x \tilde{D}_t \xi - K_t \tilde{D}_t \tau - K_u \tilde{D}_t \eta - K_q \tilde{D}_t \chi \\
\alpha_{(u)} &= \tilde{D}_u \alpha - K_x \tilde{D}_u \xi - K_t \tilde{D}_u \tau - K_u \tilde{D}_u \eta - K_q \tilde{D}_u \chi \\
\alpha_{(q)} &= \tilde{D}_q \alpha - K_x \tilde{D}_q \xi - K_t \tilde{D}_q \tau - K_u \tilde{D}_q \eta - K_q \tilde{D}_q \chi
\end{aligned} \tag{2.28}$$

and $\tilde{D}_x, \tilde{D}_t, \tilde{D}_u, \tilde{D}_q$ are total derivatives

$$\tilde{D}_x = \frac{\partial}{\partial x} + K_x \frac{\partial}{\partial K}, \qquad \tilde{D}_t = \frac{\partial}{\partial t} + K_t \frac{\partial}{\partial K},$$
$$\tilde{D}_u = \frac{\partial}{\partial u} + K_u \frac{\partial}{\partial K}, \qquad \tilde{D}_q = \frac{\partial}{\partial q} + K_q \frac{\partial}{\partial K}$$

To find the equivalence group for NLH, we follow the steps (p.29):

Step 1. Apply $\mathbb{Y}^{(0,1)}$ to the constraint equations (2.26): we obtain the following determining equations

$$\begin{aligned}
&\kappa_x = 0, && \kappa_t = 0, && \kappa_q = 0, \\
&\eta_x = 0, && \eta_t = 0, && \eta_q = 0
\end{aligned} \tag{2.29}$$

Step 2. Apply the prolonged vector field $\mathbb{Y}^{(1,0)}$ to the NLH system (2.24),

$$\begin{aligned}\mathbb{Y}^{(1,0)}\left(u_t + q_x\right) &= \eta_{(t)} + \chi_{(x)}\\ \mathbb{Y}^{(1,0)}\left(q + Ku_x\right) &= \chi + K\eta_{(x)} + u_x\kappa\end{aligned}$$

where $\eta_{(t)}, \chi_{(x)}, \eta_{(x)}$ are given by (2.27).

Step 3. Restrict to surface NLH by substituting $u_t = -q_x$, $u_x = -q/K$ to get

$$\begin{aligned}q_x(-\eta_u + \tau_t + \chi_q - \xi_x) + q_t\left(\frac{q}{K}(\xi_q + \tau_u) - \tau_x\right)\\ +q_x^2(-\tau_u - \xi_q) + \frac{q}{K}(\xi_t - \chi_u) + \chi_x = 0\\ q_x(q\xi_q + K\tau_x - q\tau_u) + q_x^2(K\tau_q) + \chi - q\eta_u + q\xi_x\\ -\frac{q^2}{K}\xi_u - \frac{q}{K}\kappa = 0\end{aligned}$$

Step 4. Set it to 0 then split them by powers of q_x, q_t to get determining equations:

$$\begin{aligned}K\chi - Kq\eta_u + Kq\xi_x - q^2\xi_u - q\kappa = 0, \qquad \tau_u + \xi_q = 0,\\ -\eta_u + \tau_t + \chi_q - \xi_x = 0, \qquad q\xi_q = 0,\\ q\xi_t + K\chi_x - q\chi_u = 0, \qquad \tau_q = 0,\\ K\tau_x - q\tau_u = 0\end{aligned}$$

Step 5. Append (2.29) which we derived earlier into the list of DEs. Further split by powers of K. Now we have a list of determining equa-

tions for the infinitesimals of the equivalence group:

$$\begin{aligned}
&\xi_{xx} = 0, && \tau_x = 0, \quad \eta_x = 0, && \chi_x = 0,\\
&\xi_t = 0, && \tau_u = 0, \quad \eta_t = 0, && \chi_t = 0,\\
&\xi_u = 0, && \tau_q = 0, \quad \eta_q = 0, && \chi_u = 0,\\
&\xi_q = 0, && \tau_{tt} = 0, \quad \eta_u = -\xi_x + \tau_t + \tfrac{1}{q}\chi, && \chi_q = \tfrac{1}{q}\chi,\\
&\kappa = K\,(2\xi_x - \tau_t)
\end{aligned}$$

Finally solving the determining equations for equivalence group and collecting vector fields respect to the constants, we have the following equivalence vector fields:

$$\begin{array}{ccc}
 & \dfrac{\partial}{\partial x}, \quad \dfrac{\partial}{\partial t}, & x\dfrac{\partial}{\partial x} + 2t\dfrac{\partial}{\partial t} - q\dfrac{\partial}{\partial q},\\
\dfrac{\partial}{\partial u}, & u\dfrac{\partial}{\partial u} + q\dfrac{\partial}{\partial q}, & x\dfrac{\partial}{\partial x} + t\dfrac{\partial}{\partial t} + K\dfrac{\partial}{\partial K}
\end{array}$$

The first three are symmetries, they apply to all nonlinear heat equations (2.18) and have trivial action on $K(u)$. It is really the other ones that are of interest.

The equivalence group is very useful in group classification because it can help to remove parameters when classifying symmetries. For example, the CRC Handbook [28] systematically uses the equivalence group to remove parameters and simplify the form of the symmetry classification. In addition, some symmetry classification methods have used the equivalence transformation as part of their process *during* classification [4, 8, 25].

For DEs connected by an equivalence transformation, their symmetry groups are identical apart from a change of coordinates. Therefore, all equations that are equivalent should be together in a symmetry classifica-

tion. In other words, good criteria for splitting cases should therefore be invariant under the transformations of the equivalence group. We will use this fundamental observation in developing new methods in Chapter 4.

Chapter 3

Computer Algebra in Symmetry Analysis

Now that we have covered the background on symmetry analysis in §2, we understand that symmetry analysis contains systematic algebraic manipulations and involves tedious calculation. The complexity of expressions increases rapidly as the numbers of independent and dependent variables and arbitrary elements increase, and as the order of the DE system increases. Therefore, computer algebra is in fact well-suited for doing symmetry analysis. As a result, the computer algebra community has been using symmetry methods as an important basis for solving DEs [66]. For example, some popular computer algebra systems such as Maple and Mathematica include package(s) that provide a suite of symmetry methods. And many more symmetry analysis programs have been produced for their own purpose.

3.1 Symmetry Analysis Packages

Computer algebra implementations for symmetry analysis started in the early 1980s. Schwarz's REDUCE package SPDE [58] derives and often successfully solves the determining equations for Lie point symmetries with minimal intervention by the user. Another REDUCE program NUSY by Nucci [29, chap. 14] also generates the determining equations for Lie point, generalised ('Lie-Bäcklund') and approximate symmetries, and then it provides interactive tools to solve them. Head [22] introduced a muMath program LIE, a program which can do many different types of symmetry analysis and also can compute the Lie vectors and their commutators. Due to the limitations of muMath, the program LIE is bounded by the 256 KB of memory for program and workspace. Thus, for a program with limited size, LIE is indeed remarkable in its achievement. Another package called SYMMGRP.MAX [12] by Champagne et al. allows the calculation of symmetry groups of arbitrarily large and complicated systems of DEs on relatively small computers. However, these packages/programs are specialised in symmetry analysis only, their purpose is to find symmetries of DEs (i.e. find and solve determining equations of DEs system).

In around 1990, Schwarz [57, 60], Reid [51, 52] and others began to change the implementations for symmetry analysis, separating the process into three parts (as shown in figure 3.1): (i) derive determining equations, (ii) reduce them, and (iii) solve them whenever possible.

The process of finding the determining equations can be easily done by tools such as LIESYMM by Carminati [10] et al. in Maple and PDELIE by Vafeades [65] in REDUCE. Now, steps (ii) and (iii) are done by general purpose computer algebra packages, which are not specific to symmetry analysis at all. Solving the determining equations is highly dependent

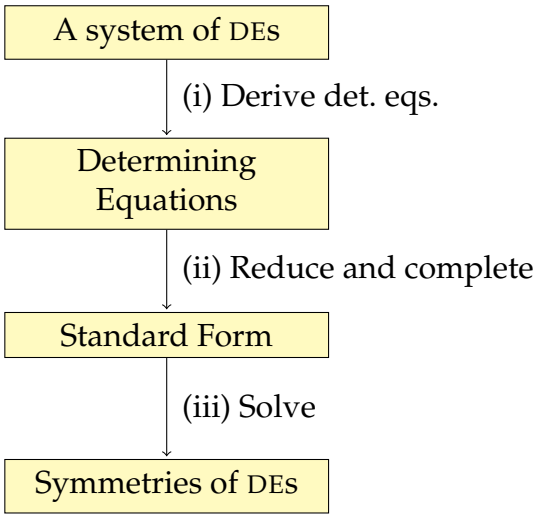


Figure 3.1: Steps for symmetry analysis. From a system of DEs, we (i) find the defining equations, (ii) reduce and complete them, and then (iii) solve them. The solutions are the symmetries of DEs.

on the complexity of the determining equations themselves. Therefore, simplifying the determining equations ("reduction" step) becomes a key part in symmetry analysis. More information reviewing computer algebra software for symmetry analysis can be found in [23, 24].

3.2 Differential Reduction & Completion Algorithms

A differential reduction & completion (DRC) algorithm is a method for simplifying polynomially nonlinear DEs to give a better chance of solving them. It reduces DEs to a certain form (which will be referred as 'standard form' in this thesis) that contains all the integrability conditions. Here is a summary of the advantages of using a DRC algorithm:

- Reducing the complexity of the DE system.

- Better chance to solve DE system.
- Extracting information from a system of DEs (such as the count of its solutions) instead of solving it.

The kernel of a DRC algorithm is reducing a DE system to a form where a local analytic existence-uniqueness (E-U) theorem can be applied to count solutions and extract other information without solving DEs. Some examples of E-U theories are:

- *Cauchy–Kovalevskaya theory* [55]: Show that certain terms in a power series can be specified freely, with all others determined by the DEs, and show the power series converges with positive radius of convergence.
- *Cartan–Kähler theory* [33]: A geometric theory using differential forms.
- *Riquier–Janet theory* [57]: Relies on isolating a derivative in each DEs, and is algorithmic only for linear PDEs.

These E-U theorems can be categorised into two types: *geometric* (e.g. Cartan–Kähler theory) and *algebraic* (e.g. Riquier–Janet theory). Some DRC algorithms like Cartan–Kuranishi [34] use geometric E-U theorems. However, an algebraic E-U theory, Riquier–Janet theory, is a common departure point for 'nonlinear' DRC methods such as differential Gröbner basis [11], Rosenfeld–Gröbner [7], and RIF [54, 67].

The development of DRC algorithms and implementation of packages were started in about 1990. Schwarz designed an algorithm INVOLUTIONSYSTEM [59, 60] which is based on the theory of Riquier and Janet. This DRC algorithm is to transform a linear system of PDEs into involutive form, it also may be applied repeatedly to determine a universal

Gröbner basis [40]. Furthermore, INVOLUTIONSYSTEM is able to determine the size of a Lie symmetry group without having to integrate the determining equations.

Mansfield developed a "differential" generalization of Buchberger's algorithm for Gröbner basis to find a *differential Gröbner basis* [40] (DGB for short) for the differential ideal generated by the DEs. However, there are a few technical difficulties to ensure termination. The KOLCHIN-RITT algorithm for DGB was implemented into a Maple package called DIFFGROB2 [40, 41, 42, 44]. If an expression is found in which the highest derivative term occurs in a factor raised to a power (i.e. differential ideal is non-radical) then the DGB method may fail [43, 44].

A REDUCE program called CRACK, developed by Wolf and Brand [68], uses the idea of Gröbner basis. This program attempts the solution of an overdetermined system of ODEs or PDEs with at most polynomial nonlinearities. Even if the DEs system is not solved, it may be possible to find analytic properties of solutions directly from the DEs.

A different REDUCE package DIMSYM by Sherring and Prince [64] was inspired by Head's program LIE, but is larger and further developed. DIMSYM can bring the determining equations to normal form and also works for systems of linear homogeneous DEs (not necessarily obtained from symmetry analysis). Furthermore, this program can find various types of symmetries, isolate special cases, etc.

The Rosenfeld–Gröbner algorithm decomposes the radical differential ideal generated by a set of algebraic DEs into 'characterisable components' [7]. Various improvements of the Rosenfeld–Gröbner algorithm have been proposed in [7]. They all avoid the factorization problem (a.k.a. factorization free). This algorithm is the first decomposition algorithm in differen-

tial algebra that has been actually implemented up to our knowledge – it forms an integral part of `diffalg` package in Maple. A more efficient implementation of the algorithm in C language by F. Boulier can be found at http://www.lifl.fr/~boulier/BLAD/.

The RIF algorithm is the DRC algorithm which we will be using in this thesis. The development history of RIF algorithm started from Reid's algorithm STANDARD_FORM [50, 52]. This algorithm has its roots in the classical Riquier–Janet theory. Instead of using the monomials of Riquier-Janet theory, it uses an equivalence class approach to avoid creating redundant equations and to provide a standard form of the systems to which it is applied. The STANDARD_FORM starts with a system of DEs and a matrix which gives a complete ranking on the derivatives appearing in the system. Then the DEs system will be reduced until it has all integrability conditions included and no more differential/algebra redundancies. In [50, 53], Reid et al. showed that structure constants $C_{i,j}^{k}$ of a Lie symmetry algebra can be found from the standard form of the determining equations without solving them.

Reid and McKinnon extend Reid's STANDARD_FORM algorithm and build a recursive algorithm called RSOLVE_PDESYS which can find particular solutions of a linear system of PDEs using only ODE solution techniques.

Later, an improved algorithm was developed by Reid et al. [54, 67] which combines features of geometric involutive form algorithms and the STANDARD_FORM algorithm. This algorithm uses a finite number of differentiation and algebraic operations to simplify any analytic nonlinear system of DEs to a *'reduced involutive form'* (RIF). The RIF form (i.e. standard form made by RIF) contains the integrability condition of the system,

such that at a point where a constant rank condition is satisfied, one can pose initial data that uniquely specifies a formal power series solution of the DEs.

A Maple package `rifsimp`, which was built by Wittkopf [67], uses the RIF algorithm. `Rifsimp` is a powerful, robust and efficient program. It contains a large number of user options such as rankings, case splits,..., etc. `Rifsimp` has therefore become one of the most widely used package for Maple users. In fact, Maple's `pdsolve` command [66, help on `DEtools`], which is the most commonly used package for solving DEs, calls `rifsimp` as a front end to reduce DEs before solving them. In this thesis, we use `rifsimp` and the RIF algorithm to classify symmetries, and therefore RIF is described in more detail in §3.2.1.

In conclusion, DRC algorithms become the major step for solving/-analysing DEs. Some widely used DRC algorithms such as differential Gröbner Basis, Rosenfeld–Gröbner, and RIF have been implemented into computer algebra systems, and they have been widely used by users. As example of performing symmetry analysis in Maple, we could first use LIESYMM to derive the determining equations of DEs; then we perform `rifsimp` to reduce them into RIF form, from which we can extract some information about the DEs; and finally we solve the determining equations using `pdsolve` (i.e. a function for solving DEs) [66, help on `DEtools`].

3.2.1 The RIF Algorithm

The RIF algorithm is a DRC algorithm for simplifying analytic systems of nonlinear PDEs to a form which can be easily transformed into an involutive form. Such a form is called "reduced involutive form" (referred to as RIF form) [54]. There are some nice features in the RIF algorithm. First,

the algorithm does terminate in finitely many steps [56]. Second, it only involves differentiation and elimination, no integration is involved during the process. Moreover, the result of the RIF process, RIF form, contains geometric properties of PDE systems, such as the dimension of the solution space of the system. Even though RIF form is coordinate-dependent, it can be easily transformed into a system which has involutive geometric properties. From RIF form it is also possible to specify local initial value problems and implicitly determine Taylor series expansions of the solutions which satisfy these initial value problems.

RIF was developed by Reid et al. [54, 67]; the algorithm is based on previous work from Reid's STANDARD_FORM (which is based on Riquier-Janet theorem) [52]. The RIF algorithm uses STANDARD_FORM for linear systems (i.e. Riquier-Janet theorem) and there are further extensions to the nonlinear case. The RIF algorithm was implemented as a Maple package called `rifsimp` [54], and it was later improved by Wittkopf [67]. `Rifsimp` has been widely used for simplifying DE system, especially in symmetry analysis, where the algorithm RIF is able to reduce the determining equations of DEs system (i.e. linear homogeneous system) into a simpler form which can count the number of symmetries of the system.

The use of ranking is a crucial aspect in the RIF algorithm (as in all algebraic DRC methods). The ranking is defined on the set of derivatives of dependent variables from a given DE system. To set a ranking in the RIF algorithm, the following conditions need to be met:

- The ranking is a total ordering (satisfies transitivity and trichotomy).
- For all I, J, L, symmetric multi-indices on $\{1, \ldots, n\}$,

$$u_I^k < u_J^k \implies u_{IL}^k < u_{JL}^k$$

for all $|L| \geq 0$.

- For symmetric multi-indices H on $\{1, \ldots, n\}$,

$$u^k < u^k_H$$

for all $|H| \geq 1$.

The role of these ranking properties in ensuring termination of RIF is described in [56].

The highest ranked derivative occurring in a given DE is called the *leading derivative* (or leader) of the equation. The DEs can then be split into two categories: those whose leading derivative occurs linearly (the 'leading linear' PDEs) and the remaining ones (the 'leading nonlinear' PDEs). A critical fact for RIF is that differentiation of any leading nonlinear PDE will yield leading linear ones.

Example 8. Consider the following DEs

$$\eta_t - u\eta_x - \eta_{xxx} = 0, \qquad \xi_{xx} - \xi_{xu} = 0, \qquad -3\xi_x + \tau_t = 0 \tag{3.1}$$

where ξ, τ, η depend on (x, t, u). We can set the ranking of two derivatives u^j_I and u^l_J to be as follows:

1. Rank by order of derivative: if $|I| < |J|$ then $u^j_I < u^l_J$.
2. If order of derivative is equal then break ties according to which dependent variables are present: $\eta < \tau < \xi$.
3. If still tied then break ties lexicographically by $u < t < x$.

The DEs (3.1) can be re-written as follows:

$$\eta_{xxx} = -\eta_t + u\eta_x, \qquad \xi_{xx} = \xi_{xu}, \qquad \xi_x = \tfrac{1}{3}\tau_t$$

where $\eta_{xxx}, \xi_{xx}, \xi_x$ are leading derivatives, as specified by the given ranking.

For the case of linear PDE systems, the process of the RIF algorithm consists of three kinds of operations. We will explain these via simple examples.

Operation 1. *Reduce a system of* DEs *modulo an equation.* For example, let $\mathcal{E}$ be a DEs system

$$\xi_{xx} = \xi_x + \eta, \qquad \xi_{xt} = 2\tau_x + \tau_t, \qquad \xi_{xu} = u\tau_x$$

To reduce $\mathcal{E}$ modulo the given equation $\tau_x = 0$, we substitute for τ_x, replacing it by 0 in $\mathcal{E}$.The system is reduced to

$$\xi_{xx} = \xi_x + \eta, \qquad \xi_{xt} = \tau_t, \qquad \xi_{xu} = 0$$

Operation 2. *Prolong an equation* by differentiating. For example, consider the equation

$$\tau_x = \xi + \eta$$

Differentiating with respect to x gives

$$\tau_{xx} = \xi_x + \eta_x$$

Operation 3. *Form integrability conditions between two equations.* For exam-

ple, consider these two equations $\mathcal{A}$, $\mathcal{B}$

$$\mathcal{A} : \xi_x = \tau_t, \qquad \mathcal{B} : \xi_t = \eta - \tau$$

where ξ_x, ξ_t are leading derivatives. Forming the integrability condition of $\mathcal{A}$ and $\mathcal{B}$ amounts to equating mixed partials ($\xi_{xt} = \xi_{tx}$):

$$\begin{aligned} \frac{\partial}{\partial t}(\tau_t) &= \frac{\partial}{\partial x}(\eta - \tau) \\ \tau_{tt} &= \eta_x - \tau_x \end{aligned} \tag{3.2}$$

So, a new equation (3.2) has been found by forming the integrability condition. This new equation may or may not contain new information.

Before we describe the steps of the RIF algorithm,we need the following definition.

Definition 3.2.1 (Reduced orthonomic form)**.** A system of DEs E is in *reduced orthonomic form* if

i Each leading derivative in E appears only once in E

ii No nontrivial derivative of any leading derivative appears in E

The steps of the RIF algorithm with a choice of ranking for a 'linear' PDE system are as follows:

Step 1. Set up a PDE system in reduced orthonomic form, denoted as $\mathcal{S}$.

Step 2. Form the integrability conditions on all pairs of equations in $\mathcal{S}$ (Operation 3). New equations found here are denoted as $\mathcal{I}$.

Step 3. Where $\mathcal{I}$ involves a derivative u^j_{IL} of a leader u^j_I from $\mathcal{S}$, prolong the equation from $\mathcal{S}$ by computing its L-th derivative (Operation 2). Let $\mathcal{K}$ denote the system $\mathcal{S}$ with these new prolonged equations appended.

Step 4. Reduce $\mathcal{I}$ modulo $\mathcal{K}$ (Operation 1), we denote these reduced equations by $\mathcal{R}$.

Step 5. If $\mathcal{R}$ is empty then finish. Otherwise, append $\mathcal{R}$ to $\mathcal{S}$ and repeat Step 1 to Step 5 again.

Once the process of RIF is finished, the final form is "fully reduced and completed" and referred as completed form or rif form. The steps only involve differentiation and substitution.

In the following example, we are going to show how RIF reduces a list of equations into a completed form with specified ranking.

Example 9. Consider a list of linear homogeneous equations

$$\begin{aligned} &\xi_x - \frac{1}{3}\tau_t = 0, \quad \tau_x = 0, \quad \eta_{uu} = 0, \qquad -\eta_t + u\eta_x + \eta_{xxx} = 0, \\ &\xi_u = 0, \quad \tau_u = 0, \quad \eta_{xu} = 0, \quad 3\eta_{xxu} + \eta + \xi_t + \frac{2}{3}u\tau_t = 0 \end{aligned} \tag{3.3}$$

where ξ, τ, η depend on (x, t, u).

These equations are in fact the determining equations for point symmetries of the KdV equation $u_t = u_{xxx} + uu_x$. We set ranking of two derivatives u^j_I and u^l_J as follows:

1. Rank by order of derivative: if $|I| < |J|$ then $u^j_I < u^l_J$.

2. If order of derivative is equal then break ties according to which dependent variables are present: $\eta < \tau < \xi$.

3. If still tied then break ties lexicographically by $u < t < x$.

Now we execute the RIF algorithm,

Step 1. Put equations (3.3) into orthonomic form, denoted by $\mathcal{S}$,

$$\begin{aligned}
&\xi_x = \frac{1}{3}\tau_t, \quad && \tau_x = 0, \quad && \eta_{uu} = 0, \quad && \eta_{xxx} = \eta_t - u\eta_x, \\
&\xi_u = 0, && \tau_u = 0, && \eta_{xu} = 0, && \eta_{xxu} = -\frac{1}{3}\eta - \frac{1}{3}\xi_t - \frac{2}{9}u\tau_t
\end{aligned}$$

Step 2. Form the integrability conditions on all pairs of DEs in $\mathcal{S}$. Some new equations are found by forming the integrability conditions. For example, the first and fifth equations $\xi_x = \frac{1}{3}\tau_t$, $\xi_u = 0$ give

$$\begin{aligned}
&\frac{\partial}{\partial u}(\frac{1}{3}\tau_t) = \frac{\partial}{\partial x}(0) \\
\implies \qquad &\frac{1}{3}\tau_{ut} = 0
\end{aligned}$$

Similarly the fourth (η_{xxx}) and eighth (η_{xxu}) equations give

$$\begin{aligned}
&\frac{\partial}{\partial u}(\eta_t - u\eta_x) = \frac{\partial}{\partial x}\left(-\frac{1}{3}\eta - \frac{1}{3}\xi_t - \frac{2}{9}u\tau_t\right) \\
\implies \qquad &\eta_{ut} - \eta_x - u\eta_{xu} = -\frac{1}{3}\eta_x - \frac{1}{3}\xi_{xt} - \frac{2}{9}u\tau_{xt} \\
\implies \qquad &\eta_{ut} - \frac{2}{3}\eta_x - u\eta_{xu} + \frac{1}{3}\xi_{xt} + \frac{2}{9}u\tau_{xt} = 0
\end{aligned}$$

Altogether there are 8 integrability conditions to form, of which two are trivial, the other 6 are:

$$\begin{aligned}
&\frac{1}{3}\tau_{ut} = 0, && 3\eta_u + 3\xi_{ut} + 2\tau_t + 2u\tau_{ut} = 0, \\
&\eta_{ut} - u\eta_{xu} - \eta_x = 0, && \eta_{uut} - u\eta_{xuu} - 2\eta_{xu} = 0,
\end{aligned}$$

$$-\frac{1}{3}\eta - \frac{1}{3}\xi_t - \frac{2}{9}u\tau_t = 0, \quad \eta_{ut} - \frac{2}{3}\eta_x - u\eta_{xu} + \frac{1}{3}\xi_{xt} - \frac{2}{9}u\tau_{xt} = 0$$

These equations are denoted as $\mathcal{I}$.

Step 3. The system $\mathcal{I}$ involves the following derivatives:

$$\{\xi_{ut}, \xi_t, \xi_{xt}, \tau_{ut}, \tau_t, \tau_{xt}, \eta_u, \eta_{ut}, \eta_{xu}, \eta_x, \eta_{uut}, \eta_{xuu}, \eta\}$$

Of these, the following are derivatives of leaders from $\mathcal{S}$:

$$\{\xi_t, \xi_{ut}, \xi_{xt}, \tau_{ut}, \eta_{xu}, \eta_{xuu}\}$$

The needed prolongations of equations from $\mathcal{S}$ are therefore:

$$\begin{aligned} &\xi_{ut} = 0, \qquad \xi_{xt} = \frac{1}{3}\tau_{tt}, \qquad \tau_{ut} = 0, \qquad \tau_{xt} = 0, \\ &\eta_{uut} = 0, \qquad \eta_{xuu} = 0 \end{aligned} \tag{3.4}$$

Step 4. Reduce $\mathcal{I}$ modulo equations ($\mathcal{S}$+(3.4)). Several equations have reduced to triviality (i.e. $0 = 0$), the remaining reduced equations $\mathcal{R}$ are

$$3\eta_u + 2\tau_t = 0, \qquad \eta_{ut} - \eta_x = 0, \qquad \eta_{ut} - \frac{2}{3}\eta_x + \frac{1}{9}\tau_{tt} = 0$$

Step 5. Because $\mathcal{R}$ is not empty, we need to append $\mathcal{R}$ to $\mathcal{S}$ and repeat Step 1 to 5 again.

We repeat Step 1 to 5 until $\mathcal{R}$ is empty (i.e. no more new equations are found in Step 4). Once it finishes, the final updated $\mathcal{S}$ is fully reduced and

completed. The fully reduced and completed RIF form is:

$$\begin{aligned} \xi_x &= -\frac{1}{2}\eta_u & \tau_x &= 0 & \eta_x &= 0 \\ \xi_t &= -\eta + u\eta_u & \tau_t &= -\frac{3}{2}\eta_u & \eta_t &= 0 \\ \xi_u &= 0 & \tau_u &= 0 & \eta_{uu} &= 0 \end{aligned}$$

For using RIF in symmetry classification in §3.3, there are two kinds of dependent variables which need to be treated differently. For this purpose we need to use the idea of block elimination ranking.

For a block ranking, we partition dependent variables u into two disjoint classes $u = (v, w)$, where all derivatives of v are ranked lower than w: we write $v \ll w$. Then RIF gives a nice partition of the equations. In particular, the low ranked variables v satisfy their own sub-system which means the high-ranked variables w have been eliminated.

Example 10. Consider the same list of linear homogeneous equations we used in Example 9,

$$\begin{aligned} \xi_x - \frac{1}{3}\tau_t = 0, \quad \tau_x = 0, \quad \eta_{uu} = 0, \qquad -\eta_t + u\eta_x + \eta_{xxx} = 0, \\ \xi_u = 0, \quad \tau_u = 0, \quad \eta_{xu} = 0, \quad 3\eta_{xxu} + \eta + \xi_t + \frac{2}{3}u\tau_t = 0 \end{aligned} \tag{3.5}$$

where ξ, τ, η depend on (x, t, u). This time we specify a block ranking in which

1. $\{\eta\} \ll \{\xi, \tau\}$

This means that all derivatives of η are ranked lower than all derivatives of ξ, τ. Then we complete the ranking as follows:

2. Rank by order of derivative: if $|I| < |J|$ then $u^j_I < u^l_J$.

3. If order of derivative is equal then break ties according to which dependent variables are present: $\xi < \tau$.

4. If still tied then break ties lexicographically by $u < t < x$.

The completed and reduced RIF form is as follows:

$$\xi_x = -\frac{1}{2}\eta_u, \qquad \xi_t = -\eta + u\eta_u, \qquad \xi_u = 0, \tag{3.6a}$$

$$\tau_x = 0, \qquad \tau_t = -\frac{3}{2}\eta_u, \qquad \tau_u = 0, \tag{3.6b}$$

$$\eta_x = 0, \qquad \eta_t = 0, \qquad \eta_{uu} = 0 \tag{3.6c}$$

The equations (3.6a) and (3.6b) contain ξ, τ, η, whereas equations (3.6c) only contain η, which is a subsystem to itself.

For the case of a nonlinear PDE system, the RIF algorithm contains two more outer loops: 'constant rank loop' and 'spawning loop'. More detail on the RIF algorithm for the nonlinear case can be found in [67]. An important feature for nonlinear PDE is that there can be a need to split into cases. For example, consider a nonlinear ODE (Clairaut equation)

$$y_x^2 + xy_x - y = 0$$

where y depends on x. Using RIF gives splits on

Case 1. If $2y_x + x \neq 0$ then $y_{xx} = 0$. (With a constraint $y_x^2 + xy_x - y = 0$.)

Case 2. If $2y_x + x = 0$ then $y = -\frac{1}{4}x^2$.

A practical implementation of RIF may process a system of DEs in a different sequence for efficiency reasons. For example, it may initially work on a subset of the equations, and leaving some parts of the system

as 'unclassified' to defer until later. As long as all equations are eventually dealt with then the algorithm still works.

3.3 Symmetry Classification Using RIF

Recalling the symmetry classification problem as described in §2.2, the determining equations for symmetry of the DEs system are often large, complicated and overdetermined. We can use a DRC method such as RIF to reduce complexity of the determining equations so that it is more likely we can solve them and find the infinitesimals. Moreover, if a system of DEs involves arbitrary elements, using RIF is one way to classify symmetries in a systematic way.

Figure 3.2 shows how the RIF algorithm is applied to symmetry classification problems.

For a system of DEs which involves arbitrary elements, there are two types of dependent variables in the determining equations: *the arbitrary elements* and *the infinitesimals*. If we use RIF to classify symmetries for such a system then we need to make sure RIF selects splitting conditions which only involve arbitrary elements. Therefore, an elimination ranking needs to be set in a form where all derivatives of all arbitrary elements are ranked lower then infinitesimals:

$$\{\text{all arbitrary elements}\} \ll \{\text{all infinitesimals}\}.$$

With this ranking we can be sure that the leading derivatives of determining equations are infinitesimals. The coefficients of these leading derivatives only involve arbitrary elements so are guaranteed to split on arbitrary elements only.

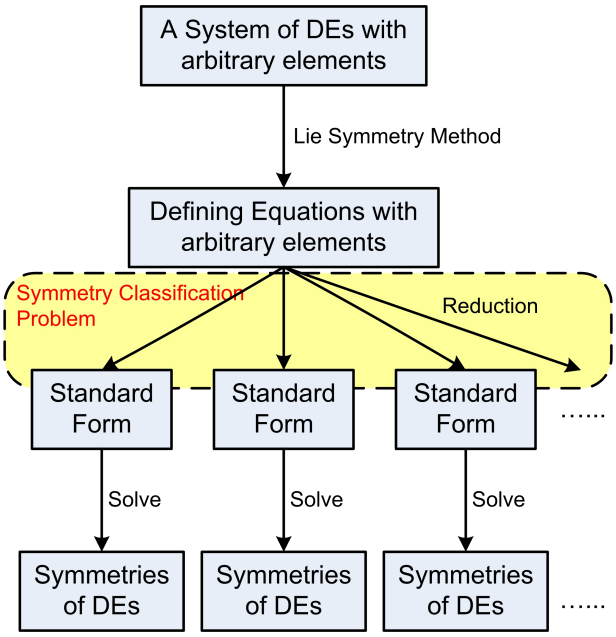


Figure 3.2: Steps for using RIF for classifying symmetries (when there exist an arbitrary element in a system of DEs)

Note that this type of elimination ranking is called *block elimination ranking*, see §3.2.1 for more detail. We shall demonstrate the block elimination ranking to perform symmetry classification.

Example 11. For the nonlinear heat equation (2.18) with the assumption that $K \neq 0$, the determining equations for point symmetries (as listed in example 4) contains the following variables:

- *arbitrary element:* K
- *infinitesimal generators:* ξ, τ, η, χ

We need to make sure that K is ranked lower than other variables; therefore, we must specify a ranking for which

1. $\{K\} \ll \{\xi, \tau, \eta, \chi\}$

To complete the ranking we choose for $\{\xi, \tau, \eta, \chi\}$, the ranking of two derivatives u_I^j and u_J^l as follows:

2. Rank $\{\xi, \tau, \eta\} \ll \{\chi\}$
3. If tied, rank by order of derivative: if $|I| < |J|$ then $u_I^j < u_J^l$.
4. If order of derivative is equal then break ties according to which dependent variables are present: $\xi < \tau < \eta$.
5. If still tied then break ties lexicographically by $x < t < u < q$.

Consider again the determining equations (2.19). Using the ranking above, we can further reduce these determining equations to

$$\begin{aligned}
&\xi_u = \xi_q = \tau_x = \tau_u = \tau_q = \eta_{uu} = \eta_q = 0, \qquad \xi_{xx} = \frac{1}{2}\frac{K_u}{K}\eta_x, \\
&\tau_t = 2\xi_x - \frac{K_u}{K}\eta, \qquad \eta_{xx} = \frac{1}{K}\eta_t, \qquad \eta_{xu} = -\frac{3}{4}\frac{K_u}{K}\eta_x - \frac{1}{2}\frac{1}{K}\xi_t, \\
&\chi = -K\eta_x + q\eta_u - q\xi_x + q\frac{K_u}{K}\eta, \qquad \frac{K_u}{K}\eta_u + \left(\frac{K_u}{K}\right)_u \eta = 0
\end{aligned} \tag{3.7}$$

Selecting the last equation from (3.7), the leading derivative is η_u. Therefore there is a case split on K_u. In this thesis, we will refer to these expressions as 'pivots'. In this case, the pivot K_u causes the system to split into two branches:

- If $K_u = 0$ then we can't solve for η_u. This sub-branch is the linear heat equation.
- If $K_u \neq 0$ then we can solve for η_u. After further reduction, a new equation

$$\left(\frac{K}{K_u}\right)_{uu} \eta = 0 \tag{3.8}$$

is discovered.

The determining equations from branch $K_u \neq 0$ can be split further. This time, from Eq. (3.8), the leading derivative is η, so our next choice of pivot is $\left(\frac{K}{K_u}\right)_{uu}$. The equations can keep splitting until no more splittings can be made, and then the classification is complete.

3.3.1 Symmetry Classification Using `Rifsimp`

The RIF algorithm is implemented in Maple as a computer algebra package called `rifsimp`. The package `rifsimp` is one of the most widely used packages for classification problems in Maple. It provides a large amount of useful functionality for users such as control of ranking, case splittings etc. In this section, we will illustrate with the previous example (Example 11) to show how to use `rifsimp` to classify symmetries.

Example 12. To classify symmetries using `rifsimp`, one requires the determining equations for point symmetries and a ranking. We first use the determining equations we derived earlier in Example 4 (or using a Maple command `PDEtools:-DeterminingPDE` [66, help on `DeterminingPDE`]) with specified ranking shown on p.52. Assume the `DEtools` package is loaded in Maple worksheet (i.e. `with(DEtools);`), and that we have assigned a list of the determining equations as `detEqs`. We type the command in a Maple worksheet:

```
rifOutput:=rifsimp(detEqs,[[chi],[eta, xi, tau],[K]],casesplit);
```

where

`detEqs` = list of determining equations for symmetries

`[[chi],[eta, xi, tau],[K]]` $= \{K\} \ll \{\eta, \xi, \tau\} \ll \{\chi\}$ (ranking)

$$\texttt{casesplit} = \text{indicate to perform classification}$$
$$\texttt{rifOutput} = \text{results from } \texttt{rifsimp}$$

Here, we need to specify `casesplit` so `rifsimp` knows to split into multiple cases if required. Without `casesplit`, `rifsimp` only returns the generic case which means all pivots are assumed to be nonzero.

`Rifsimp` uses the RIF algorithm with specified ranking to classify symmetries. A table is returned whose entries (also tables themselves) contain information such as solved equations (`Solved`), splitting conditions (`Case`), and assumptions (`Pivots`) for each case. We refer this returned table (e.g. `rifOutput` above) as rif-output. More details about rif-output can be found in [66, help on `rifsimp, output`].

The rif-output can also be viewed graphically by calling `caseplot` [66, help on `DEtools[caseplot]`], such a plot we refer to as a classification tree or case plot. Figure 3.3 shows the classification tree for the nonlinear heat equation. It has three pivots (i.e. splitting conditions):

$$p1 = K_u$$
$$p2 = -4KK_{uu} + 7K_u^2$$
$$p3 = KK_uK_{uuu} - 2KK_{uu}^2 + K_u^2K_{uu}$$

and each pivot splits into $=$ and $\neq$ branches. For example in Case 1 (the generic case) where the conditions are

$$K_u \neq 0 \qquad -4KK_{uu} + 7K_u^2 \neq 0 \qquad KK_uK_{uuu} - 2KK_{uu}^2 + K_u^2K_{uu} \neq 0$$

with its corresponding determining equations. The result of this case con-

tains three symmetries: two translation and one scaling. Other cases such as Case 2 contains four symmetries and Case 3 has five symmetries. The linear case (Case 4) appears where $K_u = 0$ which has infinitely many symmetries (because it is linear).

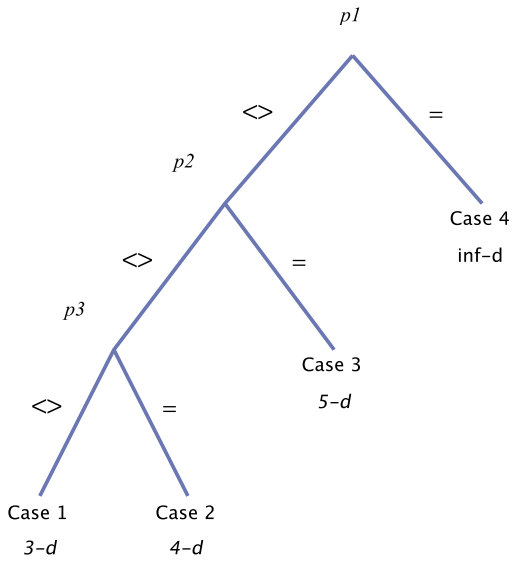


Figure 3.3: Classification tree for nonlinear heat equations. Note that $p1 = K_u, p2 = -4KK_{uu} + 7K_u^2, p3 = KK_uK_{uuu} - 2KK_{uu}^2 + K_u^2K_{uu}$. Case 1 is the generic case where K can be arbitrary. Therefore, there are at least three symmetries for nonlinear heat equations.

The choice of the next unclassified equation to split is also important during classification. As its default, `rifsimp` chooses the smallest length equation to be the next candidate. `Rifsimp` provides a few other options (`pivselect` option for `rifsimp`) such as `smalleq` (the smallest length equation), `smallpiv` (the smallest length of leading coefficient), `lowrank` (the lowest ranked leading derivative), ..., etc. Details of other options in `rifsimp` can be viewed in [67, Appendix A].

A weakness of `rifsimp` is that it does not do symmetry classification in a very intelligent way. For example, it makes an arbitrary choices of basis

and case splits are not guaranteed to be invariant under the equivalence group. This can lead to the presence of parameters in the group structure which are provably removable. The method of Lisle and Reid [38] is guaranteed to give invariant case splittings; however, some parts of their process are unsuited to computer algebra, and their method therefore cannot be called algorithmic. This leads to the objective for developing a method that is invariant under the equivalence group, which will be discussed in next chapter.

3.4 Results from Commutative Algebra

We state various results from Commutative Algebra that we will be using in §4.2. These are mostly from [15].

Let K be a field. The set of all polynomials in $x_1, \ldots, x_n$ with coefficients in K is a polynomial ring, denoted by $K[x_1, \ldots, x_n]$. A subset $I \subseteq K[x_1, \ldots, x_n]$ is an ideal if it satisfies:

i $0 \in I$.

ii If $f, g \in I$, then $f + g \in I$.

iii If $f \in I$ and $h \in K[x_1, \ldots, x_n]$, then $hf \in I$.

Definition 3.4.1 (Ideal Generated)**.** Let $f^1, \ldots, f^s$ be polynomials in the polynomial ring $K[x_1, \ldots, x_n]$. Then we set

$$\langle f^1, \ldots, f^s \rangle = \left\{ \sum_{i=1}^{s} h_i f^i : h_1, \ldots, h_s \in K[x_1, \ldots, x_n] \right\}$$

The crucial fact is that $\langle f^1, \ldots, f^s \rangle$ is an ideal, the polynomial ideal generated by $f^1, \ldots, f^s$.

A Gröbner basis is a basis for an ideal in a polynomial ring which allows us to solve problems about polynomial ideals in an algorithmic or computational fashion. A Gröbner basis can be found algorithmically by the Buchberger algorithm, and it is implemented in computer algebra e.g. Maple package `Groebner`. We need to use the following defining property of Gröbner basis [15]:

Theorem 3.4.2 (Ideal membership)**.** *Let $G = \{g^1, \ldots, g^q\}$ be a Gröbner basis for ideal I. A polynomial h is in ideal I if and only if h reduces to 0 modulo by $\{g^1, \ldots, g^q\}$.*

'Reduction' here means taking remainder on polynomial division.

In this thesis, we are interested in solution of polynomial equations:

Definition 3.4.3 (Algebraic Variety)**.** Let $f^1, \ldots, f^s$ be polynomials in the polynomial ring $K[x_1, \ldots, x_n]$. Let L be an extension field of K. Then we set

$$V(f^1, \ldots, f^s) = \{(a_1, \ldots, a_n) \in L^n : f^i(a_1, \ldots, a_n) = 0 \text{ for all } 1 \leq i \leq s\}$$

We call $V(f^1, \ldots, f^s)$ the variety defined by $f^1, \ldots, f^s$. Thus, a variety $V(f^1, \ldots, f^s) \subseteq L^n$ is the set of all solutions of the system of equations $f^1(x_1, \ldots, x_n) = \cdots = f^s(x_1, \ldots, x_n) = 0$.

Note that polynomials have coefficients in field K, but solutions are found in the extension field L. Usually L is to be algebraically closed. For example, we might have polynomials over the field of rationals $\mathbb{Q}$ (where we can compute exactly) but we look for solutions over the field $\mathbb{C}$ of complex numbers.

Theorem 3.4.4 (Hilbert Nullstellensatz)**.** *[15] Let h be a polynomial in the ring $K[x_1, \ldots, x_n]$ and the system $E = (f^1, \ldots, f^s)$ where $(f^1, \ldots, f^s)$ are polynomials in $K[x_1, \ldots, x_n]$. Let $V(E)$ be the variety in an algebraically closed extension field L. The Nullstellensatz says if h vanishes on $V(E)$ then h is in the radical of $\langle f^1, \ldots, f^s \rangle$. That is,*

$$(h)^m \in \langle f^1, \ldots, f^s \rangle \qquad \text{for some } m \geq 1$$

and $(h)^m$ means the power of h to m.

Note that this theorem is invalid if we seek solutions of polynomial equations in a field that is not algebraically closed.

Finally, we note that for the radical of an ideal, the following processes are algorithmic: testing if an ideal is radical; testing if a given polynomial is in the radical of an ideal; and finding a Gröbner basis for the radical of an ideal. These are implemented in Maple (see Help pages for `PolynomialIdeals` package, `IsRadical`, `RadicalMembership`, `Radical` [66]).

Chapter 4

Invariance Checking from Determining Equations

Based on the comments at the end of §3.3.1, we wish to make sure a whole classification tree is invariant under the action of the equivalence group. One way to do this is to select invariant case splitting conditions during classification as much as possible. Therefore, this leads to an idea of building a checking method which can check if DEs (i.e. splitting conditions) are invariant under the action of the equivalence group or not. In this chapter, we develop such a method. Furthermore, we wish the process to work entirely at the level of determining equations so that the method is purely algorithmic and can be implemented in computer algebra. A good feature of working with determining equations is that our method can deal with finite- and infinite-dimensional Lie groups equally easily.

In this chapter, we will show how to use the symmetry condition (see Theorem 2.1.4) for developing the method, and how to apply this method to a differential reduction & completion (DRC) method such as RIF, for the classification of symmetries.

4.1 Projection of Equivalence Group Action

Recall from §2.3 that an equivalence transformation is a point transformation on $X \times U \times A$ (space of independent and dependent variables and arbitrary elements) that projects down to space $X \times U$ (space of independent and dependent variables). This projection of the equivalence group is described by Ibragimov [30, §2.2.7], who denotes it by π_1.

Example 13. In Example 6, we found the equivalence operators of nonlinear heat equation (NLH) system:

$$\mathbb{Y} = (ax+b)\frac{\partial}{\partial x} + (ct+d)\frac{\partial}{\partial t} + (e+a-c)q\frac{\partial}{\partial q} + (eu+f)\frac{\partial}{\partial u} + (2a-c)K\frac{\partial}{\partial K} \tag{4.1}$$

where a, b, c, d, e, f are arbitrary constants. As described in [30, §2.2.7], the projection π_1 amounts to dropping the K component

$$\pi_1(\mathbb{Y}) = (ax+b)\frac{\partial}{\partial x} + (ct+d)\frac{\partial}{\partial t} + (e+a-c)q\frac{\partial}{\partial q} + (eu+f)\frac{\partial}{\partial u} \tag{4.2}$$

Ibragimov [30, §2.2.7] also describes another projection π_2 to a space consisting of the arbitrary elements a and those variables that a depend on. That is, π_2 drops any variables that arbitrary elements do *not* depend on.

Example 14. For the NLH system the arbitrary element K depends only on u. The projection π_2 of equivalence operator (4.1) amounts to dropping the (x, t, q) components, giving a vector field on (u, K):

$$\pi_2(\mathbb{Y}) = (eu+f)\frac{\partial}{\partial u} + (2a-c)K\frac{\partial}{\partial K} \tag{4.3}$$

Doing projections π_1, π_2 is quite clear when one has the equivalence

operator $\mathbb{Y}$ explicitly (e.g. (4.1)). However in this thesis, we are trying to work at the level of determining equations – not with the explicit form of equivalence operator. So the question is how to carry out projections π_1, π_2 when we know only the determining equations of the equivalence group. The answer is through elimination ranking.

In §3.2.1, we mentioned that block elimination ranking can force a subset of the dependent variables to obey a sub-system of its own. For projection π_1, we start with a vector field

$$\mathbb{Y} = \sum_{i=1}^{n} \xi_i \frac{\partial}{\partial x^i} + \sum_{j=1}^{m} \eta_j \frac{\partial}{\partial u^j} + \sum_{l=1}^{p} \alpha^l \frac{\partial}{\partial a^l} \tag{4.4}$$

and want to drop the α^l components, leaving only ξ^i, η^j. So in the determining equations, choose a ranking for which all derivatives of ξ^i, η^j are ranked lower than all derivatives of α^l:

$$\{\xi^1, \ldots, \xi^n, \eta^1, \ldots, \eta^m\} \ll \{\alpha^1, \ldots, \alpha^p\}$$

This will force out a subsystem for (ξ, η). Also, it is imposed in advance that ξ^i, η^j depend only on (x, u). So this subsystem is effectively for (ξ, η) as dependent variables and (x, u) as independent. That is, the subsystem corresponds to the π_1 projection of the equivalence operator.

Example 15 (π_1 Projection). For the nonlinear heat equation (2.24), take the equivalence operator

$$\mathbb{Y} = \xi \frac{\partial}{\partial x} + \tau \frac{\partial}{\partial t} + \chi \frac{\partial}{\partial q} + \eta \frac{\partial}{\partial u} + \kappa \frac{\partial}{\partial K}$$

where ξ, τ, χ depend on (x, t, q, u), η depends on u, and κ depends on (u, K). The unreduced determining equations for the infinitesimals, found

by the method from §2.3, are:

$$\begin{gathered}\xi_q = 0, \qquad \tau_q = 0, \qquad \tau_u = 0, \qquad \tau_x = 0,\\ \chi_x = 0, \qquad -\chi_u + \xi_t = 0, \qquad -\xi_x + \tau_t + \chi_q - \eta_u = 0,\\ qK\xi_x - q^2\xi_u + K\chi - qK\eta_u - q\kappa = 0 \end{gathered} \tag{4.5}$$

We wish to project the equivalence operator down to $X \times U$ (independent/dependent variables only).

Specify an elimination ranking in which:

1. $\{\xi, \tau, \chi, \eta\} \ll \{\kappa\}$.

This means that all derivatives of ξ, τ, χ, η are ranked lower then all derivatives of κ. This will force out a subsystem of the determining equations for (ξ, τ, χ, η). We complete the ranking as follows:

2. Rank by order of derivative: if $|I| < |J|$ then $u^j_I < u^l_J$.
3. If order of derivative is equal then break ties according to which dependent variables are present: $\eta < \chi < \tau < \xi$.
4. If still tied then break ties lexicographically by $u < q < t < x < K$.

To get the π_1 projection of the equivalence operator, we use a differential reduction and completion algorithm (i.e. RIF) to reduce the determining equations (15) by the ranking specified above. This can be done by the computer algebra package `rifsimp`. The reduced system is:

$$\xi_{xx} = 0, \qquad \tau_{tt} = 0, \qquad \chi_q = \frac{\chi}{q}, \qquad \eta_u = -\xi_x + \frac{\chi}{q} + \tau_t, \tag{4.6a}$$

$$\kappa = K(\xi_x - \tau_t) \tag{4.6b}$$

where ξ depends on x, τ depends on t, χ depends on q, and η depends on u only.

Due to the elimination ranking above, the subsystem (4.6a) for the π_1 projected vector field (4.2) is forced out.

For the π_2 projection, we start with the same vector field $\mathbb{Y}$ (4.4) and want to drop variables which the arbitrary elements a do not depend on. Partition the set of variables $V = \{x^1, \ldots, x^n, u^1, \ldots, u^n\}$ into two disjoint subsets as follows:

$$Z = \{y \in V \mid a^l \text{ depends on } y \text{ for some } l\}$$
$$\bar{Z} = \{y \in V \mid a^l \text{ not dependent on } y \text{ for all } l\}$$

Let the infinitesimals corresponding to $Z, \bar{Z}$ be $\Theta, \bar{\Theta}$ respectively.

Therefore, to eliminate $\bar{\Theta}$, we choose a ranking in which:

$$\{\alpha^1, \ldots, \alpha^p\} \cup \Theta \ll \bar{\Theta}$$

Moreover, because the infinitesimals in Θ do not depend on a^l, the ranking can be further refined as follows:

$$\Theta \ll \{\alpha^1, \ldots, \alpha^p\} \ll \bar{\Theta}$$

Using such a ranking will force out a subsystem for α and the infinitesimals Θ.

Example 16 (π_2 Projection). Continuing with Example 15, we now wish to project the equivalence group to the space of arbitrary elements, that is, (u, K) space where K is the arbitrary element and u is its dependencies. Partition the variables $V = \{x, t, q, u\}$ into $Z = \{u\}$, $\bar{Z} = \{x, t, q\}$, and the

corresponding infinitesimals are $\Theta = \{\eta\}$, $\bar{\Theta} = \{\xi, \tau, \chi\}$.

Specify an elimination ranking in which:

1. $\{\eta\} \ll \{\kappa\} \ll \{\xi, \tau, \chi\}$

This means all derivatives of η are ranked lower than all derivatives of other infinitesimals, followed by all derivatives of κ ranked lower than all derivatives of ξ, τ, χ. We complete the ranking as follows:

2. Rank by order of derivative: if $|I| < |J|$ then $u^j_I < u^l_J$.
3. If the order of derivatives is equal then break ties according to which dependent variables are present: $\chi < \tau < \xi$.
4. If still tied then break ties lexicographically by $q < t < x < u < K$.

Starting from the unreduced determining equations for the infinitesimals (15), and using the same procedure as in Example 15 to give a reduced form:

$$\eta_{uu} = 0, \tag{4.7a}$$

$$\kappa_K = \frac{\kappa}{K}, \tag{4.7b}$$

$$\xi_x = -\frac{1}{q}\chi + \eta_u + \frac{1}{K}\kappa, \qquad \tau_t = -\frac{2}{q}\chi + 2\eta_u + \frac{1}{K}\kappa, \qquad \chi_q = \frac{1}{q}\chi \tag{4.7c}$$

where η depends on u, and κ depends on K only. Due to the elimination ranking above, the subsystem (4.7a, 4.7b) for the π_2 projected vector field (4.3) is forced out.

4.2 Issues with Symmetry Condition

Refer back to the symmetry sufficient condition from Theorem 2.1.4, for a given DE system $E = 0$ and vector field $\mathbb{X}$, if

$$\mathbb{X}^{(k)} E = 0, \qquad \text{whenever } E = 0 \tag{4.8}$$

then $\mathbb{X}$ generates symmetries of the system E.

To enforce "whenever $E = 0$", it is usually done by substituting leading derivatives from the system E (see Step 3 in Example 2). However, when the given system of DEs is 'nonlinear' (e.g. $ff_{xxx}^2 + f_x f_{xx} f_{xxx} + f_{xx}^2) = 0$ or 'reducible variety' (e.g. $y_{xx}(z_{xx} + z) = 0$), how do we enforce $E = 0$ in condition (4.8)? Even for some systems (e.g. the following example) in which leading derivatives are easy to derive, there is still a problem with enforcing symmetry sufficient condition (4.8).

Example 17. Consider a system of DEs $E = 0$ where

$$E = \{y_{xx}(z_{xx} + z)\} \tag{4.9}$$

with the corresponding vector field

$$\mathbb{X} = \xi \frac{\partial}{\partial x} + \eta \frac{\partial}{\partial y} + \zeta \frac{\partial}{\partial z}. \tag{4.10}$$

where ξ, η, ζ depend on (x, y, z). If y_{xx} is the leading derivative of the system E then to find symmetries of the system E we apply the symmetry condition 2.1.4,

$$\mathbb{X}^{(2)} E = 0 \qquad \text{whenever } E = 0 \tag{4.11}$$

We could try to enforce "whenever $E = 0$" by substituting the leading

derivative (i.e. $y_{xx} = 0$) from the system. It gives *wrong* symmetries. One reason is because the substitution for $y_{xx} = 0$ does not include any information about $(z_{xx} + z)$, and that means we are no longer dealing with same system E. Therefore, the symmetries we found are not exactly from the system E.

Note that this type of system is not a pathological case in symmetry analysis, in fact we come across some DE systems like this when we perform symmetry classification by using RIF. Therefore, it is no longer satisfactory to just substitute out leading derivatives. We need to examine how to enforce the condition "whenever $E = 0$" more closely.

Olver [47, §2.3] states an alternative *local & analytic* form of symmetry condition, namely locally there exist functions $h_\nu(x, u, u_{(k)})$ such that

$$\mathbb{X}^{(k)} E = \sum_{\nu=1}^{s} h_\nu(x, u, u_{(k)}) f^\nu(x, u, u_{(k)})$$

What we want is to do this but not *locally*, but instead using the Nullstellensatz theorem to get an analogous algebraic condition.

Referring back to the symmetry condition Theorem 2.1.4, for $\mathbb{X}^{(k)} E$ to vanish whenever $E = 0$ is to ask that $\mathbb{X}^{(k)} E$ vanish on an algebraic variety.

However for each polynomial equation f^l in the system E, the form $\mathbb{X}^{(k)} f^l$ is

$$\mathbb{X}^{(k)} f^l = \sum_{j,I} p_{jI}(x, u, u_{(k)}) \zeta_I^j \qquad \text{where } \zeta \text{ is some } \{\xi, \eta\}$$

The p_{jI} are polynomials in jet variables $x, u, u_{(k)}$ and ζ_I^j are some derivatives of infinitesimals ξ, η.

Suppose $E = \{f^1, \ldots, f^s\}$ are polynomials in jet variables $x, u, u_{(k)}$ and

that $\{f^1, \ldots, f^s\}$ is a Gröbner basis for a radical ideal. Note the following facts:

(i) If all polynomial coefficients p_{jI} vanish on $V(f^1, \ldots, f^s)$ then $\mathbb{X}^{(k)} f^l$ vanishes on $V(f^1, \ldots, f^s)$.

(ii) So vanishing of all p_{jI} is sufficient for $\mathbb{X}$ to be a symmetry (see Theorem 2.1.4).

(iii) But by Nullstellensatz Theorem 3.4.4, p_{jI} vanish on $V(f^1, \ldots, f^s)$ if and only if $p_{jI} \in \langle f^1, \ldots, f^s \rangle$.

(iv) And since $\{f^1, \ldots, f^s\}$ is a Gröbner basis, $p_{jI} \in \langle f^1, \ldots, f^s \rangle$ if and only if p_{jI} reduces to 0 modulo $\{f^1, \ldots, f^s\}$.

Therefore, (i) – (iv) immediately imply:

Theorem 4.2.1 (Algebraic Symmetry Condition). *Let $E = \{f^1, \ldots, f^s\}$ be polynomials in jet variables $x, u, u_{(k)}$ and $\{f^1, \ldots, f^s\}$ a Gröbner basis for a radical ideal. For each polynomial f^l in E, the form $\mathbb{X}^{(k)} f^l$ is*

$$\mathbb{X}^{(k)} f^l = \sum_{j,I} p_{jI}(x, u, u_{(k)}) \zeta_I^j \qquad \text{where } \zeta \text{ is some } \{\xi, \eta\}$$

and where p_{jI} are polynomials in jet variables $x, u, u_{(k)}$ and ζ_I^j are some derivatives of infinitesimals ξ, η. If all p_{jI} reduce to 0 mod $\{f^1, \ldots, f^s\}$ then $\mathbb{X}$ is a symmetry of $E = 0$.

If the given DEs are not a Gröbner basis for a radical ideal then the algebraic symmetry condition is no longer valid. However, a Gröbner basis for a radical ideal can be found from the system of DEs by the method mentioned in §3.4.

Therefore, the following algorithm tests a sufficient condition for a vector field $\mathbb{X}$ to be a symmetry of the system $E = 0$. For a system of DEs $E = \{f^1 = 0, \ldots, f^s = 0\}$,

Step 1. Find a Gröbner basis $G = \{g^1, \ldots, g^s\}$ for the radical of the ideal $\langle f^1, \ldots, f^s \rangle$.

Step 2. Find $\mathbb{X}^{(k)}G$.

Step 3. Reduce polynomial coefficients in $\mathbb{X}^{(k)}G$ mod G.

Step 4. If all reduce to 0 then $\mathbb{X}$ is a symmetry of $G = 0$ (and therefore of $E = 0$).

For the case of *finding* symmetries of the system E, Step 4 is replaced by:

Step 4* Collect the expression by powers of jet variables u^j_I of order $|I| \geq 1$, and set coefficients to 0. These are the determining equations for symmetries.

Step 5* Solve the determining equations to get symmetries.

Example 18. Let's re-visit Example 17. This time we use the algebraic symmetry condition in Theorem 4.2.1, and follow the steps above to find symmetries.

Step 1. The system E is already a Gröbner basis G for a radical ideal, so

$$G = \{y_{xx}(z_{xx} + z)\}$$

Step 2. We start with prolonging the vector field $\mathbb{X}$ (4.10) up to order 2,

$$\mathbb{X}^{(2)} = \mathbb{X} + \eta_{(x)}\frac{\partial}{\partial y_x} + \zeta_{(x)}\frac{\partial}{\partial z_x} + \eta_{(xx)}\frac{\partial}{\partial y_{xx}} + \zeta_{(xx)}\frac{\partial}{\partial z_{xx}}$$

where

$$\eta_{(x)} = D_x\eta - y_x D_x\xi \qquad \eta_{(xx)} = D_x\eta_{(x)} - (D_x\xi)y_{xx}$$
$$\zeta_{(x)} = D_x\zeta - z_x D_x\xi \qquad \zeta_{(xx)} = D_x\zeta_{(x)} - (D_x\xi)z_{xx}$$

and the total derivative is

$$D_x = \frac{\partial}{\partial x} + y_x\frac{\partial}{\partial y} + z_x\frac{\partial}{\partial z} + y_{xx}\frac{\partial}{\partial y_x} + z_{xx}\frac{\partial}{\partial z_x}$$

Then $\mathbb{X}^{(2)}G$ is

$$\{(z_{xx}+z)\eta_{(xx)} + y_{xx}\zeta + y_{xx}\zeta_{(xx)}\}$$

Step 3. Reduce polynomial coefficients in $\mathbb{X}^{(2)}G$ mod G. That is, divide by $y_{xx}(z_{xx}+z)$ to give remainder:

$$\begin{aligned}
&\xi_{zz}z_{xx}{z_x}^2y_x + 2\,\xi_{yz}z_{xx}z_x{y_x}^2 + \xi_{yy}z_{xx}{y_x}^3 + \xi_{zz}y_{xx}{z_x}^3\\
&+ 2\,\xi_{yz}y_{xx}{z_x}^2y_x + \xi_{yy}y_{xx}z_x{y_x}^2 + \xi_z{z_{xx}}^2y_x - \eta_{zz}z_{xx}{z_x}^2\\
&+ \left(-2\,\eta_{yz} + 2\,\xi_{xz}\right)z_{xx}z_xy_x + \left(2\,\xi_{xy} - \eta_{yy}\right)z_{xx}{y_x}^2 + \xi_y{y_{xx}}^2z_x\\
&+ \left(2\,\xi_{xz} - \zeta_{zz}\right)y_{xx}{z_x}^2 + \left(-2\,\zeta_{yz} + 2\,\xi_{xy}\right)y_{xx}z_xy_x - \zeta_{yy}y_{xx}{y_x}^2\\
&+ \xi_{zz}z{z_x}^2y_x + 2\,\xi_{yz}zz_x{y_x}^2 + \xi_{yy}z{y_x}^3 - \eta_z{z_{xx}}^2 - 2\,\eta_{xz}z_{xx}z_x\\
&+ \left(\xi_zz - 2\,\eta_{xy} + \xi_{xx}\right)z_{xx}y_x - \zeta_y{y_{xx}}^2 + \left(\xi_{xx} - 2\,\zeta_{xz} - 3\,\xi_zz\right)y_{xx}z_x\\
&+ \left(-2\,\zeta_{xy} - 2\,\xi_yz\right)y_{xx}y_x - \eta_{zz}z{z_x}^2 - \left(2\,\eta_{yz} - 2\,\xi_{xz}\right)zz_xy_x\\
&+ \left(2\xi_{xy} - \eta_{yy}\right)z{y_x}^2 - \left(\eta_{xx} + \eta_zz\right)z_{xx} + \left(z\zeta_z - \zeta_{xx} - \zeta - 2z\xi_x\right)y_{xx}\\
&- 2\,\eta_{xz}zz_x - \left(2\,\eta_{xy} - \xi_{xx}\right)zy_x - \eta_{xx}z
\end{aligned}$$

Step 4* Collecting powers from the jet variables y_x, z_x, y_{xx}, z_{xx} and equat-

ing to 0, the determining equations for symmetries are as follows:

$$\xi_x = 0, \qquad \xi_y = 0, \qquad \xi_z = 0$$
$$\eta_{xx} = 0, \qquad \eta_{xy} = 0, \qquad \eta_{yy} = 0$$
$$\zeta_y = 0, \qquad \zeta_{xx} = -\zeta + z\zeta_z, \qquad \zeta_{xz} = 0, \qquad \zeta_{zz} = 0$$

Step 5* Solve these determining equations. As a result, there are 7 symmetries:

$$\frac{\partial}{\partial x}, \quad \frac{\partial}{\partial y}, \quad x\frac{\partial}{\partial y}, \quad y\frac{\partial}{\partial y}, \quad \sin x\frac{\partial}{\partial z}, \quad \cos x\frac{\partial}{\partial z}, \quad z\frac{\partial}{\partial z}$$

4.3 Invariance Using Determining Equations

This section covers the development of an algebraic method for checking invariance which uses only determining equations. It is much more reasonable to stay with the determining equations rather than find the symmetry groups because we always can get the determining equations but we may fail to solve them. For example, the Cauchy–Riemann equations

$$\xi_x = \eta_y, \qquad \eta_x = -\xi_y$$

are determining equations for a Lie (pseudo-) group. It is not convenient to write the solution of the system, but it is easy to work with the system itself.

The process of developing the method starts from the idea of the symmetry condition Theorem 2.1.4. We will show how we approach and refine the method into our final form of the invariance checking method.

Let E be a system of DEs

$$E = \left\{ f^1, \ldots, f^s \right\}, \quad \text{where } f^\nu(x, u, u_{(k)}) = 0, \quad \nu = 1, \ldots s \tag{4.12}$$

which lives in $J^{(k)}(X, U)$, and let $\mathbb{X}$ be a vector field on the base space of independent and dependent variables as

$$\mathbb{X} = \sum_{i=1}^{n} \xi^i \frac{\partial}{\partial x^i} + \sum_{j=1}^{m} \eta^j \frac{\partial}{\partial u^j} \tag{4.13}$$

where each ξ^i, η^j depends on (x, u).

Suppose the components ξ, η of $\mathbb{X}$ (4.13) are constrained to satisfy certain determining equations (denoted as $\mathcal{Q}$). Here $\mathcal{Q}$ is a list of linear homogeneous equations with independent variables (x, u) and dependent variables ξ, η. Our question is: for all the vector fields $\mathbb{G}$ built as solutions of the determining equations $\mathcal{Q}$, *are all these vector fields* $\mathbb{G}$ *symmetries of the system* E*?* Our goal in this section is to answer this question. The obvious way to answer (first approach below) is not practical, we refine this in three stages until we have an efficient, algorithmic method.

First Approach The above question can be answered easily by applying the symmetry condition 2.1.4. Let $\mathbb{G}^{(k)}$ be a vector field which is built from solving the determining equations $\mathcal{Q}$. If the condition

$$\mathbb{G}^{(k)} E = 0, \qquad \text{whenever } E = 0 \tag{4.14}$$

is satisfied for all such $\mathbb{G}$ then these vector fields $\mathbb{G}^{(k)}$ are symmetries of the system E.

Example 19. Consider the question of whether all affine transformations

are symmetries of the second order linear ODE $y_{xx} = 0$, that is,

$$E = \{y_{xx}\} \tag{4.15}$$

Here dependent variable is y, independent variable is x. Let the vector field $\mathbb{X}$ be

$$\mathbb{X} = \xi \frac{\partial}{\partial x} + \eta \frac{\partial}{\partial y} \tag{4.16}$$

where ξ, η depend on (x, y). And affine vector fields are the solutions of the determining equations $\mathcal{Q}$

$$\xi_{xx} = \xi_{xy} = \xi_{yy} = 0, \qquad \eta_{xx} = \eta_{xy} = \eta_{yy} = 0 \tag{4.17}$$

To check if the system $E = 0$ (4.15) is invariant under affine vector fields according to the method described above, we need to get the prolonged vector fields $\mathbb{G}^{(2)}$ by first solving the determining equations $\mathcal{Q}$ (4.17),

$$\xi = a_{11}x + a_{12}y + b_1, \qquad \eta = a_{21}x + a_{22}y + b_2$$

where $a_{11}, a_{12}, a_{21}, a_2, b_1, b_2$ are constants. The vector field $\mathbb{G}$ has the form:

$$\mathbb{G} = (a_{11}x + a_{12}y + b_1)\frac{\partial}{\partial x} + (a_{21}x + a_{22}y + b_2)\frac{\partial}{\partial y} \tag{4.18}$$

Second we then prolong $\mathbb{G}$ up to order 2,

$$\mathbb{G}^{(2)} = \mathbb{G} + \eta_{(x)}\frac{\partial}{\partial y_x} + \eta_{(xx)}\frac{\partial}{\partial y_{xx}}$$

where the prolongation components are (2.8)

$$\begin{aligned}\eta_{(x)} &= a_{21} + (a_{22} - a_{11})y_x - a_{12}y_x^2\\ \eta_{(xx)} &= (a_{22} - 2a_{11})y_{xx} - 3a_{12}y_x y_{xx}\end{aligned}$$

Now we apply the symmetry condition (4.14):

$$\begin{aligned}\mathbb{G}^{(2)}y_{xx} &= \eta_{(xx)}\\ &= (a_{22} - 2a_{11} - 3a_{12}y_x)y_{xx}\end{aligned}$$

Reducing mod E as per the algebraic symmetry condition 4.2.1 gives

$$\begin{aligned}\mathbb{G}^{(2)}y_{xx} \mod y_{xx} &= (a_{22} - 2a_{11} - 3a_{12}y_x)y_{xx} \mod y_{xx}\\ &= 0\end{aligned}$$

Since the result is 0, all affine vector fields are symmetries of the system $y_{xx} = 0$.

Although the condition (4.14) meets our goal for testing invariance, the process is unreliable and definitely not algorithmic because it requires explicit solution of $\mathbb{Q}$. For example, this will fail in finding all conformal symmetries of $y_{xx} = 0$ because of the awkwardness of solving the Cauchy-Riemann equations (so we failed in the very first step of the method).

Second Approach In this approach, we try to avoid solving any PDEs so the process is algorithmic, and one way to do this is to work on the level of the determining equations.

Let $\mathbb{S}$ be the determining equations for point symmetries of the system E. Recall the question. To say all the vector fields $\mathbb{G}$ are symmetries of the

system E is equivalent to say all vector fields $\mathbf{G}$ are solutions of $\mathcal{S}$. This amounts to asking whether all solutions of $\mathcal{Q}$ are solutions of $\mathcal{S}$. If $\mathcal{Q}$ is in differentially completed form (see §3.2.1) then this can be answered by reduction.

The condition is summarised as follows: If $\mathcal{Q}$ is differentially complete and if

$$reduce(\mathcal{S}) \mod \mathcal{Q} \tag{4.19}$$

is 0 then every solution of $\mathcal{Q}$ is a solution of $\mathcal{S}$. We can conclude that all the solutions of $\mathcal{Q}$ are symmetries of system E.

Example 20. Consider the question of whether all conformal transformations of the xy-plane are symmetries of $y_{xx} = 0$. The conformal vector fields obey the determining equations $\mathcal{Q}$,

$$\xi_x = \eta_y, \qquad \eta_x = -\xi_y \tag{4.20}$$

(which are the Cauchy-Riemann equations). And the determining equations $\mathcal{S}$ for point symmetries can be found in (2.17),

$$\begin{aligned} \xi_{xx} &= 0 & 2\eta_{xy} - \xi_{xx} &= 0 \\ \eta_{yy} - 2\xi_{xy} &= 0 & -\xi_{yy} &= 0 \end{aligned} \tag{4.21}$$

We need to complete the system $\mathcal{Q}$, but in fact (4.20) is already complete with the ranking:

1. rank by order of derivative: if $|I| < |J|$ then $u_I^j < u_J^l$.
2. If order of derivative is equal then break ties lexicographically by $y < x$.
3. If still tied then break ties by $\eta < \xi$.

We substitute the complete form of $\mathcal{Q}$ into $\mathcal{S}$ (4.21). If we obtain the trivial equation (i.e. $0 = 0$), then all solutions of $\mathcal{Q}$ are also solutions of $\mathcal{S}$. We have

$$reduce(\mathcal{S}) \mod \mathcal{Q} = \{-\xi_{yy} = 0, \quad -\eta_{yy} = 0\} \tag{4.22}$$

so $\mathcal{S}$ does not reduce to 0 modulo $\mathcal{Q}$. Therefore, not all solutions of $\mathcal{Q}$ are solutions of $\mathcal{S}$, and we can conclude that not all conformal transformations are symmetries of $y_{xx} = 0$.

This time the process has the virtue of being algorithmic; however, the method is extremely wasteful with calculation, since it forms the point symmetry determining equations $\mathcal{S}$. But these are irrelevant – we are only interested in those symmetries that are built from the determining equations $\mathcal{Q}$.

Final Approach – The Invariance Checking Method (ICM) The third approach combines the best features of the two approaches above. The question answered by the second approach was *are all the solutions of* $\mathcal{Q}$ *solutions of* $\mathcal{S}$*?* But this should really be rephrased as *do the solutions of* $\mathcal{Q}$ *leave the system* $E = 0$ *invariant?* We now show how to answer this without knowing the solutions of $\mathcal{Q}$. The idea is to take account of the information from $\mathcal{Q}$ during the process of building the prolonged vector field $\mathbf{G}^{(k)}$.

As a result, we developed a better method – the invariance checking method (ICM). The ICM combines features of applying symmetry condition directly (from the first approach) and works on the level of determining equations (from the second approach). In other words, the ICM uses the symmetry condition (4.14) but in such a way that the components of the prolonged vector field $\mathbf{G}^{(k)}$ have been reduced by $\mathcal{Q}$. The procedures

of the ICM are as follows:

1. Form the prolonged operator $\mathbb{G}^{(k)}$.
2. Reduce its components ξ^i, η^j_I modulo $\mathcal{Q}$. And now we have a reduced prolonged vector field, denoted as $\mathbb{G}^{(k)}_{red}$.
3. Check if the symmetry condition

$$\mathbb{G}^{(k)}_{red} E = 0 \qquad \text{whenever } E = 0 \tag{4.23}$$

   is satisfied. (For instance using the method from §4.2).

All these steps are algorithmic: step 1 is just prolongation, step 2 is reduction (substitute out leading derivatives from $\mathcal{Q}$), step 3 is applying a vector field to the system E (differentiation) and reduction modulo E (polynomial division). So there is no need to solve PDEs $\mathcal{Q}$.

The first two steps only need to be done once for a given $\mathcal{Q}$. If there are several systems $E_1, E_2, \ldots$ to be tested for invariance, only the third step needs to be done on each system. And this is quite efficient.

Example 21. Consider again Example 20, the problem of checking if all conformal transformations are symmetries of the 2nd order ODE $y_{xx} = 0$. A vector field $\mathbb{G}$ (4.16) has components ξ, η constrained by the Cauchy-Riemann equations $\mathcal{Q}$,

$$\xi_x = \eta_y, \qquad \eta_x = -\xi_y \tag{4.24}$$

Our task is to use the invariance checking method (ICM) to test if the system $E = 0$ (i.e. $y_{xx} = 0$) is invariant under conformal vector fields which are built from this $\mathcal{Q}$. Following the steps of the ICM method above:

1. *Prolong the vector field* $\mathbb{G}$ *up to order* 2. The prolonged vector field $\mathbb{G}^{(2)}$ has the form

$$\mathbb{G}^{(2)} = \xi\frac{\partial}{\partial x} + \eta\frac{\partial}{\partial y} + \eta_{(x)}\frac{\partial}{\partial y_x} + \eta_{(xx)}\frac{\partial}{\partial y_{xx}}$$

where (see Example 1)

$$\begin{aligned}\eta_{(x)} =& \eta_x + y_x\eta_y - y_x\xi_x - y_x^2\xi_y\\ \eta_{(xx)} =& \eta_{xx} + 2y_x\eta_{xy} + y_x^2\eta_{yy} - y_x\xi_{xx} - 2y_x^2\xi_{xy} - y_x^3\xi_{yy}\\ &+ y_{xx}\left(\eta_y - 2\xi_x - 3y_x\xi_y\right)\end{aligned}$$

2. *Reduce the vector field components* ξ, η, $\eta_{(x)}$, $\eta_{(xx)}$ *modulo* $\mathcal{Q}$. We substitute the prolongation of Cauchy-Riemann (4.24)

$$\begin{aligned}&\xi_x = \eta_y, \qquad \eta_x = -\xi_y,\\ &\xi_{xx} = -\xi_{yy}, \quad \xi_{xy} = \eta_{yy}, \quad \eta_{xx} = -\eta_{yy}, \quad \eta_{xy} = -\xi_{yy}\end{aligned}$$

so that the prolonged vector field becomes

$$\mathbb{G}^{(2)}_{red} = \xi_{red}\frac{\partial}{\partial x} + \eta_{red}\frac{\partial}{\partial y} + \eta_{(x)red}\frac{\partial}{\partial y_x} + \eta_{(xx)red}\frac{\partial}{\partial y_{xx}}$$

where

$$\begin{aligned}\xi_{red} &= \xi\\ \eta_{red} &= \eta\\ \eta_{(x)red} &= -(1+y_x^2)\xi_y\\ \eta_{(xx)red} &= (-3y_xy_{xx})\xi_y - y_{xx}\eta_y - (y_x + y_x^3)\xi_{yy} - (1 + 2y_x - y_x^2)\eta_{yy}\end{aligned}$$

3. *Use the symmetry condition* (4.23) *to test if* $y_{xx} = 0$ *is invariant.*

$$\begin{aligned} \mathbb{G}^{(2)}_{red} y_{xx} &= \eta_{(xx)red} \\ &= (-3y_x y_{xx})\xi_y - y_{xx}\eta_y - (y_x + y_x^3)\xi_{yy} - (1 + 2y_x - y_x^2)\eta_{yy} \end{aligned} \tag{4.25}$$

Reducing this mod E (i.e. divide by y_{xx} to get remainder) gives

$$\mathbb{G}^{(2)}_{red} y_{xx} = (-y_x + y_x^3)\xi_{yy} + (-1 - y_x^2)\eta_{yy} \qquad \text{mod } y_{xx}$$

The result is not 0 so the DE is shown as 'non-invariant'. That is, $y_{xx} = 0$ is not invariant under action of the conformal group.

The left over terms above (or (4.22)) are the obstructions to being a symmetry. Setting them to 0, we obtain the additional determining equations

$$\xi_{yy} = 0, \qquad \eta_{yy} = 0.$$

For a conformal transformation (4.16) to be a symmetry of $y_{xx} = 0$, the additional determining equations must be satisfied.

The ICM method is clearly a much better method for checking invariance because it avoids construction of the determining equations for point symmetries $\mathcal{S}$. If the system $\mathcal{Q}$ contains simple equations, the ICM takes of advantage of the simplification. In our application in the next chapter, system $\mathcal{Q}$ will be the determining equations for the equivalence group and are usually fairly simple equations. Also we will need to check invariance of multiple systems $E_1, E_2, \ldots$ with respect to the same $\mathcal{Q}$. The second method (4.19) redoes each system from the beginning, but the ICM reduces modulo $\mathcal{Q}$ once only for all systems, so is much more efficient.

4.4 Invariance Checking in Symmetry Classification

In §3.3 we showed how to use a differential reduction and completion method – RIF – to classify symmetries. Unfortunately, the classification performed by RIF does not respect the equivalence group: the choice of basis and case splits are not guaranteed to be invariant under the equivalence group. However, the invariance checking method (ICM) from the previous section can be used to help RIF to improve the symmetry classification so that it does respect the equivalence group.

The classification tree is branched by pivots which are chosen by RIF. One way to check that the classification tree is invariant under the equivalence group action is to use the ICM method on all pivots (i.e. case splits) in the classification tree. If all pivots are invariant under equivalence group action then the whole classification is invariant under the equivalence group of the given system.

So we would like to check invariance for each pivot in the classification tree. For this we use the theory from §4.3. Before we apply the method, we first need to find the determining equations $\mathcal{Q}$ for equivalence group from the DE system E. The method for finding $\mathcal{Q}$ is described in §2.3. Let *piv* be a pivot in the classification tree, the steps for checking *piv* to be invariant under the action of equivalence group are as follows :

1. Form the prolonged operator $\mathbf{G}^{(k)}$ up to same order as in *piv* once and for all.

2. Reduce its components ξ^i, η^j_I modulo $\mathcal{Q}$ once for all. The reduced prolonged vector field is denoted as $\mathbf{G}^{(k)}_{red}$.

3. Check if the symmetry condition

$$\mathrm{G}_{red}^{(k)} piv = 0 \qquad \text{whenever } piv = 0$$

is satisfied. If so then piv is invariant under the equivalence group action. Else if it is not 0 then piv is not invariant.

There are two ways we can use the ICM method to check pivots (the steps above) in the symmetry classification problem:

1. *Label pivots from classification tree.* ***After*** we have a complete classification tree, we can sweep through the tree, test the pivots and label them as 'invariant' or 'not invariant'. It gives an idea of the relationship of the case splits to the equivalence group.

2. *Guide* RIF *during classification.* ***During*** classification process, when RIF is forced to split into cases, there is a list of candidates which are eligible to be the selected as pivots (referred as splitting candidates). The idea is to test these splitting candidates and choose an invariant candidate (if possible) to be the selected pivot. And we repeat this procedure at every split until the classification is complete. This is to help RIF to improve classification so that it now respects the equivalence group.

In the following sections, we will illustrate these two applications in more detail.

4.4.1 Label pivots from classification tree

Assume we have a complete classification tree such as the one shown in Figure 4.1. We would like to apply the steps above to check the invariance

of these pivots $p_1, \ldots, p_4$.

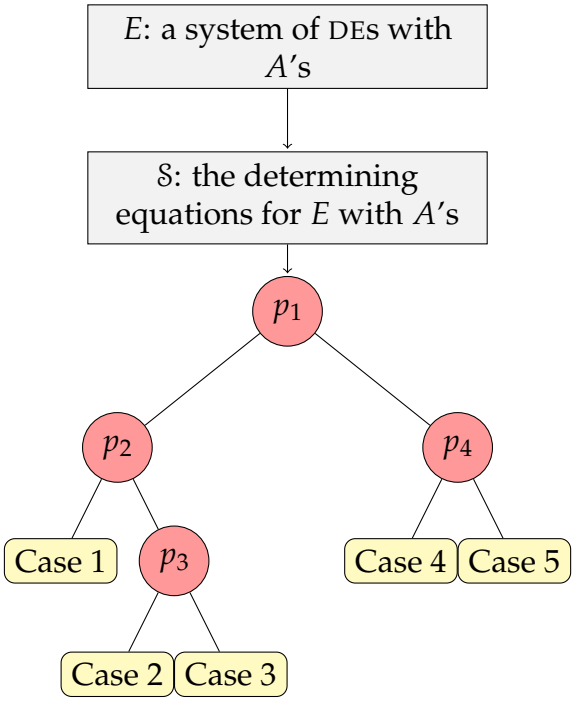


Figure 4.1: Symmetry classification by using RIF. The pivots $p_1, \ldots, p_4$ (shows in red circle) are the places where the invariance checking method is involved.

The idea of this application is to first check invariance on all pivots (using the steps above) from the completed classification tree, then provide additional information on these pivots by labeling invariant to be 'true' or 'false'. So it gives an idea on how much the classification tree respects the equivalence group. The whole process is efficient even for complicated case trees (e.g. more than 50 cases), the total number of splitting conditions are not big and the whole process only involves differentiation and reduction.

Example 22 ($(1+1)$ nonlinear heat equation).

$$\begin{aligned} u_t + q_x &= 0, \\ q &= -K(u)u_x, \qquad K(u) \neq 0 \end{aligned} \tag{4.26}$$

Using the result of symmetry classification from Example 12 in §3.3 (the classification case tree is showed in Figure 3.3), the following splitting conditions are obtained:

$$K_u = 0, \tag{4.27a}$$

$$-4KK_{uu} + 7K_u^2 = 0, \tag{4.27b}$$

$$KK_uK_{uuu} - 2KK_{uu}^2 + K_u^2K_{uu} = 0. \tag{4.27c}$$

To test whether the whole classification is invariant under the action of the equivalence group we can test these pivots one by one. That is, we can check if each splitting condition (i.e. (4.27)) is invariant or not using the method of §4.4. But first we need to find the determining equations for equivalence group.

The determining equations for equivalence group for system (4.26) (denoted as $\mathcal{Q}$) were derived in Example 7 in §2.3. Because pivots only involve variables u, K, we only need this part (π_2 projection, §4.1) of $\mathcal{Q}$:

$$\kappa_K = \frac{\kappa}{K}, \qquad \eta_{uu} = 0 \tag{4.28}$$

and the corresponding vector field

$$\mathbb{G} = \eta\frac{\partial}{\partial u} + \kappa\frac{\partial}{\partial K}$$

where η depends on u and κ depends on K only.

Our first question: *is* $K_u = 0$ (4.27a) *invariant under action of equivalence group?* The steps for the invariance checking for $K_u = 0$ are as follows:

1. Prolong the vector field $\mathbb{G}$ to order 1 (because K_u is a first order DE)

$$\mathbb{G}^{(1)} = \eta \frac{\partial}{\partial u} + \kappa \frac{\partial}{\partial K} + \kappa_{(u)} \frac{\partial}{\partial K_u} \tag{4.29}$$

   where

$$\kappa_{(u)} = \kappa_u + K_u \kappa_K - K_u \eta_u \tag{4.30}$$

   and $\kappa_{(u)}$ depends on (u, K, K_u).

2. Reduce the vector field component $\kappa_{(u)}$ by $\mathcal{Q}$ (4.28)

$$\begin{aligned} \eta_{red} &= \eta \\ \kappa_{red} &= \kappa \\ \kappa_{(u)red} &= \kappa_{(u)} \mod \mathcal{Q} \\ &= \frac{K_u}{K}\kappa - K_u \eta_u \end{aligned} \tag{4.31}$$

   The reduced version of prolonged vector field is

$$\mathbb{G}^{(1)}_{red} = \eta \frac{\partial}{\partial u} + \kappa \frac{\partial}{\partial K} + \left(-K_u \eta_u + \frac{K_u}{K}\kappa \right) \frac{\partial}{\partial K_u} \tag{4.32}$$

3. Check if $K_u = 0$ is invariant by applying the symmetry condition. We first find

$$\mathbb{G}^{(1)}_{red} K_u = -K_u \eta_u + \frac{K_u}{K}\kappa$$

then

$$\begin{aligned}\mathbb{G}^{(1)}_{red}K_u \mod K_u &= -K_u\eta_u + \frac{K_u}{K}\kappa \mod K_u \\ &= 0\end{aligned}$$

Therefore, $K_u = 0$ is tested as invariant under the action of the equivalence group.

Similarly the second splitting condition $-4KK_{uu} + 7K_u^2 = 0$ (4.27b) is tested as follows:

1. This time we are required to prolong the vector field to order 2 because of the second pivot (4.27b)

$$\mathbb{G}^{(1)} = \eta\frac{\partial}{\partial u} + \kappa\frac{\partial}{\partial K} + \kappa_{(u)}\frac{\partial}{\partial K_u} + \kappa_{(uu)}\frac{\partial}{\partial K_{uu}} \tag{4.33}$$

where $\kappa_{(uu)}$ depends on (u, K, K_u, K_{uu}), and $\kappa_{(u)}$ was found in (4.30). We can take advantage of $\kappa_{(u)red}$ to find out $\kappa_{(uu)}$

$$\begin{aligned}\kappa_{(uu)} &= D_u\kappa_{(u)} - K_{uu}D_u\eta \\ &= D_u\kappa_{(u)red} - K_{uu}D_u\eta \\ &= D_u\left(\frac{K_u}{K}\kappa - K_u\eta_u\right) - K_{uu}D_u\eta \\ &= -K_u\eta_{uu} + K_u\left(\frac{K_u}{K}\kappa_K - \frac{K_u}{K^2}\kappa\right) + K_{uu}\left(\frac{\kappa}{K} - \eta_u\right) - K_{uu}\eta_u\end{aligned}$$

2. Since we have already reduced $\kappa_{(u)}$, we only need to reduce $\kappa_{(uu)}$

mod $\mathcal{Q}$ (4.28)

$$\begin{aligned}\kappa_{(uu)red} &= \kappa_{(uu)} \mod \mathcal{Q} \\ &= \frac{K_{uu}}{K}\kappa - 2K_{uu}\eta_u \end{aligned} \tag{4.34}$$

The reduced vector field to order 2 becomes

$$\begin{aligned}\mathbf{G}^{(2)}_{red} =& \eta\frac{\partial}{\partial u} + \kappa\frac{\partial}{\partial K} \\ &+ \left(-K_u\eta_u + \frac{K_u}{K}\kappa\right)\frac{\partial}{\partial K_u} + \left(\frac{K_{uu}}{K}\kappa - 2K_{uu}\eta_u\right)\frac{\partial}{\partial K_{uu}}\end{aligned} \tag{4.35}$$

3. Check if $-4KK_{uu} + 7K_u^2 = 0$ is invariant by applying the symmetry condition again. We first find using (4.31, 4.34),

$$\begin{aligned}\mathbf{G}^{(2)}_{red}\left(-4KK_{uu} + 7K_u^2\right) =& -4K\kappa_{(uu)red} + 14K_u\kappa_{(u)red} - 4K_{uu}\kappa_{red} \\ =& -4K\left(\frac{K_{uu}}{K}\kappa - 2K_{uu}\eta_u\right) \\ &+ 14K_u\left(\frac{K_u}{K}\kappa - K_u\eta_u\right) - 4K_{uu}\kappa \\ =& 2\left(\frac{\kappa}{K} - \eta_u\right)\left(-4KK_{uu} + 7K_u^2\right)\end{aligned}$$

Then $\mathbf{G}^{(2)}_{red}\left(-4KK_{uu} + 7K_u^2\right) \mod \left(-4KK_{uu} + 7K_u^2\right)$ is 0. Therefore, $-4KK_{uu} + 7K_u^2 = 0$ is also tested as invariant under the action of the equivalence group.

The last remaining splitting condition $KK_uK_{uuu} - 2KK_{uu}^2 + K_u^2K_{uu} = 0$ can be tested using exactly the same steps as we have done for previous two splitting conditions. The vector field needs to be prolonged to order 3 this time. As a result, $KK_uK_{uuu} - 2KK_{uu}^2 + K_u^2K_{uu} = 0$ is also found to be invariant, so we can conclude the whole classification tree Fig. 3.3 is

invariant under the equivalence group.

4.4.2 Guide RIF during classification

During classification by RIF, the algorithm can reach a stage where it is forced to split into two cases. If there is more than one equation, then we may have a choice of which equation to split. For instance, recall back to §3.3.1 the Maple package `rifsimp` allows the user to specify preferences on which pivot to choose, such as choosing the smallest length pivot or the lowest rank pivot.

In each branching during classification, before a pivot is selected, there is a set of eligible pivot candidates. The idea is to use the theory from §4.3 to select the preferred 'invariant pivot' rather than just the preferred 'pivot'. ('Invariant' meaning with respect to the equivalence group.)

Therefore, when there is a branching happening during classification, assuming we have a set of pivot candidates, the steps are as follows:

1. Sort the pivot candidates according to order of preference from the most desirable to the least desirable ones. Denote the sorted pivot candidates list as $PivCand = [pc_1, \ldots, pc_n]$.

2. From the list $PivCand$, starting from pc_1, we check whether pc_i is invariant using the steps from the previous section. The first pc_i that tests to be invariant is the selected pivot.

3. If all pivot candidates are non-invariant then we choose the first pivot candidate pc_1 to be the selected pivot.

The 'order of preference' here could be for instance from smallest to largest (in size), or from lowest to highest (in ranking). The reason for Step 1 is

that sorting the pivot candidates is computationally cheap, compared with the 'checking invariance' step. By sorting, Step 2 allows exit as soon as an invariant pivot candidate is found: there is no need to test any remaining pivot candidates. Therefore, the whole process becomes much more efficient.

Example 23. Suppose that instead of the determining equations (2.19) for the nonlinear heat equation, RIF had the following (artificial) equations

$$\begin{aligned}(KK_{uu} - K_u^2)\tau_u &= 0\\ (KK_u - K^2)\eta_{xx} - (K_u - K)\eta_t + \tau_u &= 0\\ (KK_uK_{uuu} - 2KK_{uu} + K_u^2K_{uu})\tau_u &= 0\end{aligned}$$

and we are asked to choose one of them to split on. And suppose we still wish the splitting to be invariant with respect to the same equivalence group (4.28).

We specify a ranking for which

$$\{K\} \ll \{\tau, \xi, \eta\}$$

with ties broken by order of derivative. The leading derivatives of the equations are $\tau_u, \eta_{xx}, \tau_u$, respectively. Therefore, we have a set of pivot candidates:

$$\left\{KK_{uu} - K_u^2, \qquad KK_u - K^2, \qquad KK_uK_{uuu} - 2KK_{uu} + K_u^2K_{uu}\right\}$$

Using the steps above we first sort these pivot candidates in order of preference, which we take as being according to the length of pivots from

smallest to largest. We denote the sorted pivot candidates list as

$$PivCand = \left[KK_u - K^2, \qquad KK_{uu} - K_u^2, \qquad KK_uK_{uuu} - 2KK_{uu} + K_u^2K_{uu} \right]$$

The next job is test invariance (with respect to equivalence group (4.28)) using steps from previous section. The first candidate $KK_u - K^2$ is tested as non-invariant, therefore we move on to the second. Since the second candidate $KK_{uu} - K_u^2$ is tested as invariant, we select it as the pivot, that is it is the smallest invariant pivot candidate. There is no need to test the last candidate.

By using this method to guide RIF during classification, we can try to guide it away from bad (non-invariant) pivot choices. This should be able to lead to a simpler classification tree than the current version of RIF can achieve.

Chapter 5

Implementation of Invariance Checking Method

In the previous chapter, we developed a method for checking invariance of DEs under the action of some group. For symmetry classification problems, we also have established two applications (i.e. label pivots after classification and guide RIF during classification, see §4.4) where we can apply the method to help find a classification which respects the equivalence group. Because the whole process is purely algorithmic, it is desirable to implement the method in computer algebra, and this is our goal for this chapter.

Before we go any further, our first job is to choose an existing differential reduction & completion (DRC) package for our implementation. We chose the Maple package `rifsimp` which uses the RIF algorithm for the following reasons:

1. Maple is one of the most widely used computer algebra systems.

2. A useful symmetry package (as part of `PDEtools`) is available, from

Maple 11. This package includes various functionality such as finding determining equations for symmetries, prolonging vector fields, etc. We can utilise functions from this existing package as much as possible so we are able to focus on the main implementations, namely symmetry classification using `rifsimp` with the the invariance checking method (ICM).

3. `Rifsimp` is a robust and efficient DRC package in Maple. Many Maple commands (e.g. `DeterminingPDE`, `pdsolve`) use `rifsimp` as a service procedure. If the invariance checking method is implemented in `rifsimp` then it will immediately benefit many symmetry classification users.

5.1 Required Implementations

There are several required implementations to be done in order to achieve the two applications described in §4.4.

We first need to think about how we should manage/store information about the DE system. In our implementation, we often need to use information from the DE system and later add further information to it. There are two kinds of information of DE system involved in this implementation:

- *Initial information of* DE *system*, which is given by user. This includes DE system itself, list of independent/dependent variables, any constraints on the arbitrary elements,....
- *Updated information of* DE *system*, which is calculated as part of the symmetry classification process. This includes determining equa-

tions for symmetries and for the equivalence group.

Because the information on the DE system comes from various sources (i.e. given by user or resulting during classification), and because the methods described in §4.4 use all of this information, we need to make sure that *all* needed information for the DE system is bundled together. Therefore, our first two required tasks are:

(i) DEs *storage and management:*

Task 1 We need to design a storage structure to hold all information about the DE system needed for the package. We will refer to this structure as a `pdeRecord`.

Task 2 We need to provide a constructor function [46] which creates a `pdeRecord` from initial information on a given DE system.

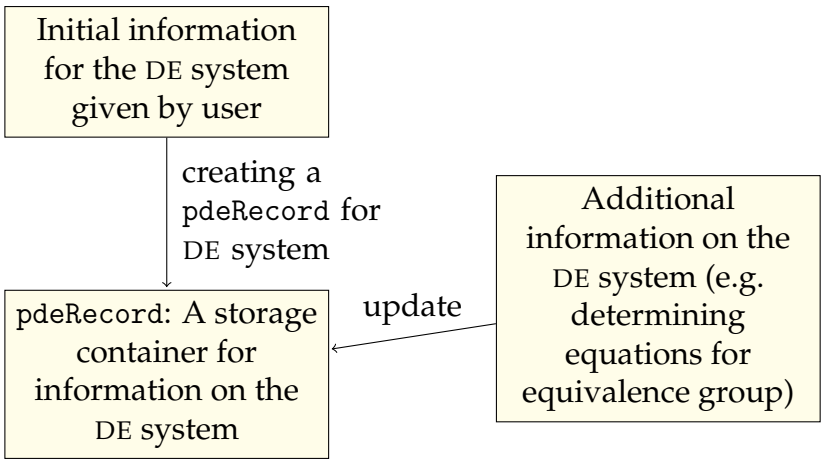


Figure 5.1: A `pdeRecord` is created by a call to a constructor function. Information in the `pdeRecord` gets updated by calling other methods. When a `pdeRecord` contains all information on the DEs, the `pdeRecord` is complete.

Next, recalling the two applications described in §4.4, the ideas for using `rifsimp` are as follows:

1. *Labelling invariant splitting conditions:* First, run normal `rifsimp` to produce a classification tree. We sweep through each pivot in the classification tree, using the ICM method to test if the pivot conditions are invariant, and label the pivot accordingly.

2. *Improving splitting condition selection:* We must guide `rifsimp` to use a more preferable pivot such as the smallest invariant pivot during case splittings. One way to do this is that we can implement a new 'pivot selection' option into `rifsimp` so the invariance checking method can be applied while performing classifications.

Therefore, the remaining required tasks are:

(ii) Pre-step – before classifying symmetries (i.e. before calling `rifsimp`):

Task 3 We need to find the determining equations for point symmetries so we can do classification.

Task 4 To test invariance under the action of the equivalence group, we need to derive determining equations for the equivalence group beforehand.

(iii) `rifsimp` with the invariance checking method:

Task 5 To implement the invariance checking method (see §4.3).

Task 6 Code for labelling invariant splitting conditions (see §4.4.1). This relies on Task 5.

Task 7 A modified version of `rifsimp` which has an additional option for choosing invariant splitting DEs during classification (see §4.4.2). This also relies on Task 5.

(iv) Display case tree structure:

Task 8 There is currently a display function `caseplot` (in the `DEtools` package) which draws case tree structure from given rif-output. However, we need to modify this function so it can show invariance information too.

5.2 Symmetry Classification Package

Now that all required tasks have been identified, we will describe how we developed a Maple package for classifying Lie point symmetries, called `SymmetryClassification`. This package serves two main purposes, first, it provides users with a front-end tool for doing symmetry classification, so the user does not need to understand how other packages (such as `rifsimp` and `PDEtools`) work. Second, it provides additional functionality for classifying symmetries while respecting the equivalence group. This functionality uses the method described in §4.4.

As shown in Figure 5.3, the symmetry classification package contains several methods (i.e. procedures), and these methods are clustered into four groups: constructor, pre-step methods, rifsimping, and displaying. In this section, we give a brief description of the procedures corresponding to the tasks as specified in §5.1. The gritty detail of each method in the symmetry classification package is attached as help pages under Appendix A. We will also illustrate how to use the package by applying it to some examples.

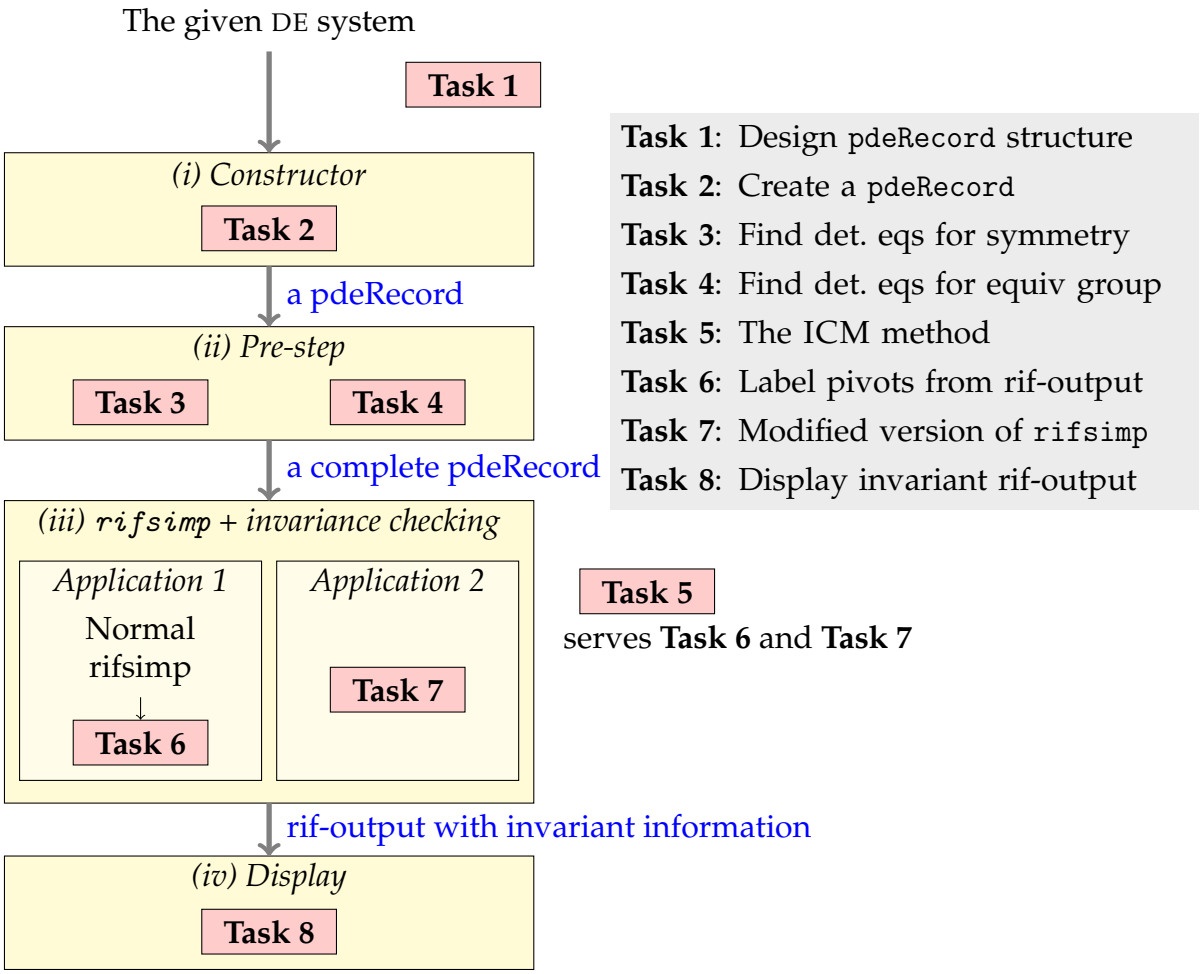


Figure 5.2: Overall structure of all required tasks. Our work is mostly on Tasks 1, 2, 3, 4, 6, 7.

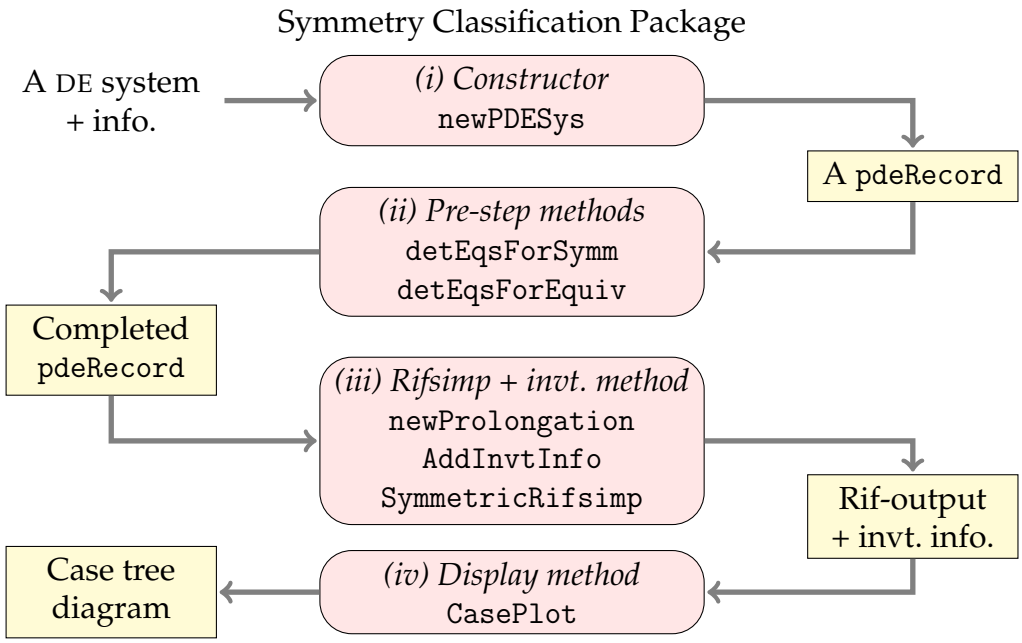


Figure 5.3: Sequence structure of the symmetry classification package where the initial information of DE system is given by the user. Methods from the package are shown in red rounded corner shape, and output information is shown in yellow rectangle shape.

5.2.1 Storage Structure for DE System

`pdeRecord` (Task 1)

A `pdeRecord` is a storage container which we designed for symmetry classification package to store all information on a DE system. This storage uses the `module` construct from Maple (see [66, help on `module`]). A `pdeRecord` contains the following main fields:

- `sys`: Basic information on the DE system which is given by a user or inferred by the constructor. This includes the DE system itself, its independent/dependent variables and arbitrary elements.

- `constraint`: Optional information on the DE system given by the user, it specifies any constraints on the arbitrary elements. This includes constraint DEs (and/or inequalities) and their independent/dependent variables.
- `detSymm`: Information on the determining equations for point symmetries of the DE system. It also includes other information such as independent/dependent variables of the determining equations.
- `detEquiv`: Information on the determining equations for equivalence group of the DE system. This field contains two different spaces on which the equivalence group may act: (1) on all variables (`fullAction`) and (2) on arbitrary elements only (`arbvarsAction`). Each action has its own determining equations and other information such as independent/dependent variables (see §4.1).

A call to the constructor (below) creates a `pdeRecord` with these four fields, but `detSymm` and `detEquiv` are not assigned yet. The information on `detSymm` and `detEquiv` will be filled in later by pre-step methods. Once a `pdeRecord` has all four fields (i.e. `sys`, `constraint`, `detSymm` and `detEquiv`) assigned, the `pdeRecord` is then complete and ready for classification using `rifsimp` + invariance checking.

`newPDESys` (Task 2)

This is the constructor [46] for a `pdeRecord` of the symmetry classification package. The purpose of `newPDESys` is to create a `pdeRecord` from a DE system given by a user. It also allows specification of constraints on the arbitrary elements, a feature which will later al-

low us to introduce differential invariants of the equivalence group to help with classification (e.g. §5.3.2). Another feature of this procedure is that it can extract independent/dependent variables and identify arbitrary elements from the DE system and record them in the `pdeRecord`. The variables can be also be specified by the user if desired; in this case, `newPDESys` checks the validity of the user's specification.

5.2.2 Pre-step methods

`detEqsForSymm` (Task 3)

The purpose of this procedure is to derive the determining equations for point symmetries. The procedure `detEqsForSymm` updates a `pdeRecord` which has been created by calling `newPDESys`. The procedure allows a user to specify 'infinitesimals', but will choose sensible defaults if not specified. `detEqsForSymm` meets its purpose by calling the existing procedure `DeterminingPDE` (from package `PDEtools`). After the determining equations for symmetries have been derived, the results are written in the `detSymm` field, and the `pdeRecord` is updated.

`detEqsForEquiv` (Task 4)

This procedure is to derive the determining equations for the equivalence group of a given DE system. This procedure is similar to `detEqsForSymm`, the procedure `detEqsForEquiv` updates a `pdeRecord` which has been created by calling `newPDESys`. Again, it allows a user to specify 'infinitesimals', but will choose sensible defaults if not specified.

This procedure derives determining equations for the equivalence group using the steps described in §2.3. Then it derives two types of determining equations on: all variables (`fullAction`) and arbitrary variables (`arbvarsAction`) spaces (see §4.1). The procedure first calls the existing procedure `DeterminingPDE` for each of DE system and constraint system, then combines these two systems and completes it by `rifsimp`.

After the determining equations for the equivalence group have been derived, the results are written in the `detEquiv` field, which updates the `pdeRecord`. The field `detEquiv` contains two sub-fields, namely `fullAction` and `arbvarsAction`.

5.2.3 `Rifsimp` with the ICM method

`newProlongation` (Task 5)

This is the code implementation of the ICM method as described in §4.3. As shown in Fig. 5.2, this procedure is outside of the main development of the implementation. The procedure `newProlongation` was written by Dr. Ian Lisle, and it is a service procedure for other procedures (i.e. `AddInvtInfo` and `SymmetricRifsimp` below).

`newProlongation` is a constructor function for creating methods for calculating a reduced prolonged vector field. It requires a vector field (e.g. vector field of equivalence group), a list of dependent variables, a list of independent variables, and the determining equations of some group. It returns a module containing methods for computing reduced prolonged vector fields and checking invariance. The main method in `newProlongation` is `checkInvariant`. The method

`checkInvariant` requires a DE system (i.e. pivot conditions) and it returns true if the system is invariant or false if not-invariant.

`AddInvtInfo` (Task 6)

The procedure `AddInvtInfo` is the implementation of application 1 as described in §4.4.1. The purpose of this procedure is to check invariance of pivot conditions and to tag additional information (i.e. invariant or non-invariant) on these pivots in a completed classification tree. This procedure requires a rif-output which has no invariant information in it, the sub-field `arbvarsAction` under `detEquiv` from the `pdeRecord` (see §5.2.1), and a list of arbitrary elements. It returns an updated rif-output which contains invariance information on all pivots. The returned rif-output includes all entries (e.g. `Pivots`, `Case`, `Solved`,...) from the original rif-output but it also includes two new entries for invariant information purpose:

- `InvtCase`: The format is similar to the entry `Case` but `InvtCase` has 'true' (as invariant) or 'false' (as non-invariant) appended.
- `InvtPivots`: Likewise, the format is similar to the entry `Pivots` but `InvtPivots` has 'true' (as invariant) or 'false' (as non-invariant) appended.

(The original entries `Case` and `Pivots` are untouched in the rif-output, so that other procedures that rely on these entries do not break.)

The procedure `AddInvtInfo` uses the the method `checkInvariant` (returned by the service procedure `newProlongation`) for each pivot condition. The steps of this procedure are as follows:

1. Collect all pivots from a given rif-output.

2. Call `newProlongation` to return a module which contains methods for the reduced prolonged vector field.
3. Use the ICM method (i.e. `checkInvariant`) to check invariance of all pivot conditions, and collect the invariant ones into a list.
4. Create two new entries : `InvtCase` & `InvtPivots` by copying `Case` & `Pivots` for each case in the classification tree.
5. Label all pivots in these two new entries for all cases, using the invariant pivot list as extracted in Step 3.
6. Append `InvtCase` & `InvtPivots` into the rif-output, and return the updated rif-output.

`SymmetricRifsimp` (Task 7)

The procedure `SymmetricRifsimp` is a modified version of `rifsimp`. The aim of this procedure is to guide `rifsimp` to select 'invariant' pivots during classification (see §4.4.2). All functionality of `rifsimp` can be applied to `SymmetricRifsimp`, but `SymmetricRifsimp` provides an additional option in `pivselect` which is to select 'invariant' pivots if wanted.

To call `SymmetricRifsimp` without providing the pivot select option (which is just calling `rifsimp` with default `pivselect` option), the procedure requires at least: determining equations for symmetries (e.g. field `sys` under `detSymm` in `pdeRecord`), a valid ranking (see §3.3), and `casesplit`.

To select invariant pivots during classification, we need to specify the `pivselect` option as `invariant` enabled, plus determining equations for the equivalence group in arbitrary elements action (e.g. field `arbvarsAction` under `detEquiv` in `pdeRecord`), and a list of arbitrary

elements. The format for this `pivselect` option looks like this

```
pivselect=['invariant', detEquivInArbvarsAction, arbVarsList];
```

Note that the order of preference for invariant pivots is assumed to be `smalleq` (smallest length equations).

The procedure returns a rif-output. If invariant pivots are requested then the rif-output has two new entries `InvtCase` and `InvtPivots` (which is the same as the output returned by calling `AddInvtInfo`).

5.2.4 Display Procedure

`CasePlot` (Task 8)

This procedure `CasePlot` is a re-written version of the original procedure `caseplot` [66, help on `DEtools`]. The idea for both procedures (i.e. `caseplot` and `CasePlot`) is to display rif-output into a graphical classification tree so it is easier to view. The procedure `CasePlot` was originally developed for a previous project, and this code was written by Dr. Ian Lisle. For this project, we extended the procedure so that it can display not just normal rif-output but also display rif-output with invariance information attached to it.

The minimum requirement for the `CasePlot` is a rif-output, it returns a graphical classification tree.

5.2.5 Front-end procedure

`classifySymmetry`

This is the front-end procedure of the `SymmetryClassification` pack-

age. The purpose of this procedure is to provide an easy way to perform symmetry classification by calling just this procedure, so that a user doesn't need to call all the procedures described above to complete a classification.

This procedure requires a `pdeRecord` which is created by calling the constructor (`newPDESys`). This `pdeRecord` does not have to be complete. It returns a rif-output. The returned rif-output can then be displayed by calling `CasePlot`.

This procedure also has other options such as specifying infinitesimals, rankings for both infinitesimals and arbitrary elements, and pivot select option. It first checks if the procedures `detEqsForSymm` and `detEqsForEquiv` need to run, then it runs `SymmetricRifsimp` according to the user ranking and pivot select option (use default ranking if none are specified).

5.3 Examples

We will demonstrate the use of the `SymmetryClassification` package by working through some examples.

5.3.1 1+1 Richards Equation

Consider the example of 1+1 Richards equation in potential form

$$v_x = u, \qquad v_t = B(u)u_x - K(u) \tag{5.1}$$

where u, v are functions of (x, t), while $B(u), K(u)$ are arbitrary elements with constraint $B(u) \neq 0$. The associated vector field is

$$\mathbb{Y} = \xi \frac{\partial}{\partial x} + \tau \frac{\partial}{\partial t} + \eta \frac{\partial}{\partial u} + \phi \frac{\partial}{\partial v} + \beta \frac{\partial}{\partial B} + \kappa \frac{\partial}{\partial K} \tag{5.2}$$

where ξ, τ, η, ϕ depend on (x, t, u, v) and β, κ depend on (x, t, u, v, B, K).

In this example, we will examine how to use the ICM method to classify symmetries by using procedures in the `SymmetryClassification` package, based on the two applications (i.e. labelling pivots & guiding `rifsimp`) described in the previous section.

Let `DE` be assigned to be the Richards equation (5.1),

```
DE:=[diff(v(x,t),x) = u(x,t),
     diff(v(x,t),t) = B(u)*diff(u(x,t),x) - K(u)];
```

with constraint $B(u) \neq 0$,

```
sysConstraint:= [B(u)<>0];
```

Recalling Figure 5.3, before we call `SymmetricRifsimp` to classify symmetries, we need to create a `pdeRecord` and have all fields assigned.

(i) Constructor – Create a `pdeRecord`: To create a `pdeRecord` we call the constructor `newPDESys`:

```
rec:= newPDESys(DE,constraint = sysConstraint);
```

This returns a new `pdeRecord` named `rec` in this example. This `pdeRecord` has information in these two fields: `sys` and `constraint`.

(ii) Pre-step procedures – Complete the `pdeRecord`: Next, we need to derive the determining equations for both symmetries and equivalence group, and complete the `pdeRecord`, meaning that the information for the

two fields `detSymm` & `detEquiv` are assigned in `rec` (see §5.2.2). The vector field (5.2) is specified by

```
vf:=[[x,xi], [t,tau], [u,eta], [v,phi], [B,beta], [K,kappa]];
```

We call `detEqsForSymm` to derive the determining equations for symmetries:

```
detEqsForSymm(rec, infinitesimals = vf);
```

This fills in information in `detSymm` field in `rec`.

We also derive the determining equations for the equivalence group (for both actions) by calling `detEqsForEquiv`:

```
detEqsForEquiv(rec, infinitesimals = vf);
```

This fills in information in the field `detEquiv` in `rec`.

After the two fields `detSymm` and `detEquiv` are assigned in `rec`, the `pdeRecord` is complete. So, now it is time to classify symmetries.

Application 1 (iii,iv) – Run `SymmetricRifsimp` without 'invariant'; label pivots, and display: In this approach, we illustrate the application 1 which is to label pivots from a classification tree. So, first we need to call `SymmetricRifsimp` without 'invariant' pivot selection enabled (which is the same as calling `rifsimp`). `SymmetricRifsimp` requires a list of DEs (e.g. the determining equations for symmetries and the constraint) and the ranking. Let `detSys` be

```
detSys:=[op(rec:-detSymm:-sys), op(rec:-constraint:-sys)];
```

where `rec:-detSymm:-sys` means the list of DEs `sys` under field `detSymm` which is stored in `rec`, and `rec:-constraint:-sys` is the constraint system (i.e. `[B(u)<>0]`).

Let the ranking be $\{B\} \ll \{K\} \ll \{\phi, \xi, \tau, \eta\}$, or in Maple syntax

```
ranking:=[[eta, tau, xi, phi],[K],[B]];
```

We call `SymmetricRifsimp` by

```
rifSys:=SymmetricRifsimp(detSys, ranking, casesplit);
```

It returns a rif-output `rifSys` and it shows there are 17 cases.

We would now like to check invariance under the equivalence group of all pivots in the rif-output `rifSys` and label them. To do this we call `AddInvtInfo`: this procedure requires the rif-output `rifSys`, the sub-field `arbvarsAction` under `detEquiv` and a list of arbitrary elements (as found in `rec:-detSymm:-arbvars`):

```
rifSys:=AddInvtInfo(rifSys,rec:-detEquiv:-arbvarsAction,
       rec:-sys:-arbvars);
```

It returns an updated rif-output `rifSys`. Last, we call `CasePlot` to display the rif-output graphically (see §5.2.4). And to ask `CasePlot` to display dimension of the symmetry group in each case, we provide a list of infinitesimals (excluding infinitesimals from the arbitrary elements),

```
CasePlot(rifSys, vars=[xi, tau, phi, eta]);
```

According to Figure 5.4, one non-invariant splitting condition $(p_4 = 0)$ is picked up during early splits. For p_4, the left branch and the right branch are very similar (i.e. they both have cases which have 3, 4, and 5 symmetries), it is very possible that both branches are in fact connected by an equivalence transformation. Therefore, p_4 is a bad splitting and gives more redundant splittings (e.g. p_6). This is because normal `rifsimp` does not take account of the equivalence group.

Application 2 (iii,iv) – Run `SymmetricRifsimp` with 'invariant', and display: Our next approach is to guide `rifsimp` to classify symmetries which

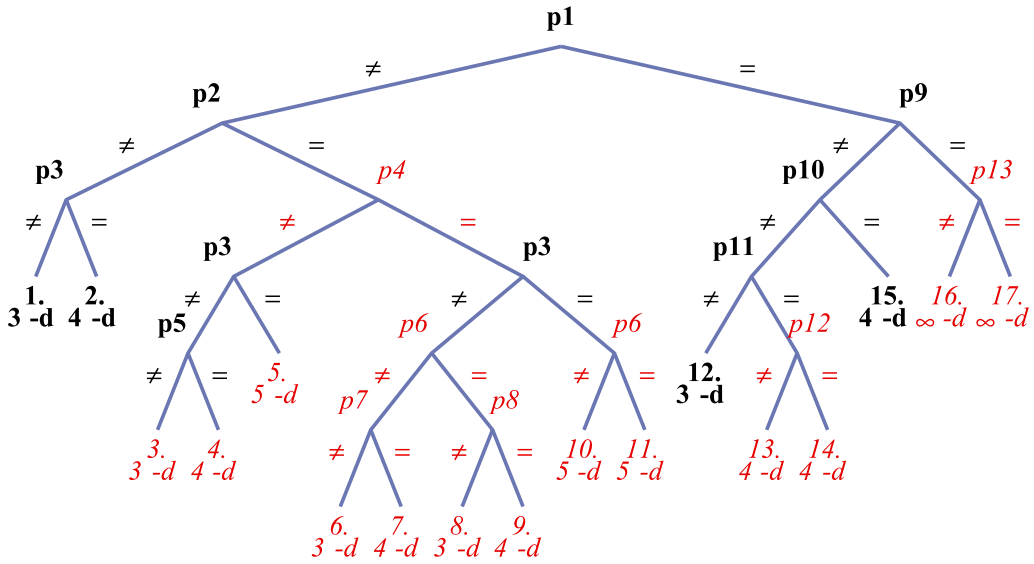


Figure 5.4: Classification tree for the 1+1 Richards equation, completed by calling `SymmetricRifsimp` with ranking $\{B\} \ll \{K\} \ll \{\xi, \tau, \eta, \phi\}$ and default pivot selection strategy. The case splitting conditions are $p_1, \ldots, p_{14}$. Invariant splitting conditions are displayed in bold, others in italic.

respect the equivalence group, by applying the ICM method. All we need to do is to set the 'invariant' pivot selection option for `symmetricRifsimp`,

```
rifInvtSys:=SymmetricRifsimp(detSys,ranking,casesplit,
   pivselect=['invariant',rec:-detEquiv:-arbvarsAction,
             rec:-sys:-arbvars]);
```

The rif-output `rifInvtSys` now contains invariance information on pivots, so we can just call `CasePlot` again to display the classification tree,

```
CasePlot(rifInvtSys, vars=[xi, tau, phi, eta]);
```

The new classification tree (see Figure 5.5) now has only 14 cases.

By using the ICM method, the new classification tree has avoided the bad splitting which we had by ordinary `rifsimp` (p_4 in Figure 5.4). Although there are still some redundant case splittings (e.g. p_6, p_{13} which

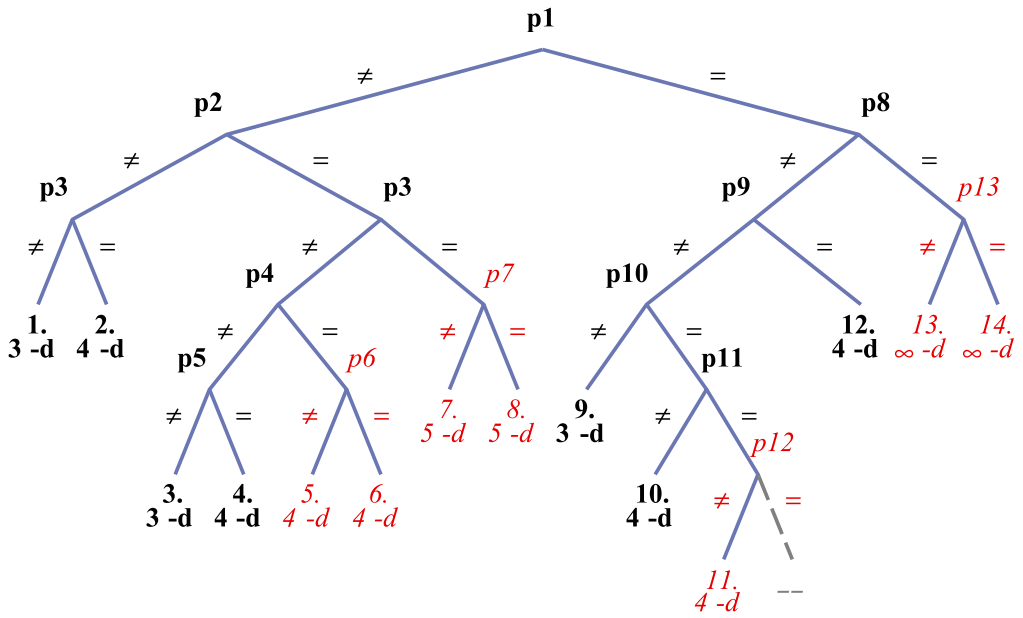


Figure 5.5: Classification tree for 1+1 Richards equation, completed by calling `SymmetricRifsimp` with pivot select option 'invariant'. Invariant splitting conditions are displayed in bold, others in italic. The numbering of the splitting conditions is not identical to Fig. 5.4.

are marked as non-invariant), these splittings are pushed down towards the leaves of the classification tree, so the classification tree is as invariant as possible. As a result, the number of cases in the classification tree has dropped by 3.

We now make some comment about performance. On an ordinary laptop, plain `rifsimp` took 8.1 sec to classify symmetries of the 1+1 Richards equation; but `AddInvtInfo` only needed 0.4 sec to add invariance properties to this classification (less than 5% time penalty). In fact, `AddInvtInfo` can check invariance of *all* 13 pivots in less time than `DeterminingPDE` takes (0.7 sec) to find determining equations of *one* pivot condition. With invariant pivot selection enabled, `SymmetricRifsimp` took 9.1 sec to complete classification. So the improved tree of Figure 5.5 is found with only

a 12% time penalty. This shows that the ICM method in this case is quite efficient.

5.3.2 Linear Hyperbolic Equation with Laplace Invariants

Consider the linear hyperbolic equation [48, §9],

$$z_{xy} + A(x,y)z_x + B(x,y)z_y + C(x,y)z = 0 \tag{5.3}$$

where z depends on (x,y) and $A(x,y), B(x,y), C(x,y)$ are arbitrary elements. Let the corresponding vector field be

$$\mathbb{Y} = \xi\frac{\partial}{\partial x} + \phi\frac{\partial}{\partial y} + \zeta\frac{\partial}{\partial z} + \alpha\frac{\partial}{\partial A} + \beta\frac{\partial}{\partial B} + \chi\frac{\partial}{\partial C}$$

where ξ, ϕ, ζ depend on (x,y,z) and α, β, χ on (x,y,z,A,B,C).

If we compare with the previous example in §5.3.1, this DEs system (5.3) is more complicated because it involves three arbitrary elements. After deriving the determining equations for point symmetries for the system (5.3), unfortunately `rifsimp` fails to classify symmetries due to time/memory usage. This is due to the complexity of the splitting conditions, it may also be because `rifsimp` chooses bad splitting conditions, causing more unnecessary splits.

One way we could help `rifsimp` to classify symmetries is to use invariants from the equivalence group of the system (5.3),

$$x' = f(x) \tag{5.4a}$$

$$y' = g(y) \tag{5.4b}$$

$$z' = w(x,y)z \tag{5.4c}$$

where $f(x)$, $g(y)$, and $w(x,y)$ are arbitrary functions. The Laplace invariants [48, §9.2]

$$h(x,y) = A_x + AB - C, \qquad k(x,y) = B_y + AB - C \tag{5.5}$$

are derived from equation (5.4c) of the equivalence group. This pair of Laplace invariants can give a big hint to `rifsimp` so it can make better decisions in choosing splitting conditions.

In the `SymmetryClassification` package, we allow to have additional information for the system. The equations (5.5) for Laplace invariants can be stored into the `pdeRecord` as a constraint system, with the variables h, k being treated as arbitrary elements.

So, let the system `sys` (5.3) be

```
sys:=[diff(z(x,y),x,y) + A(x,y)*diff(z(x,y),x)
      + B(x,y)*diff(z(x,y),y) + C(x,y)*z(x,y) = 0];
```

with the Laplace equations (5.5)

```
laplaceEqs:=[h(x,y) = diff(A(x,y),x)+A(x,y)*B(x,y)-C(x,y),
             k(x,y) = diff(B(x,y),y)+A(x,y)*B(x,y)-C(x,y)];
```

We first create a `pdeRecord` for the system (5.3),

```
rec := newPDESys(sys, constraint=laplaceEqs);
```

After assigning `detSymm` and `detEquiv` fields in the `pdeRecord` by calling `detEqsForSymm` and `detEqsForEquiv`, we call `SymmetricRifsimp` to classify symmetries.

By having the Laplace invariants in the DEs system (5.3), we would like to ask `SymmetricRifsimp` to split on h, k (the Laplace invariants) as much as possible. Therefore, we need to make sure h, k (and A, B, C) are ranked lower than other variables (e.g. $\{h, k, A, B, C\} \ll \{\zeta, \xi, \phi\}$). Therefore,

```
ranking := [[zeta, xi, phi],[h, k, A, B, C]];
```

We now run SymmetricRifsimp with this ranking, then label all pivots from the rif-output, the result is shown in Figure 5.6,

With the help of Laplace invariants (5.5), SymmetricRifsimp is able to classify symmetries with total of 21 cases. All the splitting conditions are in terms of h, k, and they are all invariant under the equivalence group (5.4).

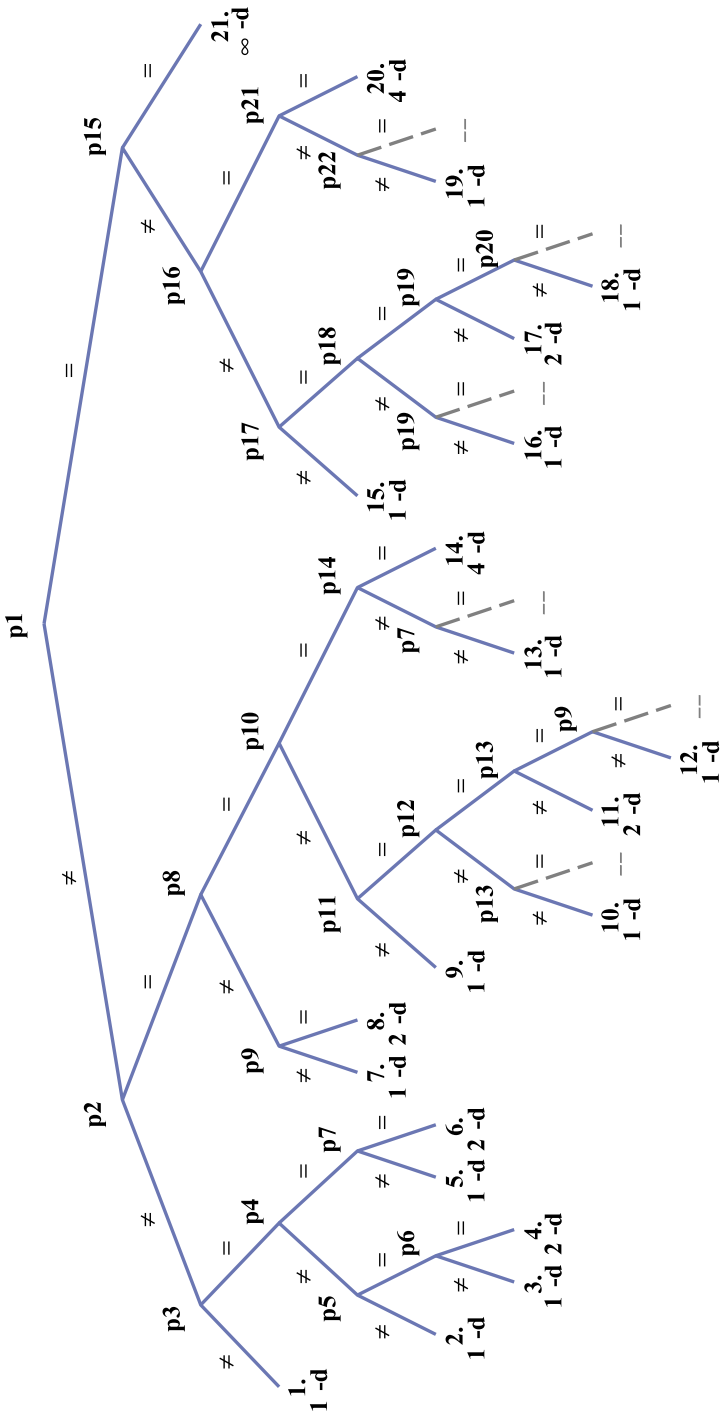


Figure 5.6: Symmetry classification for linear hyperbolic equation. Introducing the Laplace invariants helps `SymmetricRisimp` to be able to complete symmetry classification. Some splitting conditions are still complicated. For example, the longest is $p_{17} = 0$ (approx 100 terms). All splitting conditions are tested to be invariant under action of equivalence group.

Chapter 6

Conclusion

In conclusion, we have successfully reached our goal of helping a differential reduction and completion (DRC) method like RIF to improve symmetry classification by making the classification tree to be invariant under the equivalence group as much as possible. The whole method works at the level of determining equations.

To make this happen there were a number of tasks needed to be done first. We developed an invariance checking method (ICM) for testing invariance of given DEs under a group specified by determining equations. (In our case it is the equivalence group, see §4.3). This in turn requires reformulating the symmetry condition as in §4.2. The ICM method also requires the equivalence group in arbitrary elements space, which means we had to work out how to project the equivalence group determining equations as described in §4.1. With these tasks completed, we were able to apply the ICM method in RIF to check case splitting DEs for invariance (see §4.4).

A Maple package `SymmetryClassification` was developed for classifying point symmetries using `rifsimp`. To make this happen, a number of

required tasks as listed in §5.1 had to be done. First, we had to write code to derive the equivalence group determining equations (see §2.3), and to project the determining equations to the action on arbitrary elements. Second, we implemented the ICM method as a service procedure. Finally, we modified `rifsimp` so it uses the ICM method to select 'invariant' case splitting DEs during classification.

When implementing the `SymmetryClassification` package in Maple, we took care to base its design on good software design principles [46]. The code is well structured, and is easy to extend and adapt to other programs. We use the Maple `module` construct to carefully control the scope of variables and to make sure there are no global side effects. All information on a DE system is structured together as a `module`, so that this information can be passed from one procedure to another. The code implementation of the ICM method `newProlongation` is independent of the package. It would therefore be easy to adapt to other DRC Maple packages such as `diffalg` (which uses Rosenfeld-Gröbner algorithm). This gives an advantage for the ICM method to be integrated as part of future release of Maple.

We illustrated the `SymmetryClassification` Maple package by working through examples in §5.3. The example of §5.3.1 showed the classification improves when invariant case splitting DEs are selected. Also for another example, we are able to use differential invariants as constraints to the DE system to ensure `rifsimp` succeeded completed classification. The examples also showed the package is efficient when the invariance option is enabled, adding only a few seconds to the execution time of normal classification.

A good feature of the `SymmetryClassification` package is that users

only need to provide very little information – just a DE system. In particular, the equivalence group doesn't have to be given. The package is able to algorithmically work out the rest of needed information such as determining equations for symmetries and equivalence group, valid ranking etc. by itself. The reason the package can do this is that it is working on the level of determining equations, there is no need to find explicit solutions of PDEs. The steps of invariance checking (during or after symmetry classification) only involves differentiation and reduction, which is purely algorithmic. That is why the method is well suited to computer algebra.

However, there are some issues of using a DRC method in symmetry classification which still remain. When a DE system is complicated, the DRC method `rifsimp` often causes a problem of expression swell during classifying symmetries. Expression swell is a common problem in symmetry classification. Also based on the experience with using `rifsimp`, the choice of ranking used for reducing of the determining equations has a very strong influence on the outcome, and this is still true when our invariance checking is used.

There is an additional issue that comes up in invariance checking. If a splitting condition is an algebraic equation (no derivatives) then the infinitesimal methods of symmetry analysis do not work reliably.

There are geometric methods [38, 16, 17] which can be applied to symmetry classification problem, in a way that is invariant under the equivalence group. This would overcome the problem of expression swell. There exists DRC algorithms which can do geometric invariant calculation, by using non-commuting differential operators [26, 43]. However these geometric methods start with an explicit parameterisation of the equivalence group. Using our method, we don't need such a parameterisation, which

is a significant advantage of our method.

The ICM method could also be applied to other DRC method such as Rosenfeld-Gröbner rather than RIF algorithm. Also,the method can test invariance of DEs under action of some other groups (no need to be equivalence group), which gives various way to use the method in some fields other than symmetry classification.

Further Development Due to the limited time on the project, the package `SymmetryClassification` is subject to restrictions that could be removed with some more work. First, the package only allows the user to have '`smalleq`' as the invariant pivot selection strategy. With some further work on the package, it should be able to provide other invariant pivot selection strategies such as `smallpiv` or `lowrank` [66, help on `rifsimp`, `cases`].

Secondly, the form of arbitrary elements is restricted – the elements are only allowed to depend on independent and dependent variables (not derivatives); and derivatives of the arbitrary elements may not appear. Although introducing appropriate `constraint` equations can get around this restriction, it would be desirable to remove the restriction completely.

Thirdly, in the `SymmetryClassification` package, case splitting DEs in symmetry classification are assumed to generate a radical ideal. This may not be true in rare cases. Therefore, further work to the package is to provide a conversion method to ensure these case splitting DEs generate a radical ideal.

The methods described in this thesis should have wider application. The idea of finding conservation laws is similar to the problem of finding symmetries [47]. In particular, the 'multipliers' for the conservation laws

obey linear homogeneous determining equations [2]. Also, where there is a class of DEs containing arbitrary elements, we have a classification problem for conservation laws which has a good analogy to the classification problem for symmetries. In particular, the equivalence group plays similar roles as applied to symmetries [6]. Therefore, we would expect the ICM method should be able to apply to conservation laws, and it should behave in a similar way.

References

[1] W. AMES, ed., *Proceedings 14th IMACS World Congress on Computational and Applied Mathematics, Atlanta, Georgia*, vol. 1, New Brunswick, New Jersey, 1994, IMACS.

[2] S. ANCO AND G. BLUMAN, *Direct construction method for conservation laws of partial differential equations Part II: General treatment*, Euro. J. Appl. Maths., 13 (2002), pp. 567–585.

[3] S. C. ANCO AND T. WOLF, *Some symmetry classifications of hyperbolic vector evolution equations*, J. Nonlinear Math. Phys., 12 (2005), pp. 13–31.

[4] P. BASARAB-HORWATH, V. LAHNO, AND R. ZHDANOV, *The structure of Lie algebras and the classification problem for partial differential equations*, Acta Appl. Math., 69 (2001), pp. 43–94.

[5] G. BLUMAN AND S. KUMEI, *Symmetries and differential equations*, Springer-Verlag, New York, 1989.

[6] G. BLUMAN AND TEMUERCHAOLU, *Conservation laws for nonlinear telegraph equations*, J. Math. Anal. Appl., 310 (2005), pp. 459–476.

[7] F. BOULIER, D. LAZARD, F. OLLIVIER, AND M. PETITOT, *Representation for the radical of a finitely generated differential ideal*, in Proc. ISSAC '95, New York, 1995, ACM Press.

[8] V. V. BUBLIK, *Group classification of the Navier-Stokes equations for compressible viscous heat-conducting gas*, in Computer algebra in scientific computing (Samarkand, 2000), Springer, Berlin, 2000, pp. 61–67.

[9] B. CANTWELL, *Introduction to symmetry analysis*, Cambridge Univ. Press, Cambridge, 2002.

[10] J. CARMINATI, J. S. DEVITT, AND G. J. FEE, *Isogroups of differential equations using algebraic computing*, J. Symb. Comput., 14 (1992), pp. 103–120.

[11] G. CARRÀ-FERRO, *Differential Gröbner bases in one variable and in the partial case*, Math. Comput. Model., 25 (1997), pp. 1–10.

[12] B. CHAMPAGNE, W. HEREMAN, AND P. WINTERNITZ, *The computer calculation of Lie point symmetries of large systems of differential equations*, Comp. Phys. Comm., 66 (1991), pp. 319–340.

[13] A. F. CHEVIAKOV, *GeM software package for computation of symmetries and conservation laws of differential equations*, Comp. Phys. Comm., 176 (2007), pp. 48–61.

[14] G. CICOGNA, *Symmetry classification of quasi-linear PDE's containing arbitrary functions*, Nonlinear Dynam., 51 (2007), pp. 309–316.

[15] D. COX, J. LITTLE, AND D. O'SHEA, *Ideals, Varieties, and Algorithms: An introduction to computational algebraic geometry and commutative algebra*, Springer Verlag, New York, 1992.

[16] M. FELS AND P. OLVER, *Moving coframes: I A practical algorithm*, Acta. Appl. Math., 51 (1998), pp. 161–213.

[17] ——, *Moving coframes: II. Regularization and theoretical foundations*, Acta. Appl. Math., 55 (1999), pp. 127–208.

[18] L. GAGNON AND P. WINTERNITZ, *Non-Painlevé reductions of nonlinear Schrödinger equations*, Physical Review. A, General physics, 42 (1990), pp. 5029 – 5030.

[19] M. L. GANDARIAS AND N. H. IBRAGIMOV, *Equivalence group of a fourth-order evolution equation unifying various non-linear models*, Commun. Nonlinear. Sci. Numer. Simulat., 13 (2008), pp. 259–268.

[20] R. GARDNER, *The method of equivalence and its applications*, vol. CBMS-NSF 58, SIAM, Philadelphia, PA, 1989.

[21] F. GUNGOR, V. I. LAHNO, AND R. Z. ZHDANOV, *Symmetry classification of KdV-type nonlinear evolution equations*, J. Math. Phys., 45 (2004), pp. 2280 – 2313.

[22] A. HEAD, *LIE: A PC program for Lie analysis of differential equations*, Comp. Phys. Comm., 77 (1993), pp. 241–248.

[23] W. HEREMAN, *Review of symbolic software for the computation of Lie symmetries of differential equations*, Euromath Bulletin, 1 (1994), pp. 45–79.

[24] W. HEREMAN, *Symbolic software for Lie symmetry analysis*, in Ibragimov [29], ch. 13.

[25] D.-J. HUANG AND N. M. IVANOVA, *Group analysis and exact solutions of a class of variable coefficient nonlinear telegraph equations*, J. Math. Phys., 48 (2007), p. 073507.

[26] E. HUBERT, *Differential algebra for derivations with nontrivial commutation rules*, J. Pure Appl. Alg., 200 (2005), pp. 163–190.

[27] P. HYDON, *Symmetry methods for differential equations: A beginner's guide*, Cambridge Univ. Press, Cambridge, 2000.

[28] N. IBRAGIMOV, ed., *CRC handbook of Lie group analysis of differential equations*, vol. 1: Symmetries, exact solutions and conservation laws, CRC Press, Boca Raton, 1994.

[29] ——, ed., *CRC handbook of Lie group analysis of differential equations*, vol. 3: New Trends in Theoretical Developments and Computational Methods, CRC Press, Boca Raton, 1995.

[30] ——, ed., *CRC handbook of Lie group analysis of differential equations*, vol. 2: Applications in Engineering and Physical Sciences, CRC Press, Boca Raton, 1995.

[31] N. H. IBRAGIMOV, S. V. MELESHKO, AND E. THAILERT, *Invariants of linear parabolic differential equations*, Commun. Nonlinear. Sci. Numer. Simulat., 13 (2008), pp. 277–284.

[32] N. H. IBRAGIMOV AND M. TORRISI, *Equivalence groups for balance equations*, J. Math. Anal. Appl., 184 (1994), pp. 441–452.

[33] T. A. IVEY AND J. LANDSBERG, *Cartan for beginners: differential geometry via moving frames and exterior differential systems*, American Mathematical Society, 2003.

[34] M. KURANISHI, *On E. Cartan's prolongation theorem of exterior differential systems*, Amer. J. Math., 79 (1957), pp. 1–47.

[35] V. LAHNO, R. ZHDANOV, AND O. MAGDA, *Group classification and exact solutions of nonlinear wave equations*, Acta Appl. Math., 91 (2006), pp. 253–313.

[36] S. LIE, *Über die Integration durch bestimmte Integrale von einer Klasse linearer partieller Differentialgleichungen*, Arch. für Math., VI (1881), pp. 328–368.

[37] I. LISLE AND J.-Y. PARLANGE, *Analytical reduction for a concentration dependent diffusion problem*, Zeitschrift für Angewandte Mathematik und Physik, 44 (1993), pp. 85–102.

[38] I. LISLE AND G. REID, *Symmetry classification using noncommutative invariant differential operators*, Found. Comp. Math., 6 (2006), pp. 353–386.

[39] F. M. MAHOMED, *Symmetry group classification of ordinary differential equations: survey of some results*, Math. Methods Appl. Sci., 30 (2007), pp. 1995–2012.

[40] E. MANSFIELD, *Differential Gröbner bases*, PhD thesis, University of Sydney, 1991.

[41] E. MANSFIELD, *DIFFGROB2: A symbolic algebra package for analyzing systems of PDE using Maple. User's manual for release 2.*, tech. report, Department of Mathematics, University of Exeter, Exeter, UK, 1993.

[42] E. MANSFIELD AND P. CLARKSON, *Applications of the differential algebra package DIFFGROB2 to reductions of PDE*, in Ames [1], pp. 336–339.

[43] E. L. MANSFIELD, *Algorithms for symmetric differential systems*, Found. Comp. Math., 1 (2001), pp. 335–383.

[44] E. L. MANSFIELD AND P. A. CLARKSON, *Applications of the differential algebra package* `diffgrob2` *to classical symmetries of differential equations*, J. Symb. Comput., 23 (1997), pp. 517–533.

[45] S. MELECHKO, *Generalization of the equivalence transformations*, J. Nonlinear Math. Phys., 3 (1996), pp. 170–174.

[46] B. MEYER, *Object-oriented software construction*, Prentice Hall, 2nd ed., 1997.

[47] P. OLVER, *Application of Lie groups to differential equations*, Springer-Verlag, New York, 2nd ed., 1993.

[48] L. V. OVSIANNIKOV, *Group analysis of differential equations*, Academic Press, New York, 1982.

[49] J.-F. POMMARET, ed., *Proceedings ERCIM Advanced Course on Partial Differential Equations and Group Theory*, Sankt Augustin, Germany, 1992, Gesellschaft für Mathematik and Datenverarbeitung.

[50] G. REID, *Algorithmic determination of Lie symmetry algebras of differential equations*, in Lie Theory, Differential Equations and Representation Theory, V. Hussin, ed., Montréal, Canada, 1990, Proc. Annual Seminar of the Canadian Math. Soc., Publications de Centre de Recherches Mathématiques, pp. 363–372.

[51] ——, *A triangularization algorithm which determines the Lie symmetry algebra of any system of PDEs*, J. Phys., A23 (1990), pp. L853–859.

[52] ——, *Algorithms for reducing a system of PDEs to standard form, determining the dimension of its solution space and calculating its Taylor series solution*, Euro. J. Appl. Maths., 2 (1991), pp. 293–318.

[53] G. REID, I. LISLE, A. BOULTON, AND A. WITTKOPF, *Algorithmic determination of commutation relations for Lie symmetry algebras of PDEs,*, in Proc. ISSAC '92, New York, 1992, ACM Press, pp. 63–68.

[54] G. REID, A. WITTKOPF, AND A. BOULTON, *Reduction of systems of nonlinear partial differential equations to simplified involutive forms*, Euro. J. Appl. Maths., 7 (1996), pp. 604–635.

[55] R. ROGERS AND M. RENARDY, *An introduction to partial differential equations*, Springer, 1993.

[56] C. RUST, *Rankings of derivatives for elimination algorithms and formal solvability of analytic partial differential equations*, PhD thesis, University of Chicago, 1998.

[57] F. SCHWARZ, *The Riquier-Janet theory and its application to nonlinear evolution equations*, Physica, D11 (1984), pp. 243–251.

[58] ——, *Automatically determining symmetries of differential equations*, Computing, 34 (1985), pp. 91–106.

[59] ——, *An algorithm for determining the size of symmetry groups*, Computing, 49 (1992), pp. 95–115.

[60] ——, *Reduction and completion algorithms for partial differential equations*, in Proc ISSAC '92, New York, 1992, ACM Press, pp. 49–56.

[61] ——, *Algorithmic Lie theory for solving ordinary differential equations*, Chapman and Hall/CRC, Boca Raton, FL, 2008.

[62] L. SHAMPINE, *Some singular concentration dependent diffusion problems*, Zeitschrift für angewandte Mathematik und Mechanik, 53 (1973), pp. 421–422.

[63] J. SHERRING, A. K. HEAD, AND G. E. PRINCE, *Dimsym and LIE: symmetry determination packages*, Math. Comput. Model., 25 (1997), pp. 153–164. Algorithms and software for symbolic analysis of nonlinear systems.

[64] J. SHERRING AND G. PRINCE, *DIMSYM – symmetry determination and linear partial differential equations package*, Dept. of Mathematics Preprint, La Trobe University, Australia, 1992.

[65] P. VAFEADES, *PDELIE: Symbolic software for the analysis of partial differential equations by Lie group methods*, User's manual. Preprint, Department of Engineering Sciences, Trinity University, San Antonio, Texas, 1994.

[66] WATERLOO MAPLE SOFTWARE, 2009. Maple 13. Software package.

[67] A. WITTKOPF, *Algorithms and implementations for differential elimination*, PhD thesis, Department of Mathematics, Simon Fraser University, Canada, 2004.

[68] T. WOLF AND A. BRAND, *The computer algebra package CRACK for investigating PDEs*, in Pommaret [49], pp. 1–19.

[69] I. YAGLOM, *Felix Klein and Sophus Lie*, Birkhäuser, Boston, 1988.

[70] D. ZWILLINGER, *Handbook of differential equations*, Academic Press, 2nd ed., 1992.

Appendix A

The `SymmetryClassification` Package

The `SymmetryClassification` package as described in §5 was implemented in Maple, and consists of approximately 3000 lines of code. The help pages of this package are included in the following pages.

Like most software projects, the `SymmetryClassification` package is built based on prior projects. In the Maple source code, the function `SymmetricRifsimp` is a modified version of `rifsimp`, originally written by Dr. Allan Wittkopf and Dr. Greg Reid, and modified with their permission. The functions `newProlongation` and `CasePlot` and their help pages are written by Dr. Ian Lisle. `CasePlot` was adapted from a project done in previous degree. One of the local service procedure `dsubs` was included from Maple version 9.5, it was to circumvent bugs in later versions.

Overview of the SymmetryClassification Package

Calling Sequence

SymmetryClassification[**command**](**arguments**)
command(**arguments**)

Description

- The **SymmetryClassification** package provides a suite of commands to classify symmetries of a system of differential equations (DEs) containing arbitrary elements.
- The package utilises symmetry tools in the PDEtools package as well as a modified version of rifsimp.
- The steps in symmetry classification using this package are:

1. Derive the determining equations for point symmetries of the given DEs system.
2. Use differential reduction & completion (DRC) algorithm (i.e. rifsimp) to classify symmetries.
3. Solve classified determining equations to get the symmetries of the given DEs system.
   Note that step (3) is not included in the package. However, it can be done by pdsolve.

- The SymmetryClassification package can check whether case splittings that arise during classification are invariant under the equivalence group of the given DEs system.
- The following is a list of available commands.

`newPDESys`	This is a constructor function, which creates a pdeRecord data structure for storing information about the given DEs system.
`detEqsForSymm`	Derive the determining equations for point symmetries of the given DEs system.
`detEqsForEquiv`	Derive the determining equations of the equivalence group of the given DEs system.
`newProlongation`	This is a constructor function, which creates a module for working with the prolongation of a vector field.
`SymmetricRifsimp`	This is a modified version of rifsimp. It works exactly the same as rifsimp but it provides an additional option for using invariance properties in

	symmetry classification.
AddInvtInfo	Append invariance properties into a symmetry classification tree.
classifySymmetry	This is a front end procedure, which performs symmetry classification via a single function call.
CasePlot	This is a modified version of caseplot . It presents a symmetry classification tree graphically, optionally showing invariance properties of the case splits.

- The package uses a container structure [see pdeRecord] designed for storing various information about the given DEs system. This structure is used by other commands in the **SymmetryClassification** package.
- The package works at the level of determining equations. To explicitly find vector fields for symmetries or equivalence transformations, use pdSolve to solve the determining equations.
- For details of the symmetry classification problem, the equivalence group and many examples, see N.H. Ibragimov (1994), *"CRC Handbook of Lie Group Analysis of DEs"* vol 1.
- Each command in the **SymmetryClassification** package can be accessed by using either the long form or the short form of the command name in the command calling sequence
- As the underlying implementation of the **SymmetryClassification** package is a module, it is also possible to use the form **SymmetryClassification:-command** to access a command from the package. For more information, see Module Members.

Examples

```
> with(SymmetryClassification);
```

[*AddInvtInfo, CasePlot, ModuleLoad, SymmetricRifsimp, classifySymmetry, detEqsForEquiv, detEqsForSymm, newPDESys, newProlongation*] **(2.1)**

(1+1) Nonlinear Heat Equation

Consider (1+1) Nonlinear Heat Equation...

```
> PDE := [diff(u(x,t),t) + diff(q(x,t),x) = 0, q(x,t)
  = -K(u(x,t))*diff(u(x,t),x)];
```

$$PDE := \left[\frac{\partial}{\partial t}u(x,t) + \frac{\partial}{\partial x}q(x,t) = 0,\ q(x,t) = -K(u(x,t))\left(\frac{\partial}{\partial x}u(x,t)\right)\right] \quad \textbf{(2.2)}$$

The arbitrary element is the diffusivity function $K(u)$. First call the constructor to create a pdeRecord structure, specifying that $K(u) \neq 0$.

```
> NHEq:= newPDESys(PDE, constraint=[K(u)<>0]);
```

$$NHEq := \left[\frac{\partial}{\partial t} u(x,t) + \frac{\partial}{\partial x} q(x,t) = 0,\ q(x,t) = -K(u(x,t)) \left(\frac{\partial}{\partial x} u(x, t) \right) \right] \text{\&where } [K(u) \neq 0] \tag{2.3}$$

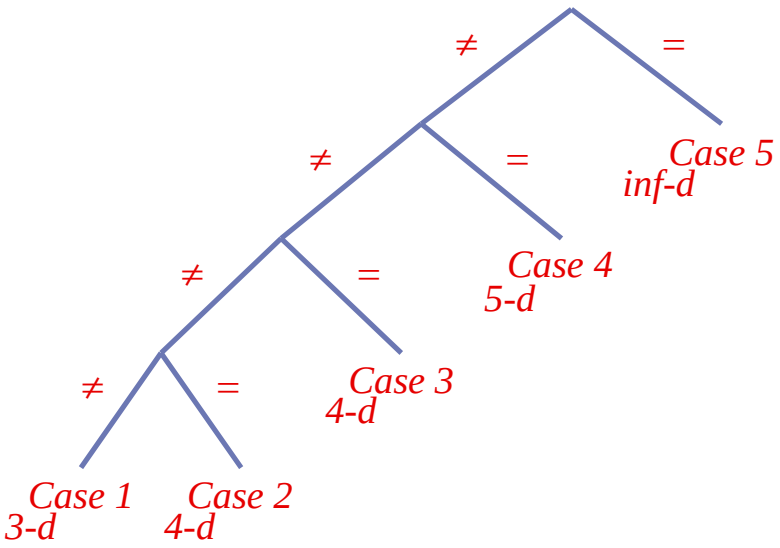
Although NHEq prints as though it is a list of equations, it is actually a module data structure containing various kinds of information about the DEs system.

2. Use the front end command to perform symmetry classification for Nonlinear Heat Equation. It returns a table structure as returned by `rifsimp`.

```
> HEqRif:=classifySymmetry(NHEq, infinitesimals=[[x,
  xi], [t,tau], [u,eta], [q,chi], [K,kappa]]):
```

3. The rif-output (HEqRif) can be displayed by calling `CasePlot`.

```
> CasePlot(HEqRif, vars=[xi,tau,eta,chi]);
```



The call to `classifySymmetry` inserted additional information into the `pdeRecord` structure. For example the determining equations for the symmetries are now present:

```
> print(NHEq:-detSymm);
```

$$\left[\frac{1}{K(u)^2} \left(\left(\frac{\partial}{\partial x} \eta(x,t,q,u) \right) K(u)^2 - q \left(\frac{\partial}{\partial u} \eta(x,t,q,u) \right) K(u) + \chi(x,t, \right. \right. \tag{2.4}$$

$$\left. q,u)\, K(u) + q \left(\frac{\partial}{\partial x} \xi(x,t,q,u) \right) K(u) - q^2 \left(\frac{\partial}{\partial u} \xi(x,t,q,u) \right) - \eta(x, \right.$$

$$t, q, u)\, q \left(\frac{\mathrm{d}}{\mathrm{d}u} K(u) \right) \Big) = 0,\ \frac{\partial}{\partial q} \tau(x, t, q, u) = 0,\ \frac{1}{K(u)} \Big(\Big(\frac{\partial}{\partial x} \tau(x,$$

$$t, q, u) \Big) K(u) + \Big(\frac{\partial}{\partial q} \eta(x, t, q, u) \Big) K(u) - q \Big(\frac{\partial}{\partial u} \tau(x, t, q, u)$$

$$- \Big(\frac{\partial}{\partial q} \xi(x, t, q, u) \Big) \Big) \Big) = 0,\ \frac{1}{K(u)} \Big(- \Big(\frac{\partial}{\partial t} \eta(x, t, q, u) \Big) K(u)$$

$$- \Big(\frac{\partial}{\partial x} \chi(x, t, q, u) \Big) K(u) + q \Big(\frac{\partial}{\partial u} \chi(x, t, q, u) - \Big(\frac{\partial}{\partial t} \xi(x, t, q,$$

$$u) \Big) \Big) \Big) = 0,\ \frac{\partial}{\partial u} \eta(x, t, q, u) + \frac{\partial}{\partial x} \xi(x, t, q, u) - \Big(\frac{\partial}{\partial t} \tau(x, t, q, u) \Big)$$
$$- \Big(\frac{\partial}{\partial q} \chi(x, t, q, u) \Big) = 0,\ \frac{\partial}{\partial q} \xi(x, t, q, u) + \frac{\partial}{\partial u} \tau(x, t, q, u) = 0,$$
$$\frac{1}{K(u)} \Big(\Big(\frac{\partial}{\partial x} \tau(x, t, q, u) \Big) K(u) - \Big(\frac{\partial}{\partial q} \eta(x, t, q, u) \Big) K(u)$$
$$- q \Big(\frac{\partial}{\partial q} \xi(x, t, q, u) + \frac{\partial}{\partial u} \tau(x, t, q, u) \Big) \Big) = 0 \Big] \text{ \&where } [K(u) \neq 0]$$

See Also

pdeRecord, PDEtools:-DeterminingPDE, DEtools[rifsimp], Module, UsingPackages, with

SymmetryClassification/pdeRecord - data structure for storing information about a DEs system containing arbitrary elements

Description

- A **pdeRecord** is a storage container for holding various kinds of information about a differential equations (DEs) system containing arbitrary elements. A **pdeRecord** is designed to suit the needs of various functions in the SymmetryClassification package.
- A **pdeRecord** is created by calling the constructor function newPDESys.
- A **pdeRecord** is a module, but is used primarily to store data. The only functions exported by a **pdeRecord** module are print methods.
- All the data fields exported by a **pdeRecord** are modules themselves. The information can be accessed by

```
pdeRecordName:-fieldName:-subFieldName:- ...;
```

- A **pdeRecord** stores two types of DE information: (1) user-specified information about a given DEs system, (2) additional information assigned by functions in the SymmetryClassification package (for example, the determining equations for symmetries and for equivalence group). These are arranged as four fields:

`pdeRecord:-`

`sys`	DEs system -- specified by user
`constraint`	Constraint system (constraints on arbitrary elements) -- specified by user
`detSymm`	Determining equations for point symmetries of DEs system -- assigned by calling detEqsForSymm
`detEquiv`	Determining equations for the equivalence group of DEs system -- assigned by calling detEqsForEquiv

- The `detEquiv` field consists of two sub-fields: `fullAction` as the determining equations for equivalence group acting on space of all variables and `arbvarsAction` as the determining equations for equivalence group acting on space of arbitrary elements only.
- After all four fields are assigned, the complete **pdeRecord** can then be used by procedures SymmetricRifsimp and AddInvtInfo.

- Each field or its sub-field contains four basic entries: `sys` as system itself, `indep` as independent variables of system, `dep` as dependent variables of system and `arbvars` as arbitrary elements (if needed).

Field "sys"

The information for field `sys` is assigned when the **pdeRecord** is created by calling newPDESys. This field contains information on the DEs system and its independent and dependent variables and arbitrary elements. Detail descriptions are listed as follows:

`pdeRecord:-sys:-`

`sys`	The DEs system, a list of equations.
`indep`	Independent variables for the DEs system, a list of names.
`dep`	Dependent variables for the DEs system, a list of function of names.
`arbvars`	Arbitrary elements for the DEs system, a list of function of names.

Field "constraint"

The information for field `constraint` is assigned when the **pdeRecord** is created by calling newPDESys. This field contains information on the constraints satisfied by the arbitrary elements, such as independent and dependent variables. Detailed description is as follows:

`pdeRecord:-constraint:-`

`sys`	Constraints on the arbitrary elements for the DEs system, a list of equations or inequations.
`indep`	Independent variables for the constraints, a list of names.
`dep`	Dependent variables for the constraints, a list of function of names.

Field "detSymm"

Information for field `detSymm` is assigned when the function detEqsForSymm is

called. This field contains information on the determining equations for point symmetries of the DEs and its independent & dependent variables and arbitrary elements. Detailed description is as follows:

`pdeRecord:-detSymm:-`

`sys`	Determining equations for symmetries for the DEs system, a list of equations.
`indep`	Independent variables for the determining system, a list of names.
`dep`	Dependent variables for the determining system, a list of function of names.
`arbvars`	Arbitrary elements for the determining system, a list of function of names.

Field "detEquiv"

Information for field `detEquiv` is assigned when the function detEqsForEquiv is called. This field contains information on the determining equations for the equivalence group of the DEs system on two different spaces which are represented as two sub-fields:

`pdeRecord:-detEquiv:-`

`fullAction`	Determining equations for equivalence group on space of all variables.
`arbvarsAction`	Determining equations for equivalence group on space of arbitrary elements.

Each sub-field contains the determining equations for the equivalence group of the DEs system, and independent and dependent variables from these determining equations. Detailed description is as follows:

`pdeRecord:-detEquiv:-fullAction:-`

`sys`	Determining equations for equivalence group from the DEs system, a list of equations.
	Independent variables of determining equations for

`indep` equivalence group, a list of names.

`dep` Dependent variables of determining equations for equivalence group, a list of function of names.

`pdeRecord:-detEquiv:-arbvarsAction:-`

`sys` Determining equations for equivalence group from the DEs system, a list of equations.

`indep` Independent variables of determining equations for equivalence group, a list of names.

`dep` Dependent variables of determining equations for equivalence group, a list of function of names.

Examples

```
> with(SymmetryClassification):
```

Nonlinear Heat Equation with $K(u) \neq 0$

```
> DEs := [diff(u(x,t),t) + diff(q(x,t),x) = 0, q(x,t)
  = -K(u(x,t))*diff(u(x,t),x)];
```

$$DEs := \left[\frac{\partial}{\partial t}u(x,t) + \frac{\partial}{\partial x}q(x,t) = 0,\ q(x,t) = -K(u(x,t))\left(\frac{\partial}{\partial x}u(x,t)\right)\right] \quad (6.1)$$

1. Create a `pdeRecord` of the Nonlinear Heat equation and its constraint by calling a constructor -- newPDESys

```
> NLHeat := newPDESys(DEs, constraint = [K(u) <>0]);
```

$$NLHeat := \left[\frac{\partial}{\partial t}u(x,t) + \frac{\partial}{\partial x}q(x,t) = 0,\ q(x,t) = -K(u(x,t))\left(\frac{\partial}{\partial x}u(x,t)\right)\right] \ \&\text{where}\ [K(u) \neq 0] \quad (6.2)$$

```
> NLHeat:-sys:-sys;
  NLHeat:-sys:-indep;
  NLHeat:-sys:-dep;
  NLHeat:-sys:-arbvars;
```

$$\left[\frac{\partial}{\partial t}u(x,t) + \frac{\partial}{\partial x}q(x,t) = 0,\ q(x,t) = -K(u(x,t))\left(\frac{\partial}{\partial x}u(x,t)\right)\right]$$

$$[x, t]$$

$$[q(x,t), u(x,t)]$$

$$[K(u)] \quad (6.3)$$

```
> NLHeat:-constraint:-sys;
```

```
NLHeat:-constraint:-indep;
NLHeat:-constraint:-dep;
```

$$[K(u) \neq 0]$$
$$[u]$$
$$[K(u)] \tag{6.4}$$

2. Assign `detSymm` field by calling detEqsForSymm.

```
> detEqsForSymm(NLHeat):
```

now we have assigned `detSymm` field. The information can be accessed by typing `NLHeat:-detSymm:-subFieldName`.

```
> NLHeat:-detSymm:-sys;
```

$$\left[\frac{1}{K(u)^2}\left(\left(\frac{\partial}{\partial x}\,\_\xi_4(x,t,q,u)\right)K(u)^2 - q\left(\frac{\partial}{\partial u}\,\_\xi_4(x,t,q,u)\right)K(u)\right.\right. \tag{6.5}$$

$$+\,\_\xi_3(x,t,q,u)\,K(u) + q\left(\frac{\partial}{\partial x}\,\_\xi_1(x,t,q,u)\right)K(u) - q^2\left(\frac{\partial}{\partial u}\,\_\xi_1(x,\right.$$

$$t,q,u)\Big) - \_\xi_4(x,t,q,u)\,q\left(\frac{\mathrm{d}}{\mathrm{d}u}\,K(u)\right)\Big) = 0,\ \frac{1}{K(u)}\left(\left(\frac{\partial}{\partial t}\,\_\xi_4(x,\right.\right.$$

$$t,q,u)\Big)K(u) + q\left(\frac{\partial}{\partial t}\,\_\xi_1(x,t,q,u)\right) - q\left(\frac{\partial}{\partial u}\,\_\xi_3(x,t,q,u)\right)$$

$$\left.\left.+\left(\frac{\partial}{\partial x}\,\_\xi_3(x,t,q,u)\right)K(u)\right) = 0,\ \frac{\partial}{\partial q}\,\_\xi_2(x,t,q,u) = 0,\right.$$

$$\frac{1}{K(u)}\left(\left(\frac{\partial}{\partial x}\,\_\xi_2(x,t,q,u)\right)K(u) + K(u)\left(\frac{\partial}{\partial q}\,\_\xi_4(x,t,q,u)\right) + q\left(\right.\right.$$

$$\left.\left.-\left(\frac{\partial}{\partial u}\,\_\xi_2(x,t,q,u)\right) + \frac{\partial}{\partial q}\,\_\xi_1(x,t,q,u)\right)\right) = 0,\ -\left(\frac{\partial}{\partial u}\,\_\xi_4(x,t,q,\right.$$

$$u)\Big) - \left(\frac{\partial}{\partial x}\,\_\xi_1(x,t,q,u)\right) + \frac{\partial}{\partial t}\,\_\xi_2(x,t,q,u) + \frac{\partial}{\partial q}\,\_\xi_3(x,t,q,u)$$
$$= 0,\ \frac{1}{K(u)}\left(-\left(\frac{\partial}{\partial x}\,\_\xi_2(x,t,q,u)\right)K(u) + K(u)\left(\frac{\partial}{\partial q}\,\_\xi_4(x,t,q,\right.\right.$$
$$\left.\left.u)\Big) + q\left(\frac{\partial}{\partial q}\,\_\xi_1(x,t,q,u) + \frac{\partial}{\partial u}\,\_\xi_2(x,t,q,u)\right)\right) = 0,\ -\left(\frac{\partial}{\partial q}\,\_\xi_1(x,\right.$$

$$t, q, u)\Big) - \left(\frac{\partial}{\partial u}\ \_\xi_2(x, t, q, u)\right) = 0\Big]$$

```
> NLHeat:-detSymm:-indep;
  NLHeat:-detSymm:-dep;
  NLHeat:-detSymm:-arbvars;
```

$$[x, t, q, u]$$

$$[\_\xi_1(x, t, q, u), \_\xi_2(x, t, q, u), \_\xi_3(x, t, q, u), \_\xi_4(x, t, q, u)]$$

$$[K(u)] \tag{6.6}$$

3. Assign `detEquiv` field by calling detEqsForEquiv. This field consists of two sub-fields: `fullAction` and `arbvarsAction`. The information can be accessed by typing `NLHeat:-detEquiv:-ActionName:-subFieldName`.

```
> detEqsForEquiv(NLHeat):
```

sub-field fullAction

```
> NLHeat:-detEquiv:-fullAction:-sys;
```

$$\left[\frac{\mathrm{d}}{\mathrm{d}x}\ \_\xi_1(x) = \frac{q\left(\frac{\mathrm{d}}{\mathrm{d}u}\ \_\xi_4(u)\right)K + \_\xi_5(K)\,q - \_\xi_3(q)\,K}{q\,K}, \frac{\mathrm{d}}{\mathrm{d}t}\ \_\xi_2(t)\right. \tag{6.7}$$

$$= \frac{2\,q\left(\frac{\mathrm{d}}{\mathrm{d}u}\ \_\xi_4(u)\right)K + \_\xi_5(K)\,q - 2\,\_\xi_3(q)\,K}{q\,K}, \frac{\mathrm{d}}{\mathrm{d}q}\ \_\xi_3(q)$$

$$\left. = \frac{\_\xi_3(q)}{q}, \frac{\mathrm{d}}{\mathrm{d}K}\ \_\xi_5(K) = \frac{\_\xi_5(K)}{K}, \frac{\mathrm{d}^2}{\mathrm{d}u^2}\ \_\xi_4(u) = 0\right]$$

```
> NLHeat:-detEquiv:-fullAction:-indep;
  NLHeat:-detEquiv:-fullAction:-dep;
```

$$[x, t, q, u, K]$$

$$[\_\xi_1(x), \_\xi_2(t), \_\xi_3(q), \_\xi_4(u), \_\xi_5(K)] \tag{6.8}$$

sub-field `arbvarsAction`

```
> NLHeat:-detEquiv:-arbvarsAction:-sys;
```

$$\left[\frac{\mathrm{d}}{\mathrm{d}K}\ \_\xi_5(K) = \frac{\_\xi_5(K)}{K}, \frac{\mathrm{d}^2}{\mathrm{d}u^2}\ \_\xi_4(u) = 0\right] \tag{6.9}$$

```
> NLHeat:-detEquiv:-arbvarsAction:-indep;
  NLHeat:-detEquiv:-arbvarsAction:-dep;
```

$$[u, K]$$

$$[\_\xi_4(u), \_\xi_5(K)] \tag{6.10}$$

See Also

newPDESys, detEqsForSymm, detEqsForEquiv, SymmetricRifsimp, AddInvtInfo, SymmetryClassification

SymmetryClassification[newPDESys] - constructor method for pdeRecord data structure

Calling Sequence

newPDESys(**system**)
newPDESys(**system, options**)

Parameters

`system` - list or set of differential equations
`options` - (optional) list of option(s) of the form **option=value** where **option** is one of **indeps**, **deps**, **arbitrary**, or **constraint**; specify options for the **newPDESys** command

Description

- **newPDESys** is a constructor function for a pdeRecord data structure, for use in the SymmetryClassification package. The purpose of this command is to create a container for storing various information on differential equations (DEs) system given by user.
- The input parameter `system` should be a DEs system with arbitrary elements (constants or functions). Such elements may represent physical properties like wave speeds, diffusivities, etc.
- **newPDESys** creates and returns a pdeRecord module data structure for storing DEs system. The pdeRecord can then be used by other commands in the SymmetryClassification package. With all DEs system information which are specified by a user will be stored in the pdeRecord, then returns the `pdeRecord`.
- Additional information to the DEs system can be specified in options. Options are listed below:
- **indep=[var1, ...]**
  Specify list of independent variables of the DEs system.
- **dep=[varFunc1, ...]**
  Specify list of dependent variables of the DEs system. Type of **varFunc1, ...** has to be function of name.
- **arbitrary=[var1, ...]**
  Specify list of arbitrary elements of the DEs system. Type of **var1, ...** can be either name or function of name.
- **constraint=[eq1, ...]**
  Specify list of constraints on the arbitrary elements of the DEs system. Type of **eq1, ...** can be either equation or inequation.
- If options `indep=`, `dep=`, or `arbitrary=` are not specified, then **newPDESys** will automatically extract this information from the input DEs system. These

options only need to be specified only when there is a risk of ambiguity. For example, for the DE $y(x)\left(\frac{d^2}{dx^2}y(x)\right) = -\frac{1}{2}f(x)$ it is necessary to specify `dep=[y(x)], arbitrary=[f(x)]`, otherwise $f(x)$ may be interpreted as an additional dependent variable (see Example 3 below).

- **newPDESys** checks the validity of the information specified by the user. It will override any invalid user specification.

Examples

Set up

```
> with(SymmetryClassification);
```

[*AddInvtInfo, CasePlot, ModuleLoad, SymmetricRifsimp, classifySymmetry, detEqsForEquiv, detEqsForSymm, newPDESys, newProlongation*] **(2.1)**

Example 1

Nonlinear Diffusion Equation

```
> DEs := [diff(u(x,t),t) + diff(q(x,t),x) = 0, q(x,t)
  = -K(u(x,t))*diff(u(x,t),x)];
```

$$DEs := \left[\frac{\partial}{\partial t}u(x,t) + \frac{\partial}{\partial x}q(x,t) = 0,\ q(x,t) = -K(u(x,t))\left(\frac{\partial}{\partial x}u(x,t)\right)\right] \quad \textbf{(2.2)}$$

Call **newPDESys** to create a `pdeRecord` for the DEs system. NLHeat is now a `pdeRecord`. To access information of the `pdeRecord`, see pdeRecord for detail.

```
> NLHeat := newPDESys(DEs);
```

$$NLHeat := \left[\frac{\partial}{\partial t}u(x,t) + \frac{\partial}{\partial x}q(x,t) = 0,\ q(x,t) = -K(u(x,t))\left(\frac{\partial}{\partial x}u(x,t)\right)\right] \quad \textbf{(2.3)}$$

Example 2

In this example, x and t are certainly independent variables. But a is interpreted as an arbitrary constant.

```
> DEs := [a*diff(u(x,t),t) + diff(u(x,t),x) = 0];
```

$$DEs := \left[a\left(\frac{\partial}{\partial t}u(x,t)\right) + \frac{\partial}{\partial x}u(x,t) = 0\right] \quad \textbf{(2.4)}$$

Call **newPDESys** to create a `pdeRecord` for the DEs system.

```
> DERecord := newPDESys(DEs);
```

$$DERecord := \left[a\left(\frac{\partial}{\partial t}u(x,t)\right) + \frac{\partial}{\partial x}u(x,t) = 0\right] \quad \textbf{(2.5)}$$

```
> DERecord:-sys:-indep;
  DERecord:-sys:-arbvars;
```

$$[x, t]$$

$$[a] \tag{2.6}$$

If a is meant to be an independent variable, it must be specified...

```
> DERecord := newPDESys(DEs, indeps=[a]);
```

$$DERecord := \left[a\left(\frac{\partial}{\partial t}u(x,t)\right) + \frac{\partial}{\partial x}u(x,t) = 0\right] \tag{2.7}$$

```
> DERecord:-sys:-indep;
  DERecord:-sys:-arbvars;
```

$$[x, t, a]$$
$$[\,] \tag{2.8}$$

Example 3

```
> DEs := [y(x)*diff(y(x),x,x)= -1/2* f(x)];
```

$$DEs := \left[y(x)\left(\frac{d^2}{dx^2}y(x)\right) = -\frac{1}{2}f(x)\right] \tag{2.9}$$

Call **newPDESys** to create a `pdeRecord` for the DEs system with $f(x)$ unspecified. The function $f(x)$ is treated as dependent variable.

```
> DERecord := newPDESys(DEs);
```

$$DERecord := \left[y(x)\left(\frac{d^2}{dx^2}y(x)\right) = -\frac{1}{2}f(x)\right] \tag{2.10}$$

```
> DERecord:-sys:-dep;
  DERecord:-sys:-arbvars;
```

$$[f(x), y(x)]$$
$$[\,] \tag{2.11}$$

Call **newPDESys** again but this time we specify the function $f(x)$ to be an arbitrary element.

```
> DERecord := newPDESys(DEs, dep=[y(x)], arbitrary=[f
  (x)]);
```

$$DERecord := \left[y(x)\left(\frac{d^2}{dx^2}y(x)\right) = -\frac{1}{2}f(x)\right] \tag{2.12}$$

```
> DERecord:-sys:-dep;
  DERecord:-sys:-arbvars;
```

$$[y(x)]$$
$$[f(x)] \tag{2.13}$$

See Also

newPDESys, detEqsForSymm, detEqsForEquiv, pdeRecord, SymmetryClassification

SymmetryClassification[detEqsForSymm] - find determining equations for point symmetries of DEs

Calling Sequence

detEqsForSymm(**DERecord**)
detEqsForSymm(**DERecord, infinitesimals=value**)

Parameters

`DERecord`	-	a `pdeRecord` module data structure
`infinitesimals=value`	-	(optional) specify a list or set of infinitesimals

Description

- Given a pdeRecord `DERecord` which contains information on a differential equations (DEs) system, **detEqsForSymm** derives the determining equations for point symmetries of the system.
- **detEqsForSymm** requires a pdeRecord, which is created via the constructor newPDESys.
- The determining equations are stored in the pdeRecord by assigning the `detSymm` field in the data structure. The returned value of **detEqsForSymm** is the updated pdeRecord.
- **detEqsForSymm** calls the existing procedure DeterminingPDE to derive the determining equations.
- **detEqsForSymm** allows user to specify the infinitesimals and their dependencies.
- **infinitesimals=value**
  This option is for the user to specify their own infinitesimals with corresponding variables. The value can be a list or set of the form

```
[[var1, infinitesimal1], [var2, infinitesimal2], ...
]
```

  where `var1, var2, ...` are the names of variables, and `infinitesimal1, infinitesimal2, ...` are the corresponding infinitesimals. The infinitesimals can be names or functions. **detEqsForSymm** will choose sensible defaults if none are specified.

Examples

```
> with(SymmetryClassification);
```

[*AddInvtInfo, CasePlot, ModuleLoad, SymmetricRifsimp, classifySymmetry, detEqsForEquiv, detEqsForSymm, newPDESys, newProlongation*] **(2.1)**

Typesetting setup

```
> with(Typesetting): Settings(userep=true):
> interface(typesetting=extended):
> Suppress({u(x,t),q(x,t), K(u)});
```

Create a pdeRecord for Nonlinear Diffusion Equation.

```
> DEs := [diff(u(x,t),t) + diff(q(x,t),x) = 0, q(x,t)
  = -K(u(x,t))*diff(u(x,t),x)];
```

$$DEs := \left[u_t + q_x = 0,\ q = -K(u)\,u_x\right] \tag{2.2}$$

```
> NLHeat := newPDESys(DEs, constraint = [K(u)<>0]);
```

$$NLHeat := \left[u_t + q_x = 0,\ q = -K(u)\,u_x\right] \&\text{where } [K \neq 0] \tag{2.3}$$

1. Find determining equations -- no infinitesimals specified

```
> detEqsForSymm(NLHeat):
```

Note that **detEqsForSymm** chooses default names for infinitesimals.

```
> print(NLHeat:-detSymm);
```

$$\left[\frac{1}{K}\left(K\frac{\partial}{\partial x}\,\_\xi_2(x,t,q,u) + \frac{\partial}{\partial q}\,\_\xi_4(x,t,q,u)\,K - q\left(\frac{\partial}{\partial u}\,\_\xi_2(x,t,q,u)\right.\right.\right. \tag{2.4}$$

$$\left.\left.\left. - \frac{\partial}{\partial q}\,\_\xi_1(x,t,q,u)\right)\right) = 0,\ \frac{1}{K}\left(-\frac{\partial}{\partial t}\,\_\xi_4(x,t,q,u)\,K - q\,\frac{\partial}{\partial t}\right.\right.$$

$$\left.\left._\xi_1(x,t,q,u) + q\,\frac{\partial}{\partial u}\,\_\xi_3(x,t,q,u) - \frac{\partial}{\partial x}\,\_\xi_3(x,t,q,u)\,K\right) = 0,$$

$$\frac{1}{K^2}\left(\frac{\partial}{\partial x}\,\_\xi_4(x,t,q,u)\,K^2 - q\,\frac{\partial}{\partial u}\,\_\xi_4(x,t,q,u)\,K + \_\xi_3(x,t,q,u)\,K\right.$$

$$\left. + q\,\frac{\partial}{\partial x}\,\_\xi_1(x,t,q,u)\,K - q^2\,\frac{\partial}{\partial u}\,\_\xi_1(x,t,q,u) - \_\xi_4(x,t,q,u)\,q\,\frac{dK}{du}\right)$$

$$= 0,\ \frac{1}{K}\left(K\frac{\partial}{\partial x}\,\_\xi_2(x,t,q,u) - \frac{\partial}{\partial q}\,\_\xi_4(x,t,q,u)\,K - q\left(\frac{\partial}{\partial u}\right.\right.$$

$$\left.\left._\xi_2(x,t,q,u) + \frac{\partial}{\partial q}\,\_\xi_1(x,t,q,u)\right)\right) = 0,\ \frac{\partial}{\partial q}\,\_\xi_2(x,t,q,u) = 0,\ \frac{\partial}{\partial u}$$

$$\_\xi_2(x,t,q,u) + \frac{\partial}{\partial q}\,\_\xi_1(x,t,q,u) = 0,\ \frac{\partial}{\partial u}\,\_\xi_4(x,t,q,u) + \frac{\partial}{\partial x}\,\_\xi_1(x,t,$$

$$q,u) - \frac{\partial}{\partial t}\,\_\xi_2(x,t,q,u) - \frac{\partial}{\partial q}\,\_\xi_3(x,t,q,u) = 0\Big] \&\text{where } [K \neq 0]$$

2. Find determining equations -- infinitesimal names specified

```
> Suppress({xi(x,t,u),tau(x,t,u),eta(x,t,u), chi(x,t,
  q,u)});
```

```
> detEqsForSymm(NLHeat, infinitesimals=[[x, xi(x,t,u)]
  , [t, tau(x,t,u)], [u, eta(x,t,u)], [q, chi]]):
```

```
> print(NLHeat:-detSymm);
```

$$\left[\frac{-\eta_x K^2 + q\,\eta_u K - q\,\xi_x K + q^2\,\xi_u + \eta\,q\,\frac{dK}{du} - \chi K}{K^2} = 0,\ -\tau_t + \eta_u - \chi_q + \xi_x \right. \tag{2.5}$$

$$= 0,\ \frac{-\tau_x K + q\,\tau_u}{K} = 0,\ \frac{\tau_x K - q\,\tau_u}{K} = 0,\ \frac{-\eta_t K - \chi_x K - q\,\xi_t + q\,\chi_u}{K}$$

$$\left. = 0,\ \tau_u = 0 \right] \ \&\text{where}\ [K \neq 0]$$

By default, printing the detSymm sub-module shows only the system of determining equations. But other other information about the determining equations is stored and can be accessed manually [see pdeRecord/detSymm for detail].

See Also

DeterminingPDE, newPDESys, pdeRecord, SymmetryClassification

SymmetryClassification[detEqsForEquiv] - find determining equations for the equivalence group

Calling Sequence

detEqsForEquiv(**DERecord**)
detEqsForEquiv(**DERecord, infinitesimals=value**)

Parameters

`DERecord` - a pdeRecord module data structure
`infinitesimals=value` - (optional) specify a list or set of infinitesimals

Description

- Given a pdeRecord `DERecord` which contains information on a differential equations (DEs) system, **detEqsForEquiv** derives the determining equations for the equivalence group of the system. The equivalence transformation is a point transformation on the space of independent and dependent variables leaving invariant the family of DEs.
- **detEqsForEquiv** requires a pdeRecord, which is created via the constructor newPDESys.
- The determining equations are stored in the pdeRecord by assigning the `detEquiv` field in the data structure. The return value of **detEqsForEquiv** is the updated pdeRecord.
- **detEqsForEquiv** derives the determining equations by first calling the procedure DeterminingPDE for each of DEs system and constraint system, then combining these two and completing it by calling rifsimp.
- **detEqsForEquiv** allows user to specify the infinitesimal names and their dependencies.
- **infinitesimals=value**
  This option is for user to specify its own infinitesimals with corresponding variables. The value can be a list or set of the form

```
[[var1, infinitesimal1], [var2, infinitesimal2], ...
]
```

  where `var1, var2, ...` are the names of variables, and `infinitesimal1, infinitesimal2, ...` are the corresponding infinitesimals. The infinitesimals can be names or functions. **detEqsForEquiv** will choose sensible default if none are specified.

Examples

```
> with(SymmetryClassification);
```

[*AddInvtInfo, CasePlot, ModuleLoad, SymmetricRifsimp, classifySymmetry, detEqsForEquiv, detEqsForSymm, newPDESys, newProlongation*] **(2.1)**

Typesetting setup

```
> with(Typesetting): Settings(userep=true, usedot=
  false, useprime=false):
> interface(typesetting=extended):
> Suppress({u(x,t), q(x,t), K(u)});
```

Create a pdeRecord for Nonlinear Diffusion Equation

```
> DEs := [diff(u(x,t),t) + diff(q(x,t),x) = 0, q(x,t)
  = -K(u(x,t))*diff(u(x,t),x)];
```

$$DEs := [u_t + q_x = 0,\ q = -K(u)\,u_x] \quad \textbf{(2.2)}$$

```
> NLHeat := newPDESys(DEs, constraint=[K(u)<>0]);
```

$$NLHeat := [u_t + q_x = 0,\ q = -K(u)\,u_x]\ \&\text{where}\ [K \neq 0] \quad \textbf{(2.3)}$$

1. Find determining equations for the equivalence group

```
> detEqsForEquiv(NLHeat):
```

Note that this is full-action...

```
> print(NLHeat:-detEquiv);
```

$$\left[\frac{\mathrm{d}}{\mathrm{d}x}\,\_\xi_1(x) = \frac{q\,\frac{\mathrm{d}}{\mathrm{d}u}\,\_\xi_4(u)\,K - \_\xi_3(q)\,K + \_\xi_5(K)\,q}{q\,K},\ \frac{\mathrm{d}}{\mathrm{d}t}\,\_\xi_2(t) \right. \quad \textbf{(2.4)}$$

$$= \frac{2\,q\,\frac{\mathrm{d}}{\mathrm{d}u}\,\_\xi_4(u)\,K - 2\,\_\xi_3(q)\,K + \_\xi_5(K)\,q}{q\,K},\ \frac{\mathrm{d}}{\mathrm{d}q}\,\_\xi_3(q) = \frac{\_\xi_3(q)}{q},$$

$$\left.\frac{\mathrm{d}}{\mathrm{d}K}\,\_\xi_5(K) = \frac{\_\xi_5(K)}{K},\ \frac{\mathrm{d}^2}{\mathrm{d}u^2}\,\_\xi_4(u) = 0\right]$$

Default names for the infinitesimals were chosen by **detEqsForEquiv** since the user didn't specify. By looking at the lists of variables it can be seen which infinitesimal goes with which variable...

```
> NLHeat:-detEquiv:-fullAction:-indep;
> NLHeat:-detEquiv:-fullAction:-dep;
```

$$[x, t, q, u, K]$$

$$[\_\xi_1(x), \_\xi_2(t), \_\xi_3(q), \_\xi_4(u), \_\xi_5(K)] \quad \textbf{(2.5)}$$

We also can view arbvars-action by

```
> print(NLHeat:-detEquiv:-arbvarsAction);
```

(2.6)

$$\left[\frac{d}{dK}\ \_\xi_5(K) = \frac{-\_\xi_5(K)}{K},\ \frac{d^2}{du^2}\ \_\xi_4(u) = 0\right] \tag{2.6}$$

2. Find determining equations for the equivalence group (with infinitesimal names specified)

```
> Suppress({xi(x), tau(t), eta(u), chi(q), kappa(K)});
> detEqsForEquiv(NLHeat, infinitesimals = [[x, xi],
  [t, tau], [u, eta], [q, chi], [K, kappa]]):
> print(NLHeat:-detEquiv);
```

$$\left[\frac{d\xi}{dx} = \frac{q\,\frac{d\eta}{du}\,K + q\,\kappa - \chi\,K}{q\,K},\ \frac{d\tau}{dt} = \frac{2\,q\,\frac{d\eta}{du}\,K + q\,\kappa - 2\,\chi\,K}{q\,K},\ \frac{d\chi}{dq} = \frac{\chi}{q},\right. \tag{2.7}$$

$$\left.\frac{d\kappa}{dK} = \frac{\kappa}{K},\ \frac{d^2\eta}{du^2} = 0\right]$$

3. Find determining equations for the equivalence group (only some infinitesimal names and dependencies specified)

In this case, **detEqsForEquiv** gives the infinitesimal κ the default dependency $\kappa(x, t, q, u, K)$. Also since no infinitesimal was specified for q, a default name is chosen and default dependency on x, t, q, u is assumed.

```
> detEqsForEquiv(NLHeat, infinitesimals = [[x, xi(x,t)
  ], [t, tau(x,t)], [u, eta(u)], [K, kappa]]):
> print(NLHeat:-detEquiv);
```

$$\left[\frac{d\xi}{dx} = \frac{q\,\frac{d\eta}{du}\,K + q\,\kappa - \_\xi_1(q)\,K}{q\,K},\ \frac{d\tau}{dt} = \frac{2\,q\,\frac{d\eta}{du}\,K + q\,\kappa - 2\,\_\xi_1(q)\,K}{q\,K},\right. \tag{2.8}$$

$$\left.\frac{d}{dq}\ \_\xi_1(q) = \frac{-\_\xi_1(q)}{q},\ \frac{d\kappa}{dK} = \frac{\kappa}{K},\ \frac{d^2\eta}{du^2} = 0\right]$$

Although the infinitesimals start out with these dependencies allowed, **detEqsForEquiv** has found constraints on the infinitesimals that allow it to drop variables from the dependency lists.

See Also

DEtools[rifsimp], DeterminingPDE, newPDESys, pdeRecord, SymmetryClassification

SymmetryClassification[newProlongation] - constructor for calculation with prolonged vector fields

Calling Sequence

newProlongation(**baseVF**, **dep**, **indep**, **detsys**)

Parameters

`baseVF` - list of ordered pairs; vector field on base space of independent and dependent variables

`dep` - list of names or functions; the dependent variables for the prolongation

`indep` - list of names; the independent variables for the prolongation

`detsys` - list of equations; determining differential equations satisfied by the base vector field

Description

- The **newProlongation** command returns a module that facilitates computation with a prolonged vector field. It is similar in intent to the Eta_k command in the PDETools package.
- `baseVF` is a list of ordered pairs giving the names and functional dependencies of the infinitesimals of the base vector field, of the form
  `[ [v1, inf1(v1,...)], ...]`
  where each variable v1, ... is the name of an independent or dependent variable, and inf1(v1, ...) specifies the name and dependency of the corresponding infinitesimal. See the examples below.
- A dependent variable given as a name (e.g. *a*) is assumed to be a constant, that is a 'dependent variable' with null dependency.
- The components of the vector field can be constrained to satisfy a system of determining equations. In this case, the module returned by **newProlongation** takes account of this determining system. For proper functioning, it is required that the determining system be reduced and completed (e.g. by the DEtools[rifsimp] command). If detsys is an empty list, then **newProlongation** finds the prolongation of a point transformation.
- **newProlongation** returns a module, whose exports give access to information about the prolongation. The exports `applyProlongedVF` and `checkInvariant` will be sufficient for most users; the other exports will mostly be of interest to programmers.

`applyProlongedVF(expr)` `applyProlongedVF([expr1, ...])`	Function that applies the prolonged vector field to expression `expr` or list of expressions `[expr1, ...]`.

`checkInvariant(eq)` `checkInvariant([eq1, ...])`	function that checks whether equation eq is invariant under the action of the prolonged vector field. Return value is boolean (true if invariant, false otherwise).
`prolongToOrder()` `prolongToOrder(k)`	The first form returns the current order of prolongation. The second form ensures that prolongation components up to order *k* have been computed.
`prolongVF(k)`	Function that returns the form of a prolonged vector field to order k, in jet notation, as a list of ordered pairs [[var1, inf1], ...]. (Not reduced modulo `detsys`.)
`jetSubsRules(k)`	Function that returns a list of substitution rules giving the values of the infinitesimals of a vector field prolonged to order k. Substituting these into the result of `prolongVF(k)` will give the prolonged vector field reduced mod `detsys`.

- Most of the commands dealing with a prolongation use jet notation (see ToJet, FromJet) for the jet variables. The corresponding infinitesimals are given a similar notation.
- **newProlongation** is used internally by various functions in the SymmetryClassification package.

Examples

```
> with(SymmetryClassification);
```

[*AddInvtInfo, CasePlot, ModuleLoad, SymmetricRifsimp, classifySymmetry, detEqsForEquiv, detEqsForSymm, newPDESys, newProlongation*] **(2.1)**

Example 1: Prolongation of scaling & translation group

Consider the vector fields $\xi(x,y)\left(\frac{\partial}{\partial x}\right)+\eta(x,y)\left(\frac{\partial}{\partial y}\right)$ in the plane whose infinitesimals ξ, η are $\xi = c1\,x + c2$, η=c3 (This generates a group of scalings and translations x' = a*x+b, y'=e*y where a,e are nonzero). Take *x* as independent variable and *y* as dependent.

```
> indep := [x];
```

$$indep := [x] \quad \textbf{(2.1.1)}$$

```
> dep   := [y(x)];
```

$$dep := [y(x)] \quad \textbf{(2.1.2)}$$

```
> baseVF := [ [x,xi(x,y)], [y,eta(x,y)]];
```

$$baseVF := [[x, \xi(x,y)], [y, \eta(x,y)]] \quad \textbf{(2.1.3)}$$

The 'determining system' here effectively is substitutions for the values of ξ, η...

```
> detsys := [xi(x,y)=c1*x+c2,
             eta(x,y)=c3*y];
```

$$detsys := [\xi(x,y) = c1\,x + c2, \eta(x,y) = c3\,y] \quad \textbf{(2.1.4)}$$

```
> pr := newProlongation(baseVF, dep, indep, detsys);
```

$$pr := \_prolongation\_of\left(\xi\left(\frac{\partial}{\partial x}\right) + \eta\left(\frac{\partial}{\partial y}\right)\right) \quad \textbf{(2.1.5)}$$

Apply the prolonged vector field to an expression...

```
> pr:-applyProlongedVF(diff(y(x),x,x));
```

$$y_{1,1}\,c3 - 2\,y_{1,1}\,c1 \quad \textbf{(2.1.6)}$$

Check whether a differential equation is invariant under the prolonged vector fields...

```
> pr:-checkInvariant(diff(y(x),x,x)=0);
```

$$true \quad \textbf{(2.1.7)}$$

Example 2: Prolongation from determining equations

Now redo Example 1, but instead of specifying the vector fields explicitly, instead give the determining equations the infinitesimals ξ, η satisfy, namely $\xi_{x,x} = 0, \xi_y = 0, \eta_x = 0, \eta_y = \frac{1\,\eta}{y}$ (This system is in differentially complete form e. g. as returned by rifsimp.)

```
> detsys := [diff(xi(x,y),x,x) = 0,
             diff(xi(x,y),y) = 0,
             diff(eta(x,y),x) = 0,
             diff(eta(x,y),y)=1/y*eta(x,y)];
```

$$detsys := \left[\frac{\partial^2}{\partial x^2}\xi(x,y) = 0, \frac{\partial}{\partial y}\xi(x,y) = 0, \frac{\partial}{\partial x}\eta(x,y) = 0, \frac{\partial}{\partial y}\eta(x,y) = \frac{\eta(x,y)}{y}\right] \quad \textbf{(2.2.1)}$$

```
> pr := newProlongation(baseVF, dep, indep, detsys);
```

$$pr := \_prolongation\_of\left(\xi\left(\frac{\partial}{\partial x}\right) + \eta\left(\frac{\partial}{\partial y}\right)\right) \quad \textbf{(2.2.2)}$$

Apply the prolonged vector field to an expression...

```
> pr:-applyProlongedVF(diff(y(x),x,x));
```

$$-\frac{y_{1,1}\left(-\eta(x,y)+2\left(\frac{\partial}{\partial x}\xi(x,y)\right)y\right)}{y} \tag{2.2.3}$$

Check whether a differential equation is invariant under the prolonged vector fields...

```
> pr:-checkInvariant(diff(y(x),x,x)=0);
```

$$true \tag{2.2.4}$$

For completeness, look at all the other exports of the prolongation module...

Check what order prolongation has been found so far...

```
> pr:-prolongToOrder();
```

$$2 \tag{2.2.5}$$

Ask explicitly for prolongation to order 3 to be computed...

```
> pr:-prolongToOrder(3);
```

$$3 \tag{2.2.6}$$

Look at the form of the prolonged vector field of order 2 (the variables are in jet notation)...

```
> pr:-prolongVF(2);
```

$$[[x,\xi],[y,\eta],[y_1,\eta_1],[y_{1,1},\eta_{1,1}]] \tag{2.2.7}$$

The components of the prolonged vector field are given by formulas returned by `jetSubsRules` ...

```
> pr:-jetSubsRules(2);
```

$$\left[\xi=\xi(x,y),\ \eta=\eta(x,y),\ \eta_1=-\frac{y_1\left(-\eta(x,y)+\left(\frac{\partial}{\partial x}\xi(x,y)\right)y\right)}{y},\ \eta_{1,1}\right. \tag{2.2.8}$$

$$\left.=-\frac{y_{1,1}\left(-\eta(x,y)+2\left(\frac{\partial}{\partial x}\xi(x,y)\right)y\right)}{y}\right]$$

So the reduced prolonged vector field is:

```
> subs(%, %%);
```

$$\left[[x,\xi(x,y)],[y,\eta(x,y)],\left[y_1,-\frac{y_1\left(-\eta(x,y)+\left(\frac{\partial}{\partial x}\xi(x,y)\right)y\right)}{y}\right],\right. \tag{2.2.9}$$

$$\left.\left[y_{1,1},-\frac{y_{1,1}\left(-\eta(x,y)+2\left(\frac{\partial}{\partial x}\xi(x,y)\right)y\right)}{y}\right]\right]$$

Example 3: Prolongation of conformal vector fields

A feature of **newProlongation** is that it works directly at the level of determining

equations. This is especially important when there is not a 'neat' way to write the vector fields from a particular group. Consider conformal vector fields in the plane $\xi\left(\frac{\partial}{\partial x}\right)+\eta\left(\frac{\partial}{\partial y}\right)+\psi\left(\frac{\partial}{\partial u}\right)$ where ξ, η satisfy the Cauchy-Riemann equations: $\eta_y=\xi_x$, $\xi_y=-\eta_x$ and where ψ=0. We take x,y as indep and u as dep..

```
> indep := [x,y];
```

$$indep := [x, y] \tag{2.3.1}$$

```
> dep   := [u(x,y)];
```

$$dep := [u(x, y)] \tag{2.3.2}$$

```
> baseVF := [ [x,xi(x,y)], [y,eta(x,y)], [u,psi(x,y,
  u)]];
```

$$baseVF := [[x, \xi(x, y)], [y, \eta(x, y)], [u, \psi(x, y, u)]] \tag{2.3.3}$$

```
> detsys := [diff(eta(x,y),x) = - diff(xi(x,y),y),
             diff(xi(x,y),x)  = diff(eta(x,y),y),
             psi(x,y,u)=0];
```

$$detsys := \left[\frac{\partial}{\partial x}\eta(x, y) = -\left(\frac{\partial}{\partial y}\xi(x, y)\right), \frac{\partial}{\partial x}\xi(x, y) = \frac{\partial}{\partial y}\eta(x, y), \psi(x, y, u) = 0\right] \tag{2.3.4}$$

```
> pr := newProlongation(baseVF, dep, indep, detsys);
```

$$pr := \_prolongation\_of\left(\xi\left(\frac{\partial}{\partial x}\right)+\eta\left(\frac{\partial}{\partial y}\right)+\psi\left(\frac{\partial}{\partial u}\right)\right) \tag{2.3.5}$$

Apply the prolonged vector field to an expression...

```
> pr:-applyProlongedVF(diff(u(x,y),x,x) + diff(u(x,
  y),y,y));
```

$$-2\,u_{1,1}\left(\frac{\partial}{\partial y}\eta(x, y)\right)-2\,u_{2,2}\left(\frac{\partial}{\partial y}\eta(x, y)\right) \tag{2.3.6}$$

Check whether a differential equation is invariant under the prolonged vector fields...

```
> pr:-checkInvariant(diff(u(x,y),x,x) + diff(u(x,y),
  y,y)=0);
```

$$true \tag{2.3.7}$$

Show expressions for the jet infinitesimals of a prolongation...

```
> pr:-jetSubsRules(1);
```

$$\left[\xi=\xi(x, y), \eta=\eta(x, y), \psi=0, \psi_1=-u_1\left(\frac{\partial}{\partial y}\eta(x, y)\right)+u_2\left(\frac{\partial}{\partial y}\xi(x,\right.\right. \tag{2.3.8}$$

$$y)\Big), \psi_2 = -u_1\left(\frac{\partial}{\partial y}\,\xi(x,y)\right) - u_2\left(\frac{\partial}{\partial y}\,\eta(x,y)\right)\Big]$$

See Also

Eta_k, SymmetryClassification

SymmetryClassification[AddInvtInfo] - label invariant properties on splitting conditions of a symmetry classification

Calling Sequence

AddInvtInfo(**rifoutput**, **detEquivInArbvars**, **arbvars**)

Parameters

`rifoutput` - output from rifsimp

`detEquivInArbvars` - the sub-field `detEquiv:-arbvarsAction` from a `pdeRecord` module data structure

`arbvars` - a list of arbitrary elements as function of names

Description

- **AddInvtInfo** takes a symmetry classification tree and labels its case splits with invariance information.
- The input parameter `rifoutput` is a table which has been returned by rifsimp with option `casesplit` [see rifismp/output for more detail]. The rif-output is assumed to be the result of running rifsimp on the determining equations of a symmetry classification problem.
- The parameter `detEquivInArbvars` specifies the determining equations of a group. The splitting conditions of the symmetry classification are tested to see if they are invariant under this group.
- The parameter `arbvars` is a list of the arbitrary elements occurring in the symmetry classification. These are the dependent variables occurring in the case splitting equations of the rif-output. They can be found in the subfield `detSymm:-arbvars` of a `pdeRecord`.
- Case splits of a symmetry classification should be invariant under the action of the equivalence group. The usual call to **AddInvtInfo** will thus set the parameter `detEquivInArbvars` to be the determining equations of the equivalence group, as found in the sub-field `detEquiv:-arbvarsAction` of a pdeRecord.
- **AddInvtInfo** returns an updated rif-output table.
- **AddInvtInfo** uses the service procedure newProlongation to test invariance of each pivot. It inserts two new entries: `InvtCase` and `InvtPivots` into the rif-output. These new entries are similar to entries `Case` and `Pivots` but `InvtCase` and `InvtPivots` have an additional boolean label, 'true' for invariant or 'false' for non-invariant.
- The updated rif-output can be displayed graphically by CasePlot.

- **AddInvtInfo** does not itself do symmetry classification, it only labels the case splits of an existing classification tree. The related procedure SymmetricRifsimp performs symmetry classification by seeking invariant case splits.

Examples

We use 1+1 nonlinear heat equation as an example. We first need to produce a rif-output and `detEquiv` field by create a `pdeRecord`, complete it, and call `rifsimp`.

```
> with(SymmetryClassification);
```

[*AddInvtInfo, CasePlot, ModuleLoad, SymmetricRifsimp, classifySymmetry, detEqsForEquiv, detEqsForSymm, newPDESys, newProlongation*] **(2.1)**

Typesetting setup

```
> with(Typesetting): Settings(userep=true):
> interface(typesetting=extended):
> Suppress({u(x,t), q(x,t), K(u)});
```

Create a `pdeRecord` for Nonlinear Heat Equation

```
> DEs := [diff(u(x,t),t) + diff(q(x,t),x) = 0, q(x,t)
  = -K(u(x,t))*diff(u(x,t),x)];
```

$$DEs := \left[u_t + q_x = 0,\ q = -K(u)\,u_x\right] \quad \textbf{(2.2)}$$

```
> NLHeat:=newPDESys(DEs, constraint=[K(u)<>0]);
```

$$NLHeat := \left[u_t + q_x = 0,\ q = -K(u)\,u_x\right] \ \&\text{where}\ [K \neq 0] \quad \textbf{(2.3)}$$

Assign fields `detSymm` and `detEquiv` by calling...

```
> detEqsForSymm(NLHeat, infinitesimals = [[x,xi], [t,
  tau], [u, eta], [q, phi]]):
> detEqsForEquiv(NLHeat, infinitesimals = [[x,xi], [t,
  tau], [u, eta], [q, phi], [K, kappa]]):
```

Now the `pdeRecord` is complete, we call `rifsimp` to classify symmetries

```
> rifSys:= DEtools[rifsimp]([op(NLHeat:-detSymm:-sys),
  op(NLHeat:-constraint:-sys)], [[eta, phi, xi, tau],
  [K]], casesplit):
```

The rif-output is a table containing cases (which are represented as tables as well). In Case 2, there are three entries: `Case`, `Solved` and `Pivots`.

```
> indices(rifSys[2]);
```

[*Case*], [*Solved*], [*Pivots*] **(2.4)**

For example the `Case` field of case 2 is:

```
> rifSys[2][Case];
```

$$\left[\left[\frac{dK}{du} \neq 0,\ \frac{\partial}{\partial q}\,\phi(x,t,q,u)\right],\left[4\,K\,\frac{d^2K}{du^2} - 7\,\frac{dK}{du}^2 \neq 0,\ \frac{\partial}{\partial u}\,\phi(x,t,q,u)\right]\right], \quad \textbf{(2.5)}$$

$$\left[4\,K\,\frac{d^2K}{du^2} - 3\,\frac{dK}{du}^2 \neq 0,\ \frac{\partial}{\partial t}\,\eta(x,t,q,u)\right],\left[K\,\frac{dK}{du}\,\frac{d^3K}{du^3} - 2\,K\,\frac{d^2K}{du^2}^2 + \frac{dK}{du}^2\,\frac{d^2K}{du^2} = 0,\ \eta(x,t,q,u)\right]\Bigg]$$

Calling **AddInvtInfo** for adding invariant properties on these splitting condition.

```
> rifSys:=AddInvtInfo(rifSys, NLHeat:-detEquiv:-
  arbvarsAction, NLHeat:-detSymm:-arbvars):
```

And now in Case 2, there two new entries: `InvtCase` and `InvtPivots`:

```
> indices(rifSys[2]);
```

$$[InvtCase],\ [Case],\ [Solved],\ [InvtPivots],\ [Pivots] \tag{2.6}$$

These new entries are the same as before except that an extra item 'true' or 'false' has been inserted as a label of whether or not it is invariant under the equivalence group:

```
> rifSys[2][InvtCase];
```

$$\left[\left[\frac{dK}{du} \neq 0,\ \frac{\partial}{\partial q}\,\phi(x,t,q,u),\ true\right],\left[4\,K\,\frac{d^2K}{du^2} - 7\,\frac{dK}{du}^2 \neq 0,\ \frac{\partial}{\partial u}\,\phi(x,t,q,u),\ true\right],\left[4\,K\,\frac{d^2K}{du^2} - 3\,\frac{dK}{du}^2 \neq 0,\ \frac{\partial}{\partial t}\,\eta(x,t,q,u),\ true\right],\left[K\,\frac{dK}{du}\,\frac{d^3K}{du^3} - 2\,K\,\frac{d^2K}{du^2}^2 + \frac{dK}{du}^2\,\frac{d^2K}{du^2} = 0,\ \eta(x,t,q,u),\ true\right]\right] \tag{2.7}$$

The meaning is that for instance $\frac{dK}{du} \neq 0$ is invariant under the equivalence group.

Here all the splitting conditions (first items) are tested as invariant is 'true'. This is the best possible outcome.

See Also

SymmetryClassification, SymmetricRifsimp, DEtools[rifsimp], DEtools[rifsimp], output, pdeRecord, newPDEsys, detEqsForSymm, detEqsForEquiv, CasePlot

SymmetryClassification[SymmetricRifsimp] - classify symmetries using the rif algorithm

Calling Sequence

SymmetricRifsimp(**system, options**)
SymmetricRifsimp(**system, vars, options**)

Parameters

`system` - list or set of polynomially nonlinear PDEs or ODEs (may contain inequations)
`vars` - (optional) list of the dependent variables
`options` - (optional) sequence of options to control the behavior of `rifsimp`

Description

- **SymmetricRifsimp** is a modified version of rifsimp. All functionalities in `rifsimp` can be applied to **SymmetricRifsimp**, but **SymmetricRifsimp** provides an additional option which is to select invariant splitting conditions if wanted. The invariant splitting condition here means the splitting DEs are invariant under action of a group.
- **SymmetricRifsimp** is designed to be applied to symmetry classification. In this case, the input `system` will be a system of determining equations for the point symmetries of some DEs, and the group is the equivalence group of these DEs. (Case splittings in a symmetry classification should be invariant under the action of the equivalence group.)
- **SymmetricRifsimp** returns a rif-output table, as described in rifsimp,output. If invariant pivot selection is requested, the rif-output has additional fields inserted in the table.
- To have invariant option enabled, the procedure **SymmetricRifsimp** requires another option, `pivselect`, in the following form

```
pivselect = ['invariant', detEquivInArbvars, arbvars]
```

  where `detEquivInArbvars` is a sub-field `detEquiv:-arbvarsAction` from a pdeRecord which contains information about the determining equations for the equivalence group, and `arbvars` is a list of arbitrary elements.
- **SymmetricRifsimp** works exactly same as rifsimp except that when the `invariant` option is enabled, it tries to select invariant case splits during classification. The preference of choice in invariant case splits are assumed to be `smalleq` [see rifsimp,cases for detail]. That is, where rifsimp would choose the smallest equation containing a case split, **SymmetricRifsimp** chooses the smallest equation containing an *invariant* case split. If no invariant case splits are available,

SymmetricRifsimp will go back to choosing the smallest equation (i.e. same choice as `rifsimp`).

- With invariant option enable, the returned rif-output has two new entries appended: `InvtCase` and `InvtPivots`. Both new entries are similar to entries `Case` and `Pivots` but `InvtCase` and `InvtPivots` have 'true' as invariant or 'false' as non-invariant appended.
- The rif-output can be displayed graphically by CasePlot.

Examples

We use 1+1 nonlinear heat equation as an example. We first need to produce a rif-ouput and `detEquiv` field by creating a `pdeRecord`, complete it, and call **SymmetricRifsimp** with `invariant` option enabled.

```
> with(SymmetryClassification);
```

[*AddInvtInfo, CasePlot, ModuleLoad, SymmetricRifsimp, classifySymmetry, detEqsForEquiv, detEqsForSymm, newPDESys, newProlongation*] **(2.1)**

Typesetting setup

```
> with(Typesetting):
  Settings(userep=true):
> interface(typesetting=extended):
> Suppress({u(x,t), q(x,t), K(u)});
```

Create a `pdeRecord` for Nonlinear Heat Equation

```
> DEs := [diff(u(x,t),t) + diff(q(x,t),x) = 0, q(x,t)
  = -K(u(x,t))*diff(u(x,t),x)];
```

$$DEs := [u_t + q_x = 0,\ q = -K(u)\,u_x] \quad \textbf{(2.2)}$$

```
> NLHeat:=newPDESys(DEs, constraint=[K(u)<>0]);
```

$$NLHeat := [u_t + q_x = 0,\ q = -K(u)\,u_x]\ \&\text{where}\ [K \neq 0] \quad \textbf{(2.3)}$$

```
> Suppress({xi(x), tau(t), eta(u), phi(q), kappa(K)});
```

Assign fields `detSymm` and `detEquiv` by calling..

```
> detEqsForSymm(NLHeat, infinitesimals = [[x,xi], [t,
  tau], [u, eta], [q, phi]]):
> detEqsForEquiv(NLHeat, infinitesimals = [[x,xi], [t,
  tau], [u, eta], [q, phi], [K, kappa]]):
```

Now the `pdeRecord` is complete, we call `rifsimp` to classify symmetries with `invariant` option enabled

```
> rifSys:=SymmetricRifsimp([op(NLHeat:-detSymm:-sys),
  op(NLHeat:-constraint:-sys)], [[eta, phi, xi, tau],
  [K]], casesplit, pivselect=['invariant', NLHeat:-
```

```
> detEquiv:-arbvarsAction, NLHeat:-detSymm:-arbvars]):
```

The rif-output is a table contains cases (which represent as tables as well). The rif-output has included two new entries: `InvtCase` and `InvtPivots`.

```
> indices(rifSys[2]);
```

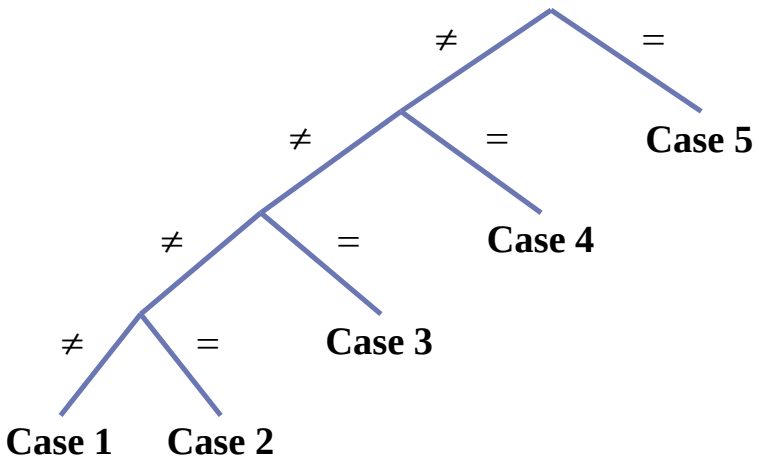
$$[Solved], [Pivots], [InvtCase], [Case], [InvtPivots] \tag{2.4}$$

```
> rifSys[2][InvtCase];
```

$$\left[\left[\frac{dK}{du} \neq 0, \frac{\partial}{\partial q}\,\phi(x,t,q,u), true\right], \left[-7\left(\frac{dK}{du}\right)^2 + 4\frac{d^2K}{du^2}K \neq 0, \frac{\partial}{\partial u}\,\phi(x,t,q,u), true\right], \left[\frac{d^2K}{du^2}K - \left(\frac{dK}{du}\right)^2 \neq 0, \frac{\partial}{\partial t}\,\eta(x,t,q,u), true\right], \left[-K\frac{dK}{du}\frac{d^3K}{du^3} + 2K\left(\frac{d^2K}{du^2}\right)^2 - \left(\frac{dK}{du}\right)^2\frac{d^2K}{du^2} = 0, \eta(x,t,q,u), true\right]\right] \tag{2.5}$$

Call `CasePlot` to display rif-output graphically..

```
> CasePlot(rifSys);
```



See Also

SymmetryClassification, DEtools[rifsimp], newPDESys, pdeRecord, detEqsForSymm, detEqsForEquiv, CasePlot

SymmetryClassification[CasePlot] - display rif-output graphically as a case tree

Calling Sequence

CasePlot(**rifsys**)
CasePlot(**rifsys**, **options**)

Parameters

`rifsys` - output from **rifsimp** or **SymmetricRifsimp** with **casesplit** active
`options` - (optional) the form **option=value** where **option** is one of **returnTree**, **vars**, **format,** or animate; specify options for the **CasePlot** command

Description

- The **CasePlot** command plots a case tree representation of a system of PDEs as returned by rifsimp or SymmetricRifsimp with option casesplit active.
- **CasePlot** performs the same function as the DEtools,caseplot command, but has some additional functionality specific to symmetry classification.
- The format of the resulting tree can be controlled by options. The **options** argument can contain one or more of the following equations.

returnTree = '*name*'
As well as plotting the tree, a representation of the tree as a table is transcribed into the variable with the given name. This option also causes information about the case splits to be drawn at the nodes of the tree. Additional information such as pivot expressions can be extracted from the table.

vars = [var1, ...]
Work out the dimension of the solution space of the rif-forms with respect to the given list of variables [var1, ...], and add this information to the tree plot.

format = *string*
Format the tree plot as specified. Possible format strings are `"standard"`, `"compact"` and `"auto"`. In `"compact"` format, the text on the tree is abbreviated as compared with `"standard"` format. In `"auto"` format, **CasePlot** switches between `"standard"` and `"compact"` automatically according to the size and complexity of the tree. The default is `"auto"`.

animate = true/false
If **animate** is true, return an animation instead of a plot. This is mostly for use within the Maplet interface to rifsimp.

Additional options can be specified: these are passed through to the plot command, which does the actual drawing.

• The following information is annotated on the tree:

– Leaves of the tree are labelled as "Case 1", "Case 2" etc.

– Any branches of the tree that were not followed by rifsimp are labelled as " (skipped)" (for format="standard") or "--" (for format="compact").
Reasons why a branch might not be followed include:
the branch was found to be inconsistent;
the branch was found to have a solution space dimension lower than specified by rifsimp option `mindim=`.
the branch took too much execution time as specified by rifsimp options

– If option `returnTree=` has been specified, then splitting nodes in the tree are labelled with 'p1', 'p2' etc. These expressions are set to be nonzero on the left subtree and zero on the right subtree. Expressions for 'p1', 'p2' etc. can be recovered from the table returned via the `returnTree` value. (See Example 1 below.)

– Branchings are labelled with $\neq$ or = (left subtrees are $\neq$, right subtrees are =).

– If option `vars=` has been specified, then cases (leaves) are labelled with the dimension of their solution space with respect to the given variables.

– For a symmetry classification made using classifySymmetry (or the lower level procedures SymmetricRifsimp and AddInvtInfo), **CasePlot** can indicate the cases and branchings that are known to be invariant under the equivalence group -- distinguished by colour and font from those that are not invariant.

– A title can be added to the case tree plot using the `title=` option (which is passed to plot).

Examples

```
> with(SymmetryClassification);
```

[*AddInvtInfo, CasePlot, ModuleLoad, SymmetricRifsimp, classifySymmetry, detEqsForEquiv, detEqsForSymm, newPDESys, newProlongation*] **(2.1)**

```
> interface(typesetting=extended):
> Typesetting:-Settings(userep=true):
```

Example 1: CasePlot of rifsimp output

```
> Typesetting:-Suppress({K(u), xi(x,t,u), tau(x,t,u)
  , eta(x,t,u), chi(x,t,u,q)});
```

Here is an overdetermined DEs system (also with an inequality constraint). These are the determining equations for point symmetries of the nonlinear heat equation written as a system.
The dependent variables ξ, τ, η, χ occur linearly, the variable K occurs nonlinearly.

```
> DEs := [K(u)<>0, -diff(tau(x, t, u), x)*K(u)+q*
```

```
diff(tau(x, t, u), u) = 0, diff(tau(x, t, u), x)*K
(u)-q*diff(tau(x, t, u), u) = 0, diff(eta(x, t, u)
, t)*K(u)+diff(chi(x, t, u, q), x)*K(u)+q*diff(xi
(x, t, u), t)-q*diff(chi(x, t, u, q), u) = 0, -
diff(xi(x, t, u), x)-diff(eta(x, t, u), u)+diff
(chi(x, t, u, q), q)+diff(tau(x, t, u), t) = 0,
diff(eta(x, t, u), x)*K(u)^2-q*diff(eta(x, t, u),
u)*K(u)+q*diff(xi(x, t, u), x)*K(u)-q^2*diff(xi(x,
t, u), u)+chi(x, t, u, q)*K(u)-eta(x, t, u)*q*diff
(K(u), u) = 0, diff(tau(x, t, u), u) = 0];
```

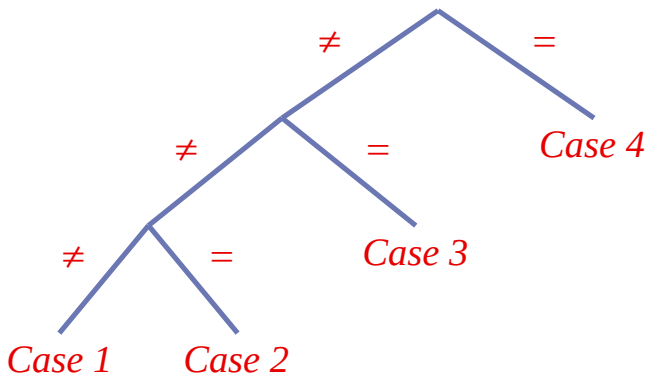
$$DEs := \left[K \neq 0, -\tau_x K + q\,\tau_u = 0, \tau_x K - q\,\tau_u = 0, \eta_t K + \chi_x K + q\,\xi_t - q\,\chi_u = 0, -\xi_x - \eta_u + \chi_q + \tau_t = 0, \eta_x K^2 - q\,\eta_u K + q\,\xi_x K - q^2\,\xi_u + \chi K - \eta\,q\,\frac{dK}{du} = 0, \tau_u = 0\right] \quad \textbf{(2.1.1)}$$

Use `rifsimp` to casesplit...

```
> rifDEs := DEtools[rifsimp](DEs, [[chi],[eta],
  [tau,xi], [K]], indep=[q,u,t,x], casesplit):
```

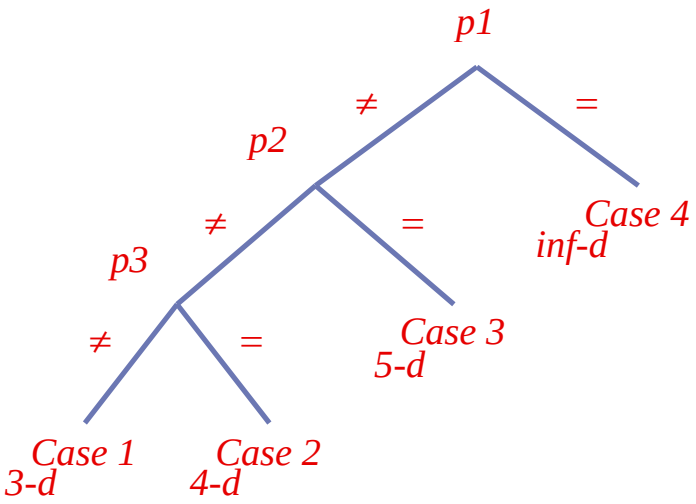
... then plot the result.
With no options, the tree is not very informative...

```
> CasePlot(rifDEs);
```



Repeat the plot, but ask to see the dimension of the solution space with respect to variables ξ, τ, η, χ .
Also ask for info about the tree to be stored in a name...

```
> CasePlot(rifDEs, returnTree='T', vars=[xi,tau,eta,
  chi]);
```



All the pivots and cases are coloured as though non-invariant. This is because invariance information was not requested during tree construction, so none of the case splits are known to be invariant.

The 'pivots' that rif split on are now contained in the tree...

```
> for j to T["PivotCount"] do
>   p||j = T["Pivots"][j][1]
> end do;
```

$$p1 = \frac{dK}{du}$$

$$p2 = -7 \left(\frac{dK}{du}\right)^2 + 4\,K \frac{d^2K}{du^2}$$

$$p3 = \frac{dK}{du}^2 \frac{d^2K}{du^2} + \frac{dK}{du} K \frac{d^3K}{du^3} - 2\,K \frac{d^2K}{du^2}^2 \qquad \textbf{(2.1.2)}$$

Example 2: CasePlot of a symmetry classification

CasePlot is designed to interact well with other functions from the SymmetryClassification package.
In particular it is able to indicate invariance information on the tree if this has been calculated.

```
> Typesetting:-Suppress({u(x,t), q(x,t), B(u), K(u)}
  );
```

Nonlinear heat equation, written as a system...

```
> RichardsEq := [diff(u(x,t),t) + diff(q(x,t),x) =
  0, q(x,t) = -B(u(x,t))*diff(u(x,t),x)+K(u(x,t))];
```

$$RichardsEq := \left[u_t + q_x = 0,\ q = -B(u)\,u_x + K(u)\right] \quad \textbf{(2.2.1)}$$

...store in a pdeRecord module...

```
> RichardsModule := newPDESys(RichardsEq,
  constraint=[B(u)<>0]);
```

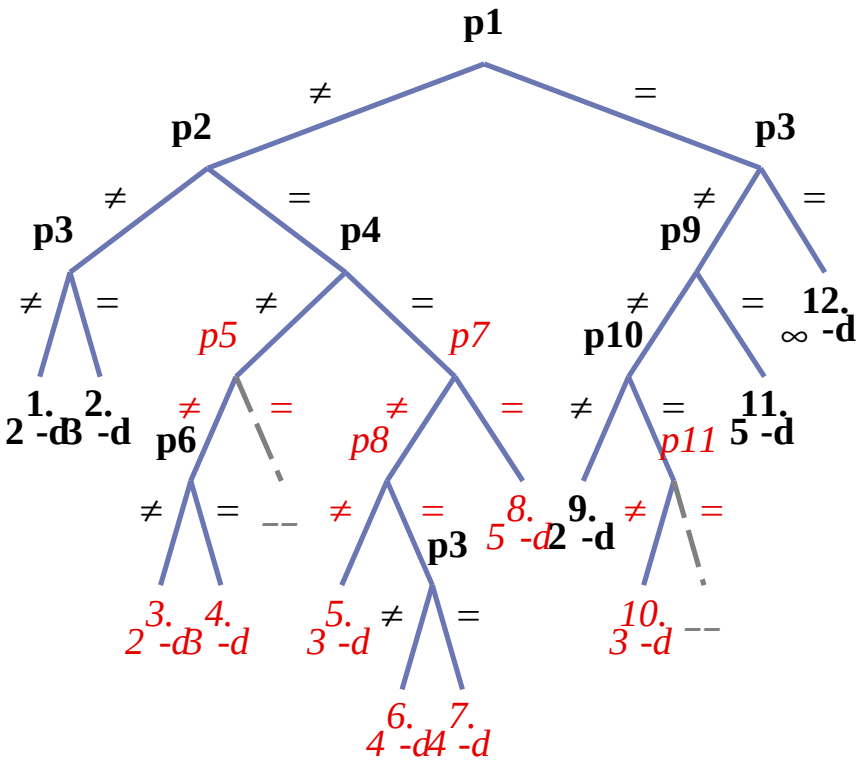
$$RichardsModule := \left[u_t + q_x = 0,\ q = -B(u)\,u_x + K(u)\right] \text{ \&where } [B \neq 0] \quad \textbf{(2.2.2)}$$

...and use classifySymmetry to set up the determining equations for symmetries and split them into cases...

```
> rifDetEqs := classifySymmetry(
    RichardsModule,
    infinitesimals=[[x,xi(x,t,u)], [t,tau(x,t,u)],
  [u,eta(x,t,u)], [q,chi(x,t,u,q)], [B, beta], [K,
  kappa]],
    pivselectOption='invariant'):
```

...finally showing a summary of the result graphically using **CasePlot**...

```
> CasePlot(rifDetEqs, vars=[xi,tau,eta,chi],
  returnTree='T', format="compact");
```



The upright **black bold** pivots are known to be invariant, while the *red italic* pivots are not invariant.
The upright **black bold** Case information at the leaves indicates that all branchings between root and leaf were invariant, so that these subcases have been split off invariantly. The *red italic* Case information has at least one pivot between root and leaf that has not tested as invariant, so that the case has not been split off invariantly.

In fact this tree is a mess -- there are many non-invariant (*red italic*) branches. This indicates that more control should be exerted over the classification e.g. by controlling the ranking better.

See Also

classifySymmetry, SymmetricRifsimp, rifsimp, rifsimp/cases, caseplot, SymmetryClassification

SymmetryClassification[classifySymmetry] - front-end procedure for symmetry classification package

Calling Sequence

classifySymmetry(**DERecord, options**)

Parameters

`DERecord` - a `pdeRecord` module data structure
`options` - (optional) sequence of options for classifySymmetry

Description

- **classifySymmetry** is a front-end procedure for the SymmetryClassification package. It classifies symmetries of a given DEs system (stored in a pdeRecord).
- **classifySymmetry** requires a pdeRecord data structure which is created via the constructor newPDESys. The procedure first complete the pdeRecord if needed, then it calls SymmetricRifsimp with given ranking and pivselect options (if none specified then it goes to default). It returns a rif-output [see rifsimp,output].
- **classifySymmetry** may call the following procedures: detEqsForSymm (deriving the determining equations for point symmetries), detEqsForEquiv (deriving the determining equations for the equivalence group), and SymmetricRifsimp.
- The rif-output can then be displayed graphically by a call to CasePlot.
- List of options possibly needed for classification:
- **infinitesimals = [...]**
  This option is for user to specify their own infinitesimals with corresponding variables. The value can be a list or set of the form

```
[[var1, infinitesimal1], [var2, infinitesimal2], ...
]
```

  where `var1, var2, ...` are the names of variables, and `infinitesimal1, infinitesimal2, ...` are the corresponding infinitesimals. The infinitesimals can be names or functions. **classifySymmetry** will choose sensible defaults if none are specified. Note that `infinitesimals =` is used for both detEqsForSymm and detEqsForEquiv, so the variables `var1, var2,...` can include independent & dependent variables and arbitrary elements of the DEs system.
- **infinitesimalRanking = [...]**
  This option is for specifying the infinitesimal ranking, which is used for SymmetricRifsimp. It should be a list of infinitesimal names, or a list of lists of such names (as in the vars option of rifsimp [see rifsimp,options]). These infinitesimals correspond to the independent and dependent variables of the DEs

(not arbitrary elements). For this option to make sense, infinitesimal names must have been specified by `infinitesimals=` option.

- **arbvarsRanking = [...]**
  This option is for specifying the ranking for arbitrary elements, which is used for SymmetricRifsimp. It should be a list of infinitesimal names, or a list of lists of such names (as in the `vars` parameter of rifsimp [see rifsimp,options]).

- **pivselectOption = ...**
  This option is to control the pivot selection strategy used by SymmetricRifsimp. Any of the values for the `pivselect=` option of rifsimp can be used [see rifsimp, cases for details]. To request invariant pivot selection, use `pivselectOption='invariant'`. This will cause the symmetry classification to try to find case splits that are invariant under the action of the equivalence group.

Examples

```
> with(SymmetryClassification);
```

[*AddInvtInfo, CasePlot, ModuleLoad, SymmetricRifsimp, classifySymmetry, detEqsForEquiv, detEqsForSymm, newPDESys, newProlongation*] **(2.1)**

Typesetting setup

```
> with(Typesetting): Settings(userep=true, useprime=
  false, usedot=false):
> interface(typesetting=extended):
> Suppress({u(x,t), q(x,t), K(u)});
```

Example 1: Nonlinear heat equation, using default options

Create a pdeRecord for Nonlinear Diffusion Equation

```
> DEs := [diff(u(x,t),t) + diff(q(x,t),x) = 0, q(x,t)
  = -K(u(x,t))*diff(u(x,t),x)];
```

$$DEs := [u_t + q_x = 0,\ q = -K(u)\,u_x] \quad \textbf{(2.2)}$$

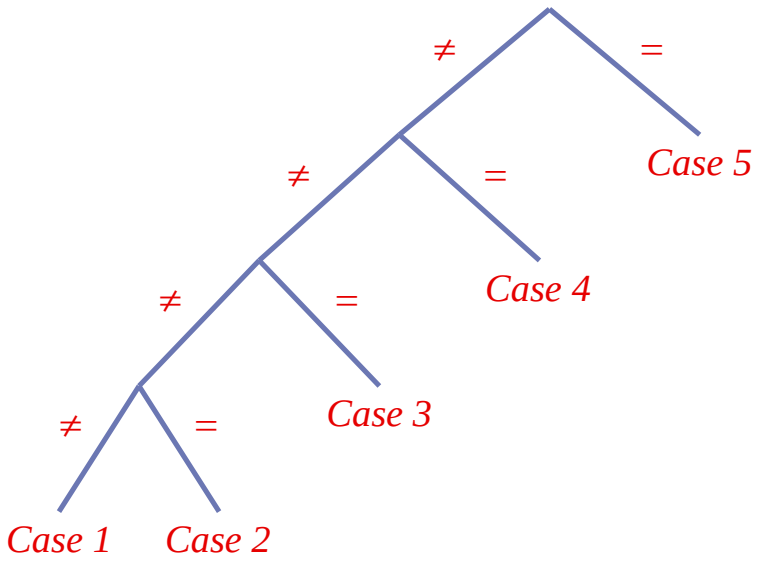
```
> NLHeat := newPDESys(DEs, constraint=[K(u)<>0]);
```

$$NLHeat := [u_t + q_x = 0,\ q = -K(u)\,u_x]\ \&\text{where}\ [K \neq 0] \quad \textbf{(2.3)}$$

Classify symmetries by calling classifySymmetry

Because invariance checking is not requested, the symmetry classification is as would be done by `DEtools[rifsimp]`, no invariance information is included in the rif-output.

```
> rifSys:= classifySymmetry(NLHeat):
> CasePlot(rifSys);
```



Example 2: Nonlinear heat equation, specifying options

```
> NLHeat := newPDESys(DEs, constraint=[K(u)<>0]);
```

$$NLHeat := [u_t + q_x = 0,\ q = -K(u)\,u_x] \ \&\text{where}\ [K \neq 0] \qquad \textbf{(2.4)}$$

With `pivselectOption='invariant'`, the symmetry classification is labelled with invariance information.

```
> rifSys:= classifySymmetry(NLHeat,
  infinitesimals=[[x, xi], [t, tau], [u, eta], [q,
  chi], [K,kappa]],
  infinitesimalRanking=[eta, chi, xi, tau],
  arbvarsRanking=[K],
  pivselectOption='invariant' ):
```

The `pdeRecord` now contains information on determining equations for point symmetries...

```
> Suppress({xi(x,t,q,u), tau(x,t,q,u), eta(x,t,q,u),
  chi(x,t,q,u)});
> print(NLHeat:-detSymm);
```

(2.5)

$$\left[\frac{-\eta_q K-\tau_x K+q\left(\tau_u-\xi_q\right)}{K}=0,\ \frac{\eta_q K-\tau_x K+q\left(\tau_u+\xi_q\right)}{K}=0,\ -\eta_u-\xi_x\right. \tag{2.5}$$

$$+\tau_t+\chi_q=0,\ \tau_q=0,\ \frac{\eta_t K+q\,\xi_t-q\,\chi_u+\chi_x K}{K}=0,\ -\tau_u-\xi_q=0,$$

$$\left.\frac{-\eta_x K^2+q\,\eta_u K-\chi K-q\,\xi_x K+q^2\,\xi_u+\eta\,q\,\frac{dK}{du}}{K^2}=0\right] \text{ \&where } [K \neq 0]$$

...and determining equations for equivalence group...

```
> print(NLHeat:-detEquiv);
```

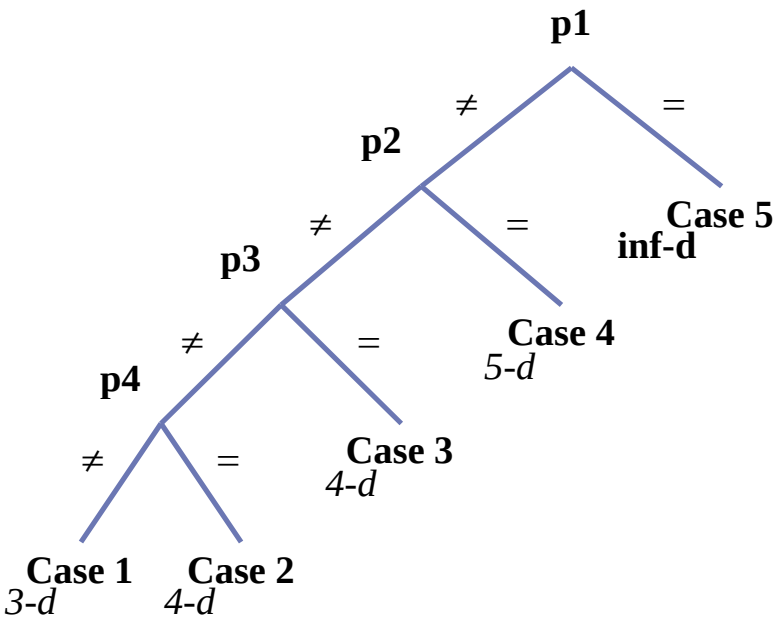
$$\left[\frac{d}{dx}\,\xi(x)=\frac{q\,\frac{d}{du}\,\eta(u)\,K+q\,\kappa(K)-\chi(q)\,K}{q\,K},\ \frac{d}{dt}\,\tau(t)\right. \tag{2.6}$$

$$=\frac{2\,q\,\frac{d}{du}\,\eta(u)\,K+q\,\kappa(K)-2\,\chi(q)\,K}{q\,K},\ \frac{d}{dq}\,\chi(q)=\frac{\chi(q)}{q},\ \frac{d}{dK}$$

$$\left.\kappa(K)=\frac{\kappa(K)}{K},\ \frac{d^2}{du^2}\,\eta(u)=0\right]$$

A call to `CasePlot` shows the symmetry classification with invariance information by colour coding.

```
> CasePlot(rifSys, vars=[xi,tau,eta,chi], returnTree=
  'T');
```



Inspect the pivots that the tree has split on...

```
> for j to T["PivotCount"] do
>   p||j=T["Pivots"][j];
> end do;
```

$$p1 = \left[\frac{dK}{du}, true \right]$$

$$p2 = \left[-7 \left(\frac{dK}{du} \right)^2 + 4 \frac{d^2K}{du^2} K, true \right]$$

$$p3 = \left[\frac{d^2K}{du^2} K - \left(\frac{dK}{du} \right)^2, true \right]$$

$$p4 = \left[-K \frac{dK}{du} \frac{d^3K}{du^3} + 2K \left(\frac{d^2K}{du^2} \right)^2 - \left(\frac{dK}{du} \right)^2 \frac{d^2K}{du^2}, true \right] \quad \textbf{(2.7)}$$

See Also

DeterminingPDE, rifsimp, rifsimp[cases], detEqsForSymm, newPDESys, SymmetryClassification

Appendix B

Published work arising from the thesis

In the following pages is a copy of the paper

> Lisle, I.G. and S.-L. T. Huang. 2009. 'Algorithmic symmetry classification with invariance.' *J. Engineering Mathematics* DOI 10.1007/s10665-009-9327-6

presented at the conference *Similarity: Generalization and applications*, University of British Columbia, Canada, 11th–15th August 2008, and published in a special issue of *Journal of Engineering Mathematics*.

UNIVERSITY OF CANBERRA
AUSTRALIA'S CAPITAL UNIVERSITY

Appendix B

This appendix has been removed due to copyright restrictions.

This appendix is available as:

Lisle, I.G. and S.-L. T. Huang. 2009. 'Algorithmic symmetry classification with invariance.' *Journal of Engineering Mathematics*. Vol. 66, no. 1-3, pp. 201-216

Links to this chapter:

Print	http://webpac.canberra.edu.au/record=b1522693~S4
Online subscribed content (UC community)	http://ezproxy.canberra.edu.au/login?url=http://www.springerlink.com/content/y567165x2502g573/
Online general public	http://www.springerlink.com/content/y567165x2502g573/
DOI	10.1007/s10665-009-9327-6

Abstract

Symmetry classification for a system of differential equations can be achieved algorithmically by applying a differential reduction and completion algorithm to the infinitesimal determining equations of the system. The branches of the classification should be invariant under the action of the equivalence group. We show that such invariance can be tested algorithmically knowing only the determining equations of the equivalence group. The method relies on computing the prolongation of a group operator reduced modulo these determining equations. The method is implemented in Maple: a novel pivot selection strategy is able to guide the *rifsimp* command towards more favourable branchings.