# Fast Differential Elimination in C: The CDiffElim Environment

Allan D. Wittkopf [a,3] and Gregory J. Reid [b,1,2,3]

[a] *Centre for Experimental and Constructive Mathematics, Simon Fraser University, Burnaby, B.C. Canada V5A 1S6*

[b] *University of Western Ontario, Department of Applied Mathematics, London, Ontario, Canada N6A 5B7*

**Abstract**

We introduce the *CDiffElim* environment, written in C, and an algorithm developed in this environment for simplifying systems of overdetermined partial differential equations by using differentiation and elimination.

This environment has strategies for addressing difficulties encountered in differential elimination algorithms, such as exhaustion of computer memory due to intermediate expression swell, and failure to complete due to the massive number of calculations involved. These strategies include low-level memory management strategies and data representations that are tailored for efficient differential elimination algorithms. These strategies, which are coded in a low-level C implementation, seem much more difficult to implement in high-level general purpose computer algebra systems.

A differential elimination algorithm written in this environment is applied to the determination of symmetry properties of classes of $n+1$–dimensional coupled nonlinear partial differential equations of form $i\mathbf{u}_t+\nabla^2\mathbf{u}+\left(a(t)|\mathbf{x}|^2+\mathbf{b}(t)\cdot\mathbf{x}+c(t)+d|\mathbf{u}|^{\frac{4}{n}}\right)\mathbf{u}=\mathbf{0}$, where $\mathbf{u}$ is an $m$-component vector-valued function. The resulting systems of differential equations for the symmetries have been made available on the web, to be used as benchmark systems for other researchers. The new differential elimination algorithm in C, runs on the test suite an average of 400 times faster than our *RifSimp* algorithm in Maple.

New algorithms, including an enhanced GCD algorithm, and a hybrid symbolic-numeric differential elimination algorithm, are also described.

*Key words:* Differential Elimination; Differential Gröbner Basis; Symbolic Computation; `CPC Program Library Classification(s)`

## PROGRAM SUMMARY

*Title of program:* DiffElim

*Catalogue number:* (supplied by Elsevier)

*Program obtainable from:* (supplied by Elsevier)

*Licensing provisions:* Binary: None; Source: Commercial

*Operating systems under which the program has been tested:* Linux/ix86

*Programming language used:* C

*Memory required to execute with typical data:* 16 Megabytes (Note that some of the more difficult systems used as benchmarks require 128M)

*Keywords:* Differential Elimination, Symbolic Manipulation

*Nature of physical problem*
Differential Elimination of linear and polynomially nonlinear systems of ODE/PDE with coefficients in $\mathbb{Z}[x1, ..., xn]$, where $x1, ..., xn$ are the independent variables.

*Method of solution*
Application of an elimination process bearing some similarity to both Gröbner Basis methods, and Gauss Elimination, but generalized to systems of PDE.

*Restrictions on the complexity of the problem*
The algorithm is restricted to systems of PDE that are fully polynomial in their dependent and independent variables. No special functions (such as sin,cos,exp,ln), fractional exponents, or rational numbers/expressions are allowed.
The implementation is currently under development. All results used as benchmarks in the paper can be obtained from the current program, but many of the features discussed (such as the program running in Rif-mode) are incomplete.
The current implementation only uses full reduction, for which some problems fail to terminate (depending on the degree of nonlinearity).
A later release of the program will have these restrictions removed, but it will still be possible to run the program in full reduction mode, as partial results of the computations running with full reduction often yield information that is useful in simplification of the input system

*Typical running time*
The presented implementation has a counterpart with some similarity implemented

[1] Support from the Natural Sciences and Engineering Research Council of Canada is gratefully acknowledged.
[2] Helpful discussions with with Ilias Kotsireas and Greg Mayer on Hybrid Symbolic-Numeric Methods are gratefully acknowledged.
[3] Discussions with Mark Hickman concerning the use of his Maple package Symmetry are also acknowledged.

in the symbolic programming language Maple (*RifSimp*). For the presented benchmark systems, the new implementation/algorithm has a running time of (on average) $\frac{1}{400}$ of that of *RifSimp*. Note that the more difficult results of the paper require at most 5 minutes on a PII 333MHz machine.

*Unusual features of the program*
The program is only capable of running on a Linux/x86 platform at this time.

**LONG WRITE-UP**

## 1 Introduction

Most differential elimination algorithms can be regarded as a differential generalization of the Gröbner Basis algorithm for canonically simplifying overdetermined polynomial systems.

Overdetermined systems arise in applications such as Lagrangian mechanical systems with constraints. There, differential elimination methods enable the determination of hidden constraints, necessary for the determination of consistent initial conditions. They also enable the reduction of the so-called *differential index* which has been cited as a measure of the numerical difficulty of solving such systems [6]. Such differential-algebraic systems also arise in industrial applications such as simulation of chemical processes [25]. Complicated and overdetermined systems of partial differential equations also arise in geometrical applications such as in Cartan's Method of Equivalence [27,12], the determination of symmetries and conservation laws, and invariant solutions of partial differential equations [3,26].

Recently algorithms have been developed for simplifying overdetermined systems of PDE using differentiation and elimination. Maple has implementations of the Rosenfeld-Gröbner algorithm [5], Mansfield's Differential Gröbner Basis algorithm [22] and our *RifSimp* algorithm [33] which is available at our web site for Maple Release 5, and is distributed as part of Maple 6. Reduce has Wolf's program Crack [48], Schwarz's programs [38], and an implementation of the Cartan Exterior Differential Systems approach by Hartley and Tucker [17]. Axiom has an implementation of the Cartan-Kuranishi Algorithm [40]. In addition to these implementations, much progress has been made in deriving theoretical results for their output, from which properties of the input systems can be determined. These results include, for example, associated existence and uniqueness theorems [37], the decoupling to subsystems, if such subsystems exist, and the answer to algebraic questions, such as if one expression is a consequence of another (the ideal membership problem [5]).

Despite the above advances, serious obstacles exist to the more widespread application of such packages. Though these algorithms theoretically terminate with the desired output, very often individual computations either take an unreasonable time to complete, or require more memory than is available. The *RifSimp* package implemented in Maple by one of us (ADW), despite considerable efforts, still suffers from the above difficulties.

This paper describes a new environment, which has strategies to address the above-mentioned difficulties. The implementation in C allows storage of the system, and all related information needed for the algorithm in a more compact way than currently seems available in general purpose computer algebra systems. The core operations required for differential elimination are coded in a way tailored to differential elimination problems. For example, special ordered arrays are used to store equations processed by the algorithms. These ordered arrays are designed around the rankings of derivatives used in differential elimination algorithms, and allow efficient storage and fast determination of quantities needed by the algorithms. A hybrid symbolic-numeric differential elimination method is also described.

To test the performance and establish the feasibility of the *CDiffElim* environment, we have written a differential elimination algorithm (the *DiffElim* algorithm) in that environment.

It is applied to the determination of symmetry properties of the $n{+}1$–dimensional coupled nonlinear vector Schrödinger system (VNLS) of the form:

$$i\mathbf{u}_t + \nabla^2\mathbf{u} + F(\mathbf{x}, t, \mathbf{u}\cdot\mathbf{u}^*)\mathbf{u} = \mathbf{0}. \tag{1}$$

Here $\mathbf{u}$ is an $m$-component vector-valued function and $\nabla^2$ is the $n$-space Laplacian. Many special cases of the above system arise in applications and have been extensively studied. The case $m = 1$, with cubic nonlinearity $Fu = (uu^*)u$, is fundamental in nonlinear optics where it describes the interaction of electromagnetic waves with different polarizations propagating in nonlinear media such as isotropic plasma [2,43]. For $m > 1$, with $F = \mathbf{u}\cdot\mathbf{u}^*$, it arises in the nonlinear interaction of $m$ waves in deep fluids [1,16], interacting on the surface or propagating at different levels as internal waves [28].

The case where $F = \mathbf{u}\cdot\mathbf{u}^*$ corresponds to cubic nonlinear Schrödinger systems. When $n = 1$ such systems have been shown to be integrable by the Inverse Scattering Transform. For a large class of initial value problems with data vanishing at infinity, this transform reduces the solution of the system to the solution of an associated linear eigenvalue problem. In these cases, the solution of the cubic NLS equation evolves asymptotically in time to a finite set of solitons. Integrability of the cubic NLS equation in the case $n = 1$ has been shown to be closely associated with symmetry or invariance properties of the systems, and specifically with the existence of infinitely many conservation laws [3]. For an extensive bibliography related to Nonlinear Schrödinger Systems, also see [14,13,10,18].

In this article, we use the *DiffElim* and *RifSimp* algorithms to study the Lie symmetries of the class of VNLS equations

$$i\mathbf{u}_t + \nabla^2\mathbf{u} + \left(a(t)|\mathbf{x}|^2 + \mathbf{b}(t)\cdot\mathbf{x} + c(t) + d|\mathbf{u}|^{\frac{4}{n}}\right)\mathbf{u} = \mathbf{0}, \tag{2}$$

where $a(t), b(t)$ and $c(t)$ are arbitrary functions of $t$. We use the class above as a benchmark problem depending on $m$ and $n$. Time and memory statistics are recorded, and a comparison of the *RifSimp* algorithm in *Maple* and the *DiffElim* algorithm in C is given.

The *DiffElim* algorithm in the *CDiffElim* environment runs on the test suite an average of 400 times faster than our *RifSimp* algorithm in Maple, requiring an average of 1/9 of the Memory.

Although the primary focus of this paper is to introduce the *CDiffElim* environment and the *DiffElim* algorithm, and demonstrate its performance relative our well-developed Maple program *RifSimp*, new results are obtained, which appear to be of potential physical interest. Namely the general class of VNLS systems (2) is identified, and found to have large symmetry groups (maximal in the case $m = n = 1$).

In Section 2 we provide an example-based description of the concepts of differential elimination. In Section 3 we give a more detailed motivation of the project, and some of the concepts being used to create the enhanced implementation. In Section 4 a description of the enhancements in the *CDiffElim* environment is given, and in Section 5 the *DiffElim* algorithm is presented. In Section 6 we describe a hybrid symbolic-numeric differential elimination algorithm. Sections 7 and 8 give performance benchmarks. In Section 9 we close with a short discussion.

## 2 Differential Elimination

A brief introduction, mainly by example, is given for differential elimination algorithms (see [5,35,37] for formal treatments). As an immediate illustration of the use of differential elimination algorithms consider the following example.

**Example 1** *Consider the* ODE $u_{xx} = u^2$. *The overdetermined system of* PDE *for the infinitesimal symmetries* $(x, u) \mapsto (\hat{x}, \hat{u}) = (x + \xi(x, u)\epsilon + \mathcal{O}(\epsilon^2), u + \eta(x, u)\epsilon + \mathcal{O}(\epsilon^2))$ *can be algorithmically obtained from the* ODE *and is:*

$$\begin{aligned} \xi_{uu} = 0, \qquad -3u^2\xi_u + 2\eta_{ux} - \xi_{xx} = 0, \\ \eta_{uu} - 2\xi_{ux} = 0, -2\eta u - 2u^2\xi_x + u^2\eta_u + \eta_{xx} = 0. \end{aligned} \tag{3}$$

*The simplified form produced by the RifSimp and DiffElim algorithms, Mansfield's algorithm [22], and Boulier's algorithm [5] is:*

$$\eta_x = 0, \quad \xi_x = -\frac{\eta}{2u}, \quad \eta_u = \frac{\eta}{u}, \quad \xi_u = 0\,. \tag{4}$$

*This form was obtained from the original system using only differentiation and elimination. The number of arbitrary constants in the solution (which is the dimension of the symmetry group of the* ODE*), can be also be algorithmically determined from the simplified form. Not only can the dimension be determined (which is two), but also the structure constants of the Lie algebra of the symmetry group and the formal power series solution for the symmetries to any finite order [31,30]. While, the example above can easily be completed by hand, the algorithms for determining the dimension and structure of unknown symmetry groups of differential equations, are guaranteed to determine this information in a finite number of steps, regardless of the complexity of the determining system.*

### *2.1 Framework*

In this paper we consider systems which are polynomially nonlinear. The most basic step in the theory of differential elimination algorithms is to replace differential equations by algebraic ones, by regarding derivatives as formal variables. For example, the differential equations on the left below, are replaced by the formal expressions on the right.

$$\frac{\partial\Psi(x,y)}{\partial x} - y\Psi(x,y) = 0 \leftrightarrow u_x - yu,$$
$$\frac{\partial\Psi(x,y)}{\partial y} - \Psi(x,y) = 0 \leftrightarrow u_y - u.$$

The differentiations on the formal expressions are carried out by using the formal commuting total derivatives:

$$D_x := \partial_x + u_x\partial_u + u_{xx}\partial_{u_x} + ..., \quad D_y := \partial_y + u_y\partial_u + u_{xy}\partial_{u_x} + ... \tag{5}$$

Above $D_y(u_x - yu) - D_x(u_y - u) = -yu_y - u + u_x \to -u$, by virtue of substitution from $u_x - yu = 0, u_y - u = 0$.

A polynomially nonlinear $q$th order system with $n$ independent variables $x = (x_1, x_2, ..., x_n)$ and $m$ dependent variables $(\Psi^1, \Psi^2, ..., \Psi^m)$ is represented as a polynomial in $x, u, \underset{1}{u}, \underset{2}{u}, ...$, with rational number coefficients. Equivalently and more formally it is represented as an element of the differential ring $\mathbb{Q}[x, u, \underset{1}{u}, \underset{2}{u}, ...]$ where

$$u = \underset{0}{u} \leftrightarrow (\Psi^1, \Psi^2, ..., \Psi^m),$$

$$\underset{1}{u} \leftrightarrow \frac{\partial \Psi^l}{\partial x_j},$$
$$\underset{2}{u} \leftrightarrow \frac{\partial^2 \Psi^l}{\partial x_j \partial x_k},$$
$$\ldots$$

The formal total derivative is:

$$D_{x_j} = \frac{\partial}{\partial x_j} + \sum_l u^l_{x_j} \frac{\partial}{\partial u^l} + ... \tag{6}$$

For sake of notational brevity, it has become customary to use the variable names as the names of the variables in the original physical formulation of the differential system. We follow this convention in our paper.

Thus we consider systems of total derivative order $q$, of form $R^0 = 0, R^1 = 0, ..., R^s = 0$, or more concisely $R = 0$, where

$$R^k : J^q \to \mathbb{C}, \quad J^q = \mathbb{C}^{N_q}, \tag{7}$$

and $N_q = n + m \begin{pmatrix} q+n \\ q \end{pmatrix}$, is the number of jet variables of order less than or equal to $q$.

### *2.2 Rankings, Elimination, and Case Splitting*

In order to perform differential elimination of a system in an structured way, it is necessary to impose a ranking (or ordering of the derivatives) on that system. This ranking must be invariant under differentiation, must be positive, and must be a total ordering. The classification of all rankings is given in [35,36] (also see [9]).

Two of many viable rankings for Example 1 containing $\xi(x, u)$, $\eta(x, u)$ are given below.

$$\xi \prec \eta \prec \xi_u \prec \eta_u \prec \xi_x \prec \eta_x \prec \xi_{uu} \prec \eta_{uu} \prec ... \tag{8}$$
$$\eta \prec \eta_u \prec \eta_{uu} \prec ... \prec \eta_x \prec \eta_{xu} \prec \eta_{xuu} \prec .... \prec \xi \prec \xi_u \prec \xi_{uu} \prec ... \tag{9}$$

Once an ranking is imposed, it is possible to select the highest ranked derivative for an equation, which is identified as the *leading derivative* of that equation. The equation is said to be *leading linear* if it is linear in the leading derivative, otherwise it is said to be *leading nonlinear*. For leading linear equations, the leading derivative can be isolated, and the equation can be used to eliminate it, and all derivatives of it from the remainder of the system.

If the coefficient of a leading derivative in a linear equation, called the *pivot*, contains derivatives, dependent functions, or constants, then it is necessary to assume that this coefficient is not identically zero when isolating the leading derivative. This results in a natural splitting into two cases. In the first case, the assumption that the pivot is non-zero is appended to the system (as an equation of the form $pivot \neq 0$). In the second case, the assumption that the pivot is identically zero is appended to the system, which changes the leading derivative of the equation being considered. These two cases are completed in turn, and each can contain additional case splittings, resulting in a binary tree of cases.

### *2.3 Integrability Conditions*

For differential systems, there is another consideration in addition to the systematic elimination of derivatives from the system, and it involves additional conditions that must be satisfied for the system to be in completed form (or *rif*-form for the *RifSimp* algorithm [35]). These are called *integrability conditions* and they result from the fact that partial differentiation is commutative, and that it is possible to isolate two derivatives of the same dependent variable in a system.

**Example 2** *Consider the system below, with $\xi = \xi(x,u)$, $\eta = \eta(x,u)$, and $\mu = \mu(x,u)$ under a ranking $\mu \prec \eta \prec \xi_u \prec \xi_x$:*

$$\xi_x = \mu, \xi_u = \eta. \tag{10}$$

*Then by equality of mixed partial derivatives, the relation*

$$0 = \xi_{ux} - \xi_{xu} = (\xi_u)_x - (\xi_x)_u = (\eta)_x - (\mu)_u = \eta_x - \mu_u,$$

*must also be satisfied for all solutions of the system.*

If an integrability condition does not reduce to zero with respect to the current system, it is appended to the system as a new equation. It should be noted that for $n$ equations, solved for derivatives of the same dependent variable, the number of conditions to be checked by equality of mixed partials is $\mathcal{O}(n^2)$. More elaborate and efficient algorithms for avoiding redundant integrability conditions are discussed in [30,35,4].

### 2.4 Initial Data

Once a system is in *rif*-form, it is possible to algorithmically determine the data required for the existence of a unique formal power series solution. This data is called the *initial data* for the system, and consists of the values for all derivatives of all dependent variables in the system that cannot be eliminated by any equations in the system.

**Example 3** *For the system $\xi = \xi(x, u)$ and $\eta = \eta(x, u)$,*

$$\xi_{xx} = \xi, \xi_u = \xi, \eta_x = \xi, \tag{11}$$

*the initial data is given by*

$$\xi(x_0, u_0) = c_1, \xi_x(x_0, u_0) = c_2, \eta(x_0, u) = f_3(u), \tag{12}$$

*where $c_1$, $c_2$, and $f_3(u)$ are the initial data needed to construct a unique formal power series solution at $(x_0, u_0)$.*

Algorithms for describing initial data go back to [19,34]. Efficient versions are given in [30] (also see [38]), and versions in the presence of constraints are described in [33,35].

## 3 The DiffElim and RifSimp Algorithms

The *RifSimp* package (designed by the authors) is a mature package that allows a convenient and detailed analysis of over-determined differential systems.

The primary motivation behind the design and implementation of the *CDiffElim* environment, and the *DiffElim* algorithm was to address several difficulties encountered when using the *RifSimp* algorithm, specifically memory blow-up and speed issues. As the *RifSimp* package evolved, more and more complicated strategies were devised which resulted in significant changes to its core algorithms. This resulted in a code that performs very well for many problems, but is difficult to change and adapt for other problems.

These issues motivated the long term project that evolved into the *CDiffElim* environment. The first step of the project was to design compact data structures needed to efficiently implement basic operations of differential elimination. Once the design and implementation of these data structures was in place, the basic algorithms were built. This was followed by the implementation of some of the more general algorithms (such as reduction, differential reduction, square-free factorization, etc.).

Once these elements were all in place, the next phase in the *CDiffElim* Project was to implement the *DiffElim* algorithm using these components.

Advantages of the *CDiffElim* environment over a general computer algebra system include the following:

- As is shown in this article, a careful implementation in C, can offer significant reduction in the time and memory usage required to simplify large and/or complex differential systems over an equivalent algorithm written in a general purpose computer algebra system.
- The algorithms and data structures can be optimized for efficiency when working with differential elimination problems. In contrast, an efficient implementation of equations with ordered terms within Maple is not currently available at the time of writing this paper.
- The general structure of the high-level algorithms should be kept simple and flexible, so that changes can be made easily.

Disadvantages of a low-level stand alone implementation compared with implementations in a general purpose computer algebra system include more difficult bug fixing, and lack of access to the large number of packages and general advances available in a general purpose computer algebra system.

## 4 The CDiffElim Implementation

The *CDiffElim* environment uses a number of well known algorithms for enhancing the speed of symbolic computations. One of these, for example, is the *binary buddy* memory management system, which is a system that allocates large blocks of data using *malloc*, then parcels out the memory for each allocation based on a binary storage structure (see [20]). It was found that use of this memory management scheme gave a speed increase of a factor of 2 (for a Linux libc5 system) over use of the raw *malloc* and *free* functions in the C standard library, which are implemented using system calls.

At present, the *CDiffElim* environment is only capable of working with polynomial systems in the dependent and independent variables, though future plans include extension to more general analytic functions. In many cases, it is possible to represent functions such as $\sin(x)$, $\ln(x)$, and $\exp(x)$ using differential extensions, (i.e. using differential equations to define these functions), so this limitation is not as severe as it would initially seem.

In this section we present some of the core data structures being used in the *CDiffElim* implementation, and briefly discuss some associated enhancements.

### *4.1 Considerations*

For most differential elimination problems, the input systems are relatively sparse in their indeterminates (with respect to their algebraic degree), so our focus will be on methods for sparse equations in many indeterminates. For this reason, equations are stored using a sparse distributed representation.

In differential elimination, the ability to store the terms of equations in the order determined by the ranking is vital to an efficient implementation. Without this ordered storage, the process of reduction requires $\mathcal{O}(k)$ term comparisons for a $k$ term equation to determine the leading term (i.e. the term of maximal rank). Application of a reduction process to the entire $k$ term equation Checking each term in the equation for a reduction, even when no reduction occurs, will then require $\mathcal{O}(k^2)$ operations.

Though ordered storage is possible in Maple using lists or arrays, the core algorithms (such as multiplication, division, and GCDs) are not implemented for these structures. At present, each use of these algorithms would require the conversion of the ordered representation into a Maple equation, and the conversion of the result back to ordered form. The conversion back would require a sort, which is an order $\mathcal{O}(k\ \log_2(k))$ expense for a $k$ term polynomial.

Development of the *CDiffElim* environment in C gives us the ability to design an ordered equation data type, and build algorithms to efficiently perform basic operations on these ordered equations.

### *4.2 Data Structures*

**Numerical Representation**: The numerical data type (hereafter referred to as a *varint*) has the ability to represent either an arbitrary precision integer or a modular integer (required for fast modular GCD and resultant computations). Arbitrary precision integers are stored in base 65536 allowing fast machine computation of integer addition, subtraction, division, multiplication, and GCD (see [21]). The Karatsuba algorithm for fast multiplication has not been implemented, but the well known binary GCD algorithm is in place.

**Indeterminates**: The unknowns in the equations (currently including dependent variables, independent variables, and constants), are stored in a global symbol table. This table also stores information allowing rapid computation of the derivatives of these indeterminates. All objects external to this table are stored by reference.

**Terms**: Monomial terms are constructed of two components, a coefficient (which is a varint), and an indeterminate list. The indeterminate list is ordered (with respect to the ranking of the derivatives), and contains the reference to each indeterminate, and it's algebraic degree for that term. It is currently under investigation if the use of a global term table will give significant memory savings without seriously impacting running time in the implementation.

**Equations**: Equations are constructed as an ordered list of terms, in the order prescribed by the term ranking. This ordered representation can be stored as a partially balanced binary tree (see red-black trees in [11]), or as a linked list. The most commonly used form is the linked list structure, but the binary tree representation is needed for algorithms where the ordered form of the equations cannot easily be retained (input parsing and differentiation for example).

**Other Structures**: In addition to these base data structures, more specialized algorithm-specific data structures were also implemented. For fast GCD computation, for example, modular univariate equations and modular matrix data structures were needed.

### *4.3 Basic Operations*

Basic operations on integers are implemented based on the algorithms discussed in [21]. This includes multi-precision addition, subtraction, multiplication, division, and the binary GCD algorithm. Polynomial addition and subtraction is implemented as a merge of the two ordered input polynomials, with $\mathcal{O}(k+l)$ cost, where $k$,$l$ are the number of terms in each of the polynomials.

#### *4.3.1 Polynomial Multiplication*

The ordered structure of equations allows for a recursive multivariate polynomial multiplication algorithm that maintains the ordered structure at little additional asymptotic cost. Given that we have multivariate polynomials with $k$ and $l$ terms, the cost of performing the multiplication is $\mathcal{O}(k\,l)$. A straightforward implementation of this multiplication, followed by a merge sort of the terms would have an asymptotic complexity of $\mathcal{O}(kl\ \log(kl))$. Rather than using this method, the smaller of the two factors is split into two equations of equal size, and the routine is called recursively. Once there is only one term remaining, the product with the larger equation is computed and the result returned. When the routine receives the two results, it then performs a single-pass merge of the computed results requiring the same number of comparisons as the total number of terms in the two equations. Without loss of generality it can be assumed that $k > l$.

| Depth | Number of calls | Multiplications per call (exact) | Term comparisons per call (bound) |
|---|---|---|---|
| 1 | 1 | 0 | $lk$ |
| 2 | 2 | 0 | $\frac{l}{2}k$ |
| ... | ... | ... | ... |
| $\log_2(l) - 1$ | $\frac{l}{2}$ | 0 | $2k$ |
| $\log_2(l)$ | $l$ | $k$ | 0 |

There are a total of $kl$ multiplications, and an upper bound of $kl\ \log_2(l)$ term comparisons, so the worst case running time of this ordered polynomial multiplication algorithm is $\mathcal{O}(kl\ \log_2(l))$. It should also be noted that the worst case only occurs when there are very few like terms in the resulting product. For most practical computations involving large polynomials this is not the case, and the complexity is dominated by the $\mathcal{O}(kl)$ multiplication cost.

### *4.3.2 Polynomial Division*

The ordered structure for polynomials discussed above facilitates an efficient implementation of division. The resulting division algorithm is a modification of the classical polynomial division algorithm that expedites the division of polynomials having many terms by polynomials having relatively few. This modification is most relevant to division of ordered polynomial data structures.

We consider the problem: *given* $a, b \in \mathbb{Z}[x, y, ...]$ *find* $q$ *such that* $q \in \mathbb{Z}[x, y, ...]$ *and* $a = b\,q$. We assume that $a$ and $b$ are stored in order with respect to a ranking $\prec$. For purposes of the following analysis we identify that $b$ has $l_1$ terms, $q$ has $l_2$ terms, and $a$ has $k \leq l_1 l_2$ terms (terms here refer to the number of distinct monomials). Note that we do not consider the class of problems where there is a remainder.

It is straightforward to see that classical division of $a$ by $b$ will require $l_2$ steps, each involving one term division, $l_1$ multiplications, and $l_1$ additions, so the complexity of classical division is $\mathcal{O}(l_1 l_2)$. One may also observe that when $l_1 \ll l_2$, or $l_2 \ll l_1$, the expense of performing the division is roughly $\mathcal{O}(\max(l_1, l_2))$ and that the worst case occurs in the *balanced* case when $l_1 \approx l_2$ with complexity $\mathcal{O}({l_1}^2)$.

A problem arises in the implementation of the classical polynomial division algorithm for ordered equations. Since polynomial addition is implemented as a merge of two ordered equations, division when $l_1 \ll l_2$ can have complexity $\mathcal{O}({l_2}^2)$. As an illustration of this problem, consider the ordered division of the polynomial

$$r = xz^d + xz^{d-1} + ... + x - yz^d - yz^{d-1} - ... - y \tag{13}$$

by $x - y$ (we will refer to the polynomial $r$ as the remainder). Since $(xz^d)/x = z^d$, the first term of the quotient is $z^d$, and in the first step of the division process $xz^d - yz^d$ is subtracted from the remainder, resulting in the cancellation of the $xz^d$ term. In subtracting the $yz^d$ term from the remainder, $d$ term comparisons must be made to find the location of the $yz^d$ term to perform the subtraction. A similar pattern repeats $d$ times. The total number of term comparisons required for this division is $d(d+1)/2$, which is $\mathcal{O}({l_2}^2)$, which is the same as the balanced case.

So we see that the problem occurs as the direct result of having to find the location in the remainder of a term from the product of the divisor term and the dividend. Examination of the process of division easily reveals a few facts. The quotient terms resulting from each step of the division process are generated in strictly descending order with respect to the ranking. This is only natural, as the equation is being stored in descending order with respect to the ranking, and the ranking is preserved under multiplication. Also, using a similar observation, the location of the product of a term of the divisor and the current term of the quotient will always have a location in the remainder that follows the product of that term of the divisor and the prior term in the quotient.

To avoid the efficiency difficulties mentioned earlier, we take advantage of these observations to drastically reduce the number of term comparisons in the implementation of the division process in *CDiffElim*.

In the first division step, we store a location in the remainder where we begin searching for the location of each term of the quotient-divisor term product for each term of the divisor. In all following division steps, we use these stored locations as a starting point to determine where the product of the next quotient term and the corresponding term of the divisor belongs, and adjust the stored location as necessary.

As an illustration of the concepts described above, consider the state of the remainder of (13) after the first division step (where the stored locations are marked with the underbrace):

$$\underbrace{xz^{d-1}} + ... + x - \underbrace{yz^{d-1}} - ... - y. \tag{14}$$

In the next division step, the stored locations correspond exactly to the terms required when subtracting $xz^{d-1} - yz^{d-1}$ from the remainder.

An asymptotic result for this modified algorithm can be obtained as follows:

The total number of comparisons needed for the first division step is at most $k < l_1 l_2$. The worst case number of comparisons needed in the following division steps is $\mathcal{O}(l_1 l_2)$ for each term of the divisor (here $l_1 l_2$ is the maximum possible length of the dividend). The reasoning for this is that the stored location for each term of the dividend is only ever moved to lower order terms in the remainder, and the maximum number of terms in the remainder is $l_1 l_2$. So we obtain the number of comparisons as $\mathcal{O}(k + {l_1}^2 l_2)$, which when $l_1 \ll l_2$ gives $\mathcal{O}(l_2)$, the same order as the number of arithmetic operations required for the division.

In conclusion, when $b$ has few terms relative to $a$, we can expect we are in the $l_1 \ll l_2$ case, and use this modified division algorithm. When the number of terms of $a$ and $b$ are on the same order, the unmodified algorithm should be used instead. It should be noted that there are also issues involving the size of the coefficients in the input expressions. This lends itself well to a fast modular division algorithm that can better handle large coefficients. This will be discussed in future work.

### *4.3.3 Differentiation*

The ordered approach used for addition, multiplication and division does not lend itself well towards differentiation, so an alternative approach is needed.

The derivative of an equation is constructed through differentiation of each term. As each term is differentiated, it is inserted into a partially balanced binary tree (see [11]). This has the advantage that any common terms, resulting from differentiation of different terms, will not incur more than a single term memory allocation, if implemented correctly.

For example, consider the differentiation of a quadratic polynomial having $l$ terms, with a result having $k \leq 2l$ terms. The cost of the differentiation into ordered form requires approximately $2\,l\,\log_2(k)$ comparisons (due to the use of the partially balanced binary tree), and $k$ memory allocations. An algorithm that were to perform all differentiations and sort afterwards, would require the same number of term comparisons if performed with a fast sorting algorithm, but would require $2l$ memory allocations instead.

The same structure is used for parsing of the input equations.

### *4.3.4 Multivariate polynomial GCDs*

The basis for all multivariate GCD code in *CDiffElim* is a recent modification of Brown's dense modular GCD algorithm (see [7], [24]). The algorithm includes an inexpensive probabilistic method for obtaining degree bounds for each variable in the GCD. This method allows, with high probability, rapid determination of a trivial GCD, which is the most common case in many of the operations of differential elimination.

We briefly describe the process of computation of a dense GCD using a Brown-type algorithm here. Suppose we have a GCD problem in the polynomial ring $\mathbb{Z}[w, x, y, z]$. First a main variable is selected, say $w$, and the polynomials are mapped into the field $\mathbb{Z}_p[w]$ by selection of an appropriate prime $p$, and evaluation of $x, y, z$ at values in $\mathbb{Z}_p$. We call the polynomials undergoing these evaluations *images* in $\mathbb{Z}_p[w]$. The image of the GCD in $\mathbb{Z}_p[w]$ is then computed from the images of the input polynomials by Euclidean algorithm (as $\mathbb{Z}_p[w]$ is a Euclidean domain). Another evaluation is performed keeping the values of $y, z$, and $p$ the same, but changing the value of $x$, obtaining another image of the GCD in $\mathbb{Z}_p[w]$. This process continues until a sufficient number of GCD images in $\mathbb{Z}_p[w]$ are obtained to reconstruct the dependence of the GCD on $x$ via interpolation. A similar process is followed to reconstruct the dependence of the GCD on $y$ using images in $\mathbb{Z}_p[w, x]$, and again a similar process for $z$. At this point it should be noted that the algorithm has complexity that is exponential in the number of variables. i.e. obtaining each image in $\mathbb{Z}_p[w, x, y]$ requires a number of images in $\mathbb{Z}_p[w, x]$, each of which require a number of images in $\mathbb{Z}_p[w]$. Finally, using the required number of images in $\mathbb{Z}_p[w, x, y, z]$ (determined via a GCD coefficient size bound), the GCD in $\mathbb{Z}[w, x, y, z]$ can be recovered through chinese remaindering. For more detail, please consult [7], [24].

An algorithm that is much better for sparse polynomials in many variables is the Zippel Sparse GCD algorithm (see [49]). The algorithm computes at least one dense image at each level (i.e. in $\mathbb{Z}_p[w, x]$, $\mathbb{Z}_p[w, x, y]$, and $\mathbb{Z}_p[w, x, y, z]$), then uses modular linear algebra, and the form of the dense image to obtain the necessary images for reconstruction in the current variable. So, for the above problem in $\mathbb{Z}[w, x, y, z]$, and assuming that the GCD was sparse, say we have an image in $\mathbb{Z}_p[w, x, y]$. This gives us the form of the GCD with high probability (assuming the primes are sufficiently large compared to the number of terms in the GCD), and the known form is used to construct the image for the next $z$ value more cheaply than could be accomplished using the recursive approach. Only univariate GCDs in $\mathbb{Z}_p[w]$ and the solution of a $\mathbb{Z}_p$ linear algebra problem are needed, instead of calculating trivariate GCDs in $\mathbb{Z}_p[w, x, y]$. Once this is done for a sufficient number of $z$ values, the dependence of the GCD on $z$ can be reconstructed via dense interpolation.

The additional knowledge of tight degree bounds for the GCD reduces the number of images required for the dense interpolation. This has the most pronounced effect on the Brown algorithm, but since the Zippel algorithm also requires at least one dense interpolation for each variable, it also benefits from these bounds.

One problem not discussed above arises in the interpolation process, and it is known as the *normalization* problem. The root of the problem is that the images must be correctly scaled (by a factor in $\mathbb{Z}_p$) before performing the interpolation, as otherwise the interpolation will not produce the correct result. Note that this does not occur for GCD that are monic in the main variable. For the Brown algorithm, this difficulty can be solved by use of a single univariate GCD computation. For the Zippel algorithm, since the dependence of many variables is being reconstructed at one time, the additional GCD computation is no longer univariate. Performing this computation would seriously degrade the performance of the algorithm, so an alternative method is required.

It was possible to map the Zippel sparse GCD reconstruction for the non-monic case into a larger linear algebra problem in $\mathbb{Z}_p$. This has the added benefit that no additional GCD computations must be done for the non-monic case, but the linear algebra problem is slightly more difficult.

## 5 The DiffElim Algorithm

As a test of the *CDiffElim* environment, the *DiffElim* algorithm has been implemented in this environment. A basic outline of the algorithm resembles a modification of the Buchberger algorithm from [8].

Consider systems which are linear in the dependent variables and their derivatives, with coefficients which are rational functions of the independent variables. For such systems the output of the *DiffElim* algorithm is in *rif*-form, and yields an existence and uniqueness theorem [37,35] for the input system. Although *DiffElim* is capable of taking inputs which are nonlinear in the dependent variables and their derivatives, this part of the algorithm is still being implemented, and will be described in future work [46].

Given the input system, one equation is selected at a time, and reduced with respect to the already simplified equations. The D-polynomials, in the linear context, are the integrability conditions as described in [35]. For nonlinear systems, they represent an extended form of integrability condition combined with algebraic S-polynomials that will be described in [46]. Moreover, the input ranking to the algorithm is one of those described in [35,36].

Rather than forming the integrability conditions (hereafter referred to as DPolys), references are stored, and the DPolys are computed when needed. Once all DPolys are either appended to the system (possibly resulting in new DPolys) or reduced to zero, the algorithm terminates, and returns the reduced system with all integrability conditions satisfied.

### *5.1 DiffElim Algorithm Pseudocode*

```
Input:
   U:  input system of PDE
   <_d: ranking
   C:  criteria for the behavior of the algorithm
Output:
   G:  the reduced form of the input system

   G := {}, S := {}
   while U ≠ {} or S ≠ {} do
      neweq := SelectFrom(U,S,<_d,C_select)
      if neweq is an DPoly then
         S := S - neweq
         neweq := MakeDPoly(neweq,G)
      else
         U := U - neweq
      end if
      neweq := Reduce(neweq,G,<_d,C_reduce)
      if CheckCriteria(neweq,C_accept) is false then
         U := U ∪ neweq
      else if neweq is not identically zero then
         G := Reduce(G,neweq,<_d,C_reduce)
         G_r := HasReduced(G)
         if elements of G have reduced (i.e. G_r ≠ {}) then
            G := G - G_r
            U := U ∪ G_r
            S := S - GetRefs(G_r,S)
         end if
         S := S ∪ BuildDPolys(neweq,G,<_d,C_compat)
         G := G ∪ neweq
      end if
   end while
   return G
```

We now present a brief description of some of the components of the algorithm. The details of the algorithm for nonlinear systems are not presented at this time [46].

**SelectFrom**: Determines the next best equation to use for the system. The determination of the equation or DPoly to be used is based on the $C_{select}$ criteria. The selection criteria has the capability of choosing to compute a DPoly instead of selecting an equation from $U$, if none of the equations in $U$ satisfy the criteria.

**MakeDPoly**: Constructs DPolys, which for linear systems are those described in [35].

**Reduce**: Reduces the equations given in the first input by the equations in the second input. The type of reduction to perform is based on the $C_{reduce}$ criteria. For linear systems the reduction is that defined in [35].

**CheckCriteria**: Validates that the equation still satisfies the selection criteria after being reduced. This is necessary as the process of reduction can cause the complexity of the input equation to increase significantly. For example the equation can become excessively large, or become more dense in the indeterminates of lower rank. The decisions are made based on the $C_{accept}$ criteria.

**HasReduced**: Determines which of the input equations have been changed by the last reduction. This is allows the implementation to be *update* driven.

**GetRefs**: Obtains all DPoly references from the second input that depend on the equations in the first input. Again required for the update driven implementation.

**BuildDPolys**: Constructs all DPolys between the input equation, and the current solved form system $G$.

## *5.2 Adjustable Criteria*

There are various criteria that can be specified for a run of the algorithm. These criteria, in combination with the input ranking, offer some flexibility as to the manner in which the system is reduced. The currently available criteria are:

$\mathbf{C_{select}}$:
The selection criteria is used to determine a good candidate equation or DPoly to adjoin to the simplified system. Currently this criteria can be based on 4 measures:

1 The overall size of the equation;
2 The differential order of the equation;
3 The algebraic degree of the equation;
4 The size of the coefficient of the leading derivative in the equation.

A hybrid of the above criteria may also be specified. For DPolys, the quantities are estimated from the pair of equations that are used to create the DPoly.

$\mathbf{C_{accept}}$:
As mentioned in the discussion of *CheckCriteria*, the complexity of an equation can increase greatly in the process of reduction. This criteria can be set to delay the use of equations of this type until further equations are uncovered (that may reduce this equation to a simpler form). This may be used to limit the growth of equations during reduction to a certain percentage of the original size. Alternatively it can be used as a screen, with a very loose setting for $C_{select}$, to force the algorithm to consider the reduced form of the DPolys.

$\mathbf{C_{reduce}}$:
This criteria controls whether pure reduction, pseudo-reduction, or a combination of the two is being used. By a combination of the two, we refer to pseudo-reduction over a restricted set of variables. In this case of limited pseudo-reduction (the type of reduction used for the RifSimp algorithm), any leading nonlinear equations are treated as purely algebraic constraints.

$\mathbf{C_{compat}}$:
This criteria controls how DPolys are generated, and is closely related to $C_{reduce}$. If pseudo-reduction is in use, this generates the spawn of any leading nonlinear equations (by differentiating each equation with respect to each independent variable) and normal integrability conditions for any leading linear equations. The spawn method is needed to enforce any differential consequences of leading nonlinear equations. If pure reduction is in use, the extended form of the DPolys is used. If the reduction form is mixed, then a combination of the methods is used.

For all systems that are simplified in this paper, the algorithm was used with the settings that give it the same behavior as *RifSimp*.

## 6 Hybrid Symbolic-Numeric Differential Elimination Method

In the previous sections purely symbolic methods were outlined to counter expression explosion in differential elimination algorithms. In this section we briefly describe an alternative approach based on combining techniques from numerical analysis and symbolic computation. This work falls under the new area of hybrid symbolic-numeric computation [41,42].

Symbolic elimination algorithms (e.g. the Gaussian elimination and Buchberger algorithms) can be easily converted to symbolic-numeric ones by replacing exact arithmetic with floating point arithmetic. The resulting methods, however, are almost always unstable with respect to small changes in the coefficients of the input systems.

The approach outlined in this section uses only symbolic differentiation, and avoids symbolic elimination, addressing the significant expression swell seen in many symbolic elimination algorithms. In contrast, symbolic differentiation is relatively cheap. Techniques from Numerical Linear Algebra [15,45], in particular the Singular Value Decomposition (SVD) are used.

Our approach is motivated by the geometric differential elimination methods that arose in the approaches of Cartan and Kuranishi. This work has also been influenced by [39] in which an approach is proposed for ordinary differential systems that generate prime differential ideals. That approach uses symbolic differentiations, and avoids nonlinear eliminations, instead using linear eliminations based on substitution of a random point.

The spirit of the geometric approach to differential equations is that it is concerned with the (jet) variety of the system:

$$\mathcal{R} = V(R) := \{(x, u, \underset{1}{u}, ..., \underset{q}{u}) \in J^q : R^k(x, u, \underset{1}{u}, ..., \underset{q}{u}) = 0\}, \tag{15}$$

where $\underset{r}{u}$ represents the $r$th order derivatives as described in Section 2. If all of the equations in the system have derivative order exactly $q$ then a single symbolic prolongation of the system is taken to be:

$$D(R) := \{(x, u, \underset{1}{u}, ..., \underset{q}{u}) \in J^q : R^k = 0,\ D_{x_i} R^k = 0\}. \tag{16}$$

If there are equations with derivative order less than $q$ then derivatives of these equations are appended to the system, and this process continued until no undifferentiated equations of lower order remain.

The prolongations of a system can be obtained more efficiently by omitting the obviously equivalent equations resulting from the commutativity of partial differentiation. For example, given $\frac{\partial R^1}{\partial x_1}$, $\frac{\partial R^1}{\partial x_2}$, in the first prolongation, $\frac{\partial}{\partial x_1}\frac{\partial}{\partial x_2}R^1$ only needs to be listed once in the second prolongation. These and many more refined efficiency improvements are discussed in [40].

The corresponding geometric operation to elimination is that of geometric projection. Indeed a single geometric projection is defined as:

$$E(R) := \{(x, u, \underset{1}{u}, ..., \underset{q-1}{u}) \in J^{q-1} : R^k(x, u, \underset{1}{u}, ..., \underset{q-1}{u}, \underset{q}{u}) = 0\}. \tag{17}$$

We call the systems: $ER$, $E^2R$, ... , $E^qR$ projected systems. They are of respective orders: $q$, $q-1$, ... ,0. A differential system of order $q$ is said to be involutive [29,40], if it passes the elimination test: $(E \circ D)R = R$ and has involutive symbol.

The symbol of a $q$th order system is the linearized highest order part of the system, which is given by the matrix:

$$\text{Symbol } R := \frac{\partial R}{\partial \underset{q}{u}}. \tag{18}$$

The symbol of a $q$th order system $R = 0$ is involutive if

$$\text{rank Symbol}(DR) = \Sigma_{k=1}^{n} k\beta_k^{(q)} \tag{19}$$

and the $\beta_k^{(q)}$, are the dimensions of certain subspaces of the Null Space of the Symbol of $R$. Details of this test are given in [40,29].

Involution enables a local existence and uniqueness theorem to stated for the system. It has been used to determine consistent initial conditions for the numerical solution of systems. In addition, involutive systems are of low (0 or 1) differential index, and reduce certain aspects of the numerical difficulty of solving such systems.

Typical symbolic implementations to obtain involutive systems try to keep the order of the systems as low as possible through use of symbolic elimination (as is the case with the *RifSimp* algorithm). Thus a symbolic implementation of the involutive form algorithm uses Gröbner Bases, or symbolic Gauss Elimination to perform the symbolic eliminations [40].

The approach which will be described here does not use symbolic elimination, and this necessitates the application of the Elimination Test to projected systems.

The main idea of the symbolic-numeric algorithm is to differentiate the order $q$ system until some projection of the differentiated system (of order $\geq q$) is unchanged by a single prolongation and projection, and the symbol of the projected system is involutive.

**Example 4** *Consider the system of ordinary differential equations* $u_{xx} - w =$

*$0, u_{xx} = 0$ with dependent variables $u, w$ and independent variable $x$. In multi-index notation, where $u = u_{(0)}, u_x = u_{(1)}, u_{xx} = u_{(2)}$, etc, this is $u_{(2)} - w_{(0)} = 0, u_{(2)} = 0$. The $k$th prolongation, $D^k(R)$, of the system is $u_{(2+k)} - w_{(k)} = 0, u_{(2+k)} = 0$. At every prolongation $(E \circ D)D^k(R)$, yields the new relation, $w_{(1+k)} = 0$. Consequently, the prolonged system never passes the Elimination Test of the classical geometric theory. In the symbolic implementations of the geometric theory, this difficulty does not arise, since symbolic eliminations would first be applied to the original system, uncovering the equation $w = 0$, which would be then prolonged. Uncovering this equation here is trivial. However for general (especially nonlinear) systems this is the step that can lead to massive expression explosion, as symbolic methods such as Gröbner bases are used to uncover such equations.*

The method is outlined for linear homogeneous systems. A linear homogeneous differential system can be written in the matrix form:

$$B^{(q)}(x)\, \underset{q}{v} = \mathbf{0}, \quad \underset{q}{v} = \begin{pmatrix} \underset{q}{u} \\ \cdot \\ \cdot \\ \underset{1}{u} \\ u \end{pmatrix}. \tag{20}$$

Successive prolongations (differentiations) of the system yield $R, DR, D^2R, ...$, and a sequence of corresponding linear homogeneous matrix systems:

$$B^{(q)}(x)\, \underset{q}{v} = \mathbf{0}, \;\; B^{(q+1)}(x)\, \underset{q+1}{v} = \mathbf{0}, \;\; B^{(q+2)}(x)\, \underset{q+2}{v} = \mathbf{0}, ... \tag{21}$$

where the zero vector in the right hand side of each system has the appropriate dimension for that system.

For linear homogeneous systems our hybrid symbolic-numeric approach is to choose a random point $x = x^0$, and substitute it into the sequence of systems above. This yields a sequence of constant matrix systems:

$$B^{(q)}(x^0)\, \underset{q}{v} = \mathbf{0}, \;\; B^{(q+1)}(x^0)\, \underset{q+1}{v} = \mathbf{0}, \;\; B^{(q+2)}(x^0)\, \underset{q+2}{v} = \mathbf{0}, ... \tag{22}$$

For each prolongation, $D^kR$, the projected systems $E^lD^kR$, $l = 0, 1, ..., k$, are numerically constructed, by replacing the symbolic projection operator $E$ with a numeric projection operator $\hat{E}$. This results in the family of systems $\hat{E}^l(D^k(R))$ given in the table below.

| | order $= q$ | order $= q+1$ | order $= q+2$ | ... |
|---|---|---|---|---|
| Prolongation 0 | $R$ | | | |
| Prolongation 1 | $\hat{E}(DR)$ | $DR$ | | |
| Prolongation 2 | $\hat{E}^2(D^2R)$ | $\hat{E}(D^2R)$ | $D^2R$ | |
| ... | | | | |

The projected elimination test is that for each order of prolongation $k$, the largest $l \leq k$ is determined, if it exists, such that

$$\dim\, \hat{E}^l(D^kR) = \dim\, \hat{E}^{l+1}(D^{k+1}R). \tag{23}$$

The classical Elimination Test is the special case $l = 0$.

**Definition 1** *The system $R = 0$ is said to be symbolically involutive at order $k$, and projected order $l$, if $E^l(D^k(R))$ is involutive. Equivalently it satisfies the projected elimination test (23), and Symbol $E^l(D^kR)$ is involutive. We call the latter test the projected involutive symbol test.*

The numerical implementation of the tests above are briefly discussed. In the numerical implementation, the symbolic elimination $E$ is replaced with a numerical projection $\hat{E}$. We first find the SVD of $B^{(q+k)}(x^0)$ at a random point $x^0$:

$$B^{(q+k)}(x^0) = U\Sigma V^t \tag{24}$$

using one of the available numerical packages (in our case the NAG library). Here $U$ is a unitary matrix, $V^t$ is an orthogonal matrix, and $\Sigma$ is a diagonal matrix whose diagonal entries $\sigma_i$, called singular values, are positive and decreasing. The number of nonzero singular values is the rank of $B^{(q+k)}(x^0)$. The submatrix of $V^t$ obtained by deleting the first $r$ rows of $V^t$ is a basis for the Null Space of $B^{(q+k)}(x^0)$.

In the approximate case, the rank $r$ can be approximated by rounding the singular values of the SVD to zero when they are below a given tolerance. Deleting the first $r$ rows of $V^t$ yields an approximate basis for the Null Space of $B^{(q+k)}(x^0)$. This yields an estimate for dim $(D^kR)$. To estimate $\dim(\hat{E}(D^kR))$ the components of the approximate basis for $D^kR$ corresponding to the highest order ($(q+k)$th order) derivatives are deleted. This projected basis yields an approximate spanning set for $\hat{E}(D^kR)$. Proceeding in the same way, deleting components corresponding to highest order jet variables from the approximate spanning set just obtained, yields an approximate spanning set for $\hat{E}^2(D^kR)$, and then for $(\hat{E}^3(D^kR))$, etc. Application of the SVD to each of these approximate spanning sets yields the approximate dimensions of $E(D^kR), E^2(D^kR), E^3(D^kR), ....$ In this way the dimensions necessary for the application of the approximate projected elimination test (23) are determined.

To execute the projected involutive symbol test (19) bases for the spaces $E(D^kR)$ are first constructed as above. Then the subspaces of their symbols are extracted by using the SVD based Subspace Intersection Algorithm described in [15]. The integer from the left hand side of this test is easily approximated using the methods discussed above. The approximation of the $\beta_k^{(q)}$ is more complicated. In this case, the symbol, or rather approximate bases for each of the subspaces corresponding to the $\beta_k^{(q)}$ are determined by further use of the SVD based Subspace Intersection Algorithm [15].

Note that when a nonlinear system

$$R(x, u, ..., \underset{q}{u}) = 0 \tag{25}$$

is differentiated it becomes linear in its highest ($(q + 1)$-th) order derivatives and has the form

$$\underset{q+1}{A}\,(x, u, ..., \underset{q}{u})\ \underset{q+1}{v} = \underset{q+1}{b}\,(x, u, ..., \underset{q}{u}) \tag{26}$$

Then the new methods of Numerical Algebraic Geometry [41,42] can be used to find an approximate point $(x^0, u^0, ..., \underset{q}{u}^0))$ on a component of the variety of $R = 0$. Substitution of the point into the differentiated system, yields a constant matrix-vector system

$$\underset{q+1}{A}\,(x^0, u^0, ..., \underset{q}{u}^0)\ \underset{q+1}{v} = \underset{q+1}{b}\,(x^0, u^0, ..., \underset{q}{u}^0) \tag{27}$$

Then the methods of Numerical Linear Algebra can be extended to this, the nonlinear case, and this is the focus of current work. Details will appear elsewhere.

We conclude this section with a simple example.

**Example 5** *Consider again the determining system from our simple introductory Example (3). When we write this system in matrix form it becomes:*

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & -2\,u^2 & 0 & 0 & u^2 & 0 & -2\,u \\ -1 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & -3\,u^2 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \underset{2}{v} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

*where* $\underset{2}{v}$ *is the transpose of the vector* $[\xi_{x,x}, \eta_{x,x}, \xi_{x,u}, \eta_{x,u}, \xi_{u,u}, \eta_{u,u}, \xi_x, \eta_x, \xi_u, \eta_u, \xi, \eta]$.

*Our implementation of the symbolic-numeric algorithm produced the following table of approximate dimensions:*

**Table of dim** $\hat{E}^l(D^k R)$

| *Order* $(q+k)$ | *2* | *3* | *4* | *5* | *6* | *7* | *8* |
|---|---|---|---|---|---|---|---|
| *Prolongn k=0* | *8* | | | | | | |
| *Prolongn k=1* | *8* | *8* | | | | | |
| *Prolongn k=2* | *6* | *6* | *6* | | | | |
| *Prolongn k=3* | *4* | *4* | *4* | *4* | | | |
| *Prolongn k=4* | *3* | *3* | *3* | *3* | *3* | | |
| *Prolongn k=5* | *2* | *2* | *2* | *2* | *2* | *2* | |
| *Prolongn k=6* | *2* | *2* | *2* | *2* | *2* | *2* | *2* |

*From the above table we see that the approximate projected elimination test dim* $\hat{E}^l(D^k R) =$ *dim* $\hat{E}^{l+1}(D^{k+1} R)$ *is passed when* $k = 1$, $l = 1$. *However the involutive symbol test is not passed in this case. The approximate elimination test is also passed next when* $k = 5$, *and* $l = 0, 1, 2, 3, 4, 5, 6$. *In all of these cases it can be shown that the projected systems pass the approximate projected involutive symbol test. Assuming that the system also passes the projected involution test implies that the general solution of the system should depend on two arbitrary constants. This can be independently verified by applying a symbolic differential elimination algorithm.*

## 7 Applications to determination of symmetries of Nonlinear Schrödinger Systems

In this section, we discuss our primary benchmark for the *CDiffElim* environment, the application of differential elimination to the infinitesimal symmetry defining systems for vector nonlinear Schrödinger systems of the form (1). This provides a large test suite of overdetermined differential systems, in which both the spatial dimension $n$, and the number of components in the vector potential $m$ can be increased arbitrarily. The suite was challenging enough so that many weaknesses in the *CDiffElim* environment were identified, and efficiency improvements were made. The process of using this as a benchmark has also provided results that may be useful to other researchers.

The formation of symmetry defining systems for infinitesimal symmetries is described. The form of a scalar Nonlinear Schrödinger equation having a maximal dimensional symmetry group is then obtained, and this form is generalized to vector nonlinear systems.

### *7.1 Generating Symmetry Defining Systems for the Nonlinear Schrödinger Systems*

One direct way to determine the symmetries of (1) is to substitute transformations of the form

$$(t, \mathbf{x}, \mathbf{u}) \mapsto (\hat{t}, \hat{\mathbf{x}}, \hat{\mathbf{u}}) = (T(t, \mathbf{x}, \mathbf{u}), \mathbf{X}(t, \mathbf{x}, \mathbf{u}), \mathbf{U}(t, \mathbf{x}, \mathbf{u})) \tag{28}$$

into the system (1). This results in a complex and highly nonlinear overdetermined system of PDE for the unknown functions $T, \mathbf{X}, \mathbf{U}$. Lie's breakthrough was to simplify the problem of finding continuous symmetries by linearizing these transformations about the identity. Following this approach we seek linearized or infinitesimal symmetries of the form

$$(t, \mathbf{x}, \mathbf{u}) \mapsto (\hat{t}, \hat{\mathbf{x}}, \hat{\mathbf{u}}) = (t + \tau\epsilon + \mathcal{O}(\epsilon^2), \mathbf{x} + \boldsymbol{\xi}\epsilon + \mathcal{O}(\epsilon^2), \mathbf{u} + \boldsymbol{\eta}\epsilon + \mathcal{O}(\epsilon^2)), \tag{29}$$

where $\tau$, $\boldsymbol{\xi}$, $\boldsymbol{\eta}$ are unknown functions of $t, \mathbf{x}, \mathbf{u}$. A simple, but inefficient way to obtain the symmetry determining system for $\tau$, $\boldsymbol{\xi}$, $\boldsymbol{\eta}$, is to substitute the linearized form of the transformations above into (1). The determining system is then obtained from the coefficient of the $\epsilon$ term. Lie [26] gave a much more elegant and efficient method in which the determining equations result from applying the prolongation of the linearization operator to the PDE.

Lie Theory is an analytic theory in which all quantities are regarded as complex, including dependent and independent variables. Thus the VNLS system can not be treated directly [23] since it contains the non-analytic function $\mathbf{u} \cdot \mathbf{u}^* = |\mathbf{u}|^2$. To circumvent this difficulty $\mathbf{u}$ is expressed in terms of its real and imaginary parts. This embeds the given system in an analytic system.

Alternatively, one may use the formal complex conjugate of (1):

$$-i\mathbf{u}_t^* + \nabla^2\mathbf{u}^* + F\left(\mathbf{x}, t, \mathbf{u}\cdot\mathbf{u}^*\right)\mathbf{u}^* = \mathbf{0}. \tag{30}$$

Here we have assumed that $\mathbf{x}$ and $t$ are real, and $F$ is a real analytic function, so that $F^* = F$, and $\mathbf{u}^*$ is treated as a new dependent variable. The application of the the prolongation of the linearization operator to the PDE in these variables also yields a determining system. Both methods are equivalent through a simple change of coordinates.

### *7.2 Maximal symmetry group of* $\mathbf{iu_t} + \mathbf{u_{xx}} + \mathbf{F}(\mathbf{x}, \mathbf{t}, \mathbf{uu}^*)\mathbf{u} = \mathbf{0}$

We now determine the form of the scalar ($m = 1$) nonlinear Schrödinger equation in a single spatial dimension ($n = 1$) having the maximal dimension Lie symmetry group. The condition that $F$ is a function of $uu^*$ is imposed by adjoining the differential equation

$$u^*F_u - uF_{u^*} = 0 \tag{31}$$

to the determining system which was generated for an arbitrary form of $F$.

The *RifSimp* algorithm is applied to the determining system following the strategy described in [32] to find the forms of $F$ having a maximal dimension symmetry group. In the first step, the strategy determines whether there are any forms of $F$ having an infinite dimensional symmetry group by running *RifSimp* with a setting to reject any cases with a dimension less than $d = \infty$. It performs binary branching on the vanishing or non-vanishing of pivots (in this case differential expressions in $F$), resulting in a number of cases. Computation on a case is halted whenever it is found that the dimension of the symmetry group is bounded below $d$, and the upper bound of the dimension for that case is recorded. If no cases meeting the dimension requirement are found, then $d$ is set to the largest upper bound, and *RifSimp* is run again with this new restriction. This process is repeated until cases having a $d$ dimensional group are found.

This strategy has the advantage that it rejects lower dimensional cases, in which the investigator may not be interested. These low dimensional cases are often the most difficult to compute.

As a result of the calculation we have the following theorem.

**Theorem 1** *When $m = n = 1$, (1) admits a maximal 6 dimensional symme-*

*try group iff it has the form*

$$iu_t + u_{xx} + \left(a(t)x^2 + b(t)x + c(t) + d|u|^4\right) u = 0 \tag{32}$$

*where $a(t), b(t), c(t)$ are arbitrary functions of $t$ and $d$ is an arbitrary nonzero constant. When $m = n = 1$ the form*

$$iu_t + u_{xx} + \left(b(t)x + c(t) + d|u|^{2\sigma}\right) u = 0 \tag{33}$$

*admits a 5 dimensional symmetry group where $b(t), c(t)$ are arbitrary functions of $t$, $d$ is an arbitrary nonzero constant, and $\sigma \neq 0, 2$.*

We note that there are other forms of $F$ for the case $m = n = 1$ with 5 dimensional symmetry groups that are not discussed here.

This result is a generalization of the well-known result [3] for linear Schrödinger equations of form $iu_t + u_{xx} + F(x,t)u = 0$. In particular the result is that disregarding the infinite superposition group, such an equation has a maximal symmetry group iff $F(x,t) = a(t)x^2 + b(t)x + c(t)$. We also direct the reader to a classification of $F = f(x,t) + g(x,t)|u|^2$ given by Gagnon and Winternitz [18].

### *7.3 Generalization to VNLS systems*

We were motivated to consider higher dimensional generalizations of the results in Theorem 1, and [47]. In [47], the case where $F = |\mathbf{u}|^2$ is investigated, and we empirically obtain the formula for the dimension of the associated group $\mathcal{G}$ as

$$\dim(\mathcal{G}) = 2 + m^2 + n(n+3)/2 + \delta(n-2) \tag{34}$$

where $\delta(n-2)$ is the Kronekker-Delta function ($\delta(0) = 1$, $\delta(x) = 0$ for $x \neq 0$).

We consider systems of the form:

$$i\mathbf{u}_t + \nabla^2\mathbf{u} + \left(a(t)|\mathbf{x}|^2 + \mathbf{b}(t)\cdot\mathbf{x} + c(t) + d|\mathbf{u}|^{2\sigma}\right)\mathbf{u} = \mathbf{0}. \tag{35}$$

Calculations for low values of $m, n$, made possible through the use of *DiffElim*, led us to the class where $\sigma = \frac{2}{n}$, or

$$i\mathbf{u}_t + \nabla^2\mathbf{u} + \left(a(t)|\mathbf{x}|^2 + \mathbf{b}(t)\cdot\mathbf{x} + c(t) + d|\mathbf{u}|^{\frac{4}{n}}\right)\mathbf{u} = \mathbf{0}, \tag{36}$$

and to the table of the group dimension results below:

| Spatial Dimension (n) | Number of components (m) 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 6 | 9 | 14 | 21 |
| 2 | 9 | 12 | 17 | 24 |
| 3 | 13 | 16 | 21 | |
| 4 | 18 | 21 | 26 | |
| 5 | 24 | 27 | | |
| 6 | 31 | 34 | | |
| 7 | 39 | | | |
| 8 | 48 | | | |

**Table 1: Group dimension results for** (36)

We were able to fit the results in Table 1 to a formula, which agrees with the results of [47] for the overlapping cases with $n = 2$ and $a(t) = \mathbf{b}(t) = c(t) = 0$. This led to the following conjecture:

**Conjecture 1** *The dimension of the Lie group of symmetries of (36) is:*

$$dim(\mathcal{G}) = m^2 + n(n+3)/2 + 3. \tag{37}$$

*Moreover, this is the maximal group of symmetries of (1).*

A large portion of the 1999 book *The Nonlinear Schrödinger Equation* [43] is devoted to the nonlinear Schrödinger equation with power law nonlinearities of the form

$$iu_t + \nabla^2 u + |u|^{2\sigma} u = 0. \tag{38}$$

The existence and behavior of solutions, particularly blowup or wave collapse solutions, depends strongly on the exponent $\sigma$ [43].

In the supercritical case, $\sigma > \frac{2}{n}$ with rapidly decaying initial conditions, functional analytic methods show that blowup occurs in finite time (e.g. see Theorem 5.2, page 95, of [43] and many variations on this result). Additionally the blowup is self-similar. A similar result, requiring stronger assumptions on the initial conditions, guaranteeing blowup, also holds in the critical case $\sigma = \frac{2}{n}$. Some global existence results are known for the subcritical case $\sigma < \frac{2}{n}$.

It is interesting that the critical case, $\sigma = \frac{2}{n}$, is picked out in our results as the maximal dimension symmetry case for $n = m = 1$. This extra symmetry was noted by Taranov [44] (also see [43], pp. 33 – 37).

The class (36) is much broader than (38). It would be interesting to see if the criticality, supercriticality and subcriticality behavior of blowup solutions is retained by our extended class (36).

## 8 Statistics on speed, memory and relative performance

In this section, the family of VNLS equations (36) is used to test the our implementation of the *DiffElim* algorithm and compare it to the *RifSimp* algorithm implemented in Maple. Note that all determining systems used in this paper, and available on the web, have been obtained using Mark Hickman's Symmetry package.

### *8.1 Other systems*

We have made a number of determining systems and benchmarks available at our web sites:

- The classical Laplace equation for 3-16 spatial dimensions;
- The classical harmonic oscillator for 3-16 spatial dimensions;
- The VNLS system with potential $F = |u|^2$ for 1-4 components in 1-10 spatial dimensions;
- The VNLS system with general potential for 1-4 components in 1-10 spatial dimensions.

All but the last of the above systems are discussed in [47] and are available at our web sites. The last collection of systems has been provided for experimentation, allowing different forms of the potential to be used. This collection of systems and benchmarks will be extended, so that researchers can easily test and improve their algorithms.

## 8.2 *Statistics on the input systems*

When Hickman's Maple package *Symmetry*, was applied to the VNLS system for the general form of the potential, the numbers of determining equations it generated are listed in Table 4 below.

| Spatial Dimension (n) | Number of components (m) 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 46 | 248 | 678 | 1408 |
| 2 | 120 | 624 | 1704 | 3552 |
| 3 | 222 | 1144 | 3126 | 6528 |
| 4 | 352 | 1808 | 4944 | 10336 |
| 5 | 510 | 2616 | 7158 | 14976 |
| 6 | 696 | 3568 | 9768 | 20448 |
| 7 | 910 | 4664 | 12744 | 26752 |
| 8 | 1152 | 5904 | 16176 | 33888 |

**Table 2: Number of determining equations for the VNLS system (1)**

It was found that the number of PDE in the determining systems we generated, CountDets$(n, m)$, fit an exact polynomial in $m$ and $n$ and given by

$$\text{CountDets}(n, m) = (4n^2 + 8n)m^3 + (10n^2 + 24n + 8)m^2 - 8m. \quad (39)$$

The total length of the PDE present in the input determining systems (as measured by Maple's length function) is also recorded in Table 3 below.

These are the lengths of the determining system for the arbitrary form of the potential $F$ in (1). Substitution of the particular form of $F$ from (2) increases their size by $35\% - 60\%$.

| Spatial Dimension (n) | Number of components (m) 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 5983 | 32239 | 96003 | 218539 |
| 2 | 12967 | 69611 | 209583 | 483283 |
| 3 | 23171 | 123711 | 373143 | 863147 |
| 4 | 37027 | 196459 | 591723 | 1368499 |
| 5 | 54967 | 289775 | 870363 | 2009707 |
| 6 | 77423 | 405579 | 1214103 | 2797139 |
| 7 | 104827 | 545791 | 1627983 | 3741163 |
| 8 | 137611 | 712331 | 2117043 | 4852147 |

**Table 3: Length of determining systems for the VNLS system (1)**

Remarkably it was also possible to fit this data to a polynomial function, LengthDets$(n, m)$, that is cubic in $n$, and quartic in $m$:

$$
\begin{aligned}
\mathrm{LengthDets}(m, n) = {} & (56n^2 + 112n)m^4 \\
& + (16n^3 + 356n^2 + 876n + 616)m^3 \\
& + (40n^3 + 584n^2 + 1542n + 1204)m^2 \\
& + (16n^3 + 182n^2 + 416n - 36)m + 3.
\end{aligned}
$$

To obtain and verify this result, we also generated determining systems for $m = 5, 6$ which are not included in Tables 2 and 3. These relationships for the input equation count and total length hold *exactly* for all determining systems that were generated, which is quite surprising considering the sensitivity of Maple's length function to mild changes in the form of the input. Maple's length function gives a rough estimate of size, but also considers the length of names in the input. As an example, the expression $aa + bb$ is 11 in length, 2 for the $aa$, 2 for it's coefficient (1), 2 for the $bb$, 2 for it's coefficient, 1 for the + operation, and 1 for each item in the sum. This would mean that the lengths of $a + b + c$ and $aaa + bbb$ are both 13.

Similar results were found for the classical Laplace equations and the oscillator Schrödinger equation, and are presented in [47].

Since the input equation counts and system sizes were found to be exact polynomial functions of $m, n$, it is natural to enquire whether the same is true for the output. This was found to be the case for the equation count for all systems successfully simplified, and this relationship is given by

$$\mathrm{CountOut}(m,n) = \frac{1}{3}m^3 + \frac{5}{2}m^2 + (\frac{49}{6} + 4n)m + \frac{1}{6}n^3 + n^2 + \frac{17}{6}n + 1, \quad (40)$$

where additional systems for $m = 5, 6$ not present in the table were also used to validate the result.

The computed output equation counts are given in Table 4 below.

| Spatial Dimension (n) | Number of components (m) 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 20 | 42 | 73 | 115 |
| 2 | 31 | 57 | 92 | 138 |
| 3 | 46 | 76 | 115 | |
| 4 | 66 | 100 | 143 | |
| 5 | 92 | 130 | | |
| 6 | 125 | 167 | | |
| 7 | 166 | | | |
| 8 | 216 | | | |

**Table 4: Output Determining System Equation Counts for (36)**

The total length of PDE present in the output determining systems, as measured by Maple's length function, was also recorded, and is given in Table 5 below:

| Spatial | Number of components (m) | | | |
|---|---|---|---|---|
| Dimension (n) | 1 | 2 | 3 | 4 |
| 1 | 10312 | 55011 | 189599 | 493215 |
| 2 | 24782 | 101519 | 289161 | 673075 |
| 3 | 47177 | 166767 | 420947 | |
| 4 | 77578 | 250853 | 585057 | |
| 5 | 118698 | 359112 | | |
| 6 | 168984 | 488021 | | |
| 7 | 232633 | | | |
| 8 | 306252 | | | |

**Table 5: Output Determining System Lengths (36)**

Unfortunately it was not possible to polynomially fit the lengths of the output systems. For the dependence on $n$, it initially appears to be exponential for small $n$ values, but this pattern seems to fail for larger $n$.

It is remarkable that the output length is always larger than the input length, but what is not so obvious is the structure of the output. Though larger, it is more useful, as it provides significant additional information that the input system does not (such as the dimension and structure of the symmetry groups).

### *8.4 Comparison of running times for RifSimp and DiffElim*

We have recorded the running times for the *RifSimp* algorithm and the *DiffElim* algorithm in Table 6 below as a function of $m, n$. In all tables, time is measured in cpu seconds running on a PII 333MHz Linux PC, and memory is measured in megabytes. The *DiffElim* algorithm times are given above those for *RifSimp*. Any cases that failed due to insufficient memory, are marked by **Mem**, while cases not attempted (as the result of the failure of an easier system for the same $m$) are blank.

| Spatial Dimension (n) | Number of components (m) 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0.1<br>10.1 | 2.5<br>1446.0 | 7.1<br>15238.6 | 77.4<br>**Mem** |
| 2 | 0.2<br>29.0 | 3.6<br>709.2 | 36.1<br>18202.4 | 70.0 |
| 3 | 0.4<br>59.7 | 25.4<br>3065.7 | 106.0<br>**Mem** | **Mem** |
| 4 | 0.7<br>115.2 | 26.7<br>**Mem** | 309.5 | |
| 5 | 1.2<br>242.9 | 103.9<br>**Mem** | **Mem** | |
| 6 | 1.9<br>451.6 | 309.5 | | |
| 7 | 3.0<br>940.2 | **Mem** | | |
| 8 | 4.5<br>1572.8 | | | |

**Table 6: Running times. The (m,n) entry is: $\dfrac{\textbf{DiffElim Time}}{\textbf{RifSimp Time}}$ for reducing the determining system of (36).**

The average ratio of *DiffElim* running time, to *RifSimp* running time in the above table is about 400.

Using least squares, an exponential function of the form $\mathrm{Time}(m,n) := C\exp(an)\exp(bm)$ was approximately fitted to the above data yielding the very rough approximations

$$\begin{aligned}
&\mathrm{RifSimp}: \ \mathrm{Time}(m,n) \approx 0.3\,\mathrm{e}^{3.4n}\,\mathrm{e}^{0.7m}, \\
&\mathrm{DiffElim}: \mathrm{Time}(m,n) \approx 0.006\,\mathrm{e}^{2.4n}\,\mathrm{e}^{0.7m}.
\end{aligned}$$

We have liberally rounded the coefficients in the formulae since they are very rough and based on few data points. Indeed the standard deviation of the linear fit of the log of the run time to the linear relation $an + bm + \ln(C)$ here is 0.96 for *DiffElim*, and 0.40 for *RifSimp*.

### *8.5 Comparison of Memory Use for RifSimp and DiffElim*

The memory use for the *RifSimp* algorithm and the *DiffElim* algorithm, as a function of $m, n$, is shown in the table below. Again, any cases that failed due to insufficient memory, are marked by **Mem**, while cases not attempted are blank.

| Spatial | Number of components (m) | | | |
|---|---|---|---|---|
| Dimension (n) | 1 | 2 | 3 | 4 |
| 1 | 0.8<br>3.1 | 1.7<br>24.1 | 2.6<br>51.9 | 16.3<br>**Mem** |
| 2 | 0.8<br>3.9 | 2.2<br>12.7 | 10.5<br>201.0 | 10.4 |
| 3 | 0.9<br>4.6 | 6.3<br>31.2 | 23.5<br>**Mem** | **Mem** |
| 4 | 1.1<br>6.1 | 6.3<br>**Mem** | 19.9 | |
| 5 | 1.3<br>9.2 | 29.8 | **Mem** | |
| 6 | 1.5<br>11.4 | 48.3 | | |
| 7 | 1.9<br>16.0 | **Mem** | | |
| 8 | 2.5<br>20.9 | | | |

**Table 7: Memory usage The (m,n) entry is: $\dfrac{\textbf{DiffElim Mem}}{\textbf{RifSimp Mem}}$ for reducing the determining system of (36).**

The average ratio of *DiffElim* memory use, to *RifSimp* memory use in the above table is about 9.

We also fitted, approximately using least squares, and exponential function of the form, $\mathrm{Mem}(m,n) := C\exp(an)\exp(bm)$ to the above data, obtaining the very rough approximation:

$$
\begin{aligned}
\mathrm{RifSimp}: \mathrm{Mem}(m,n) &\approx 0.4\,\mathrm{e}^{1.7n}\,\mathrm{e}^{0.3m},\\
\mathrm{DiffElim}: \mathrm{Mem}(m,n) &\approx 0.1\,\mathrm{e}^{1.2n}\,\mathrm{e}^{0.3m}
\end{aligned}
$$

Once again the usual cautionary remarks should be made about the above fits.

## 9 Discussion

For the class of problems studied, the implementation of the *DiffElim* algorithm in the new *CDiffElim* environment outperforms *RifSimp* on average by a factor of 400, using significantly less memory in the process.

The *CDiffElim* environment has an equation data structure well suited to differential elimination, enhanced memory management, and fast algorithms for standard operations on ordered equations. We hope that the description given in this paper helps researchers in coding special purpose algorithms for their particular application areas. It was found that the enhanced GCD implementation described in section 4.3.4 made a significant difference for the more difficult systems in the class of problems used for the benchmark.

The *DiffElim* algorithm was designed to be flexible and still retain its efficiency, in order to allow the differential elimination algorithms of other researchers to be easily coded as a modification of this algorithm. We also provided a number of classes of determining systems for other researchers to use in testing their algorithms.

A broad class of vector nonlinear Schrödinger system having a high dimension Lie symmetry groups was explored, and the relationship of this class to physical phenomena described by other researchers was discussed.

Finally, we included a description of a new hybrid symbolic-numeric algorithm that offers an alternative approach to classical differential elimination. This probabilistic method obtains information from a determining system using differentiation and numerical linear algebra, but not symbolic elimination.

The *CDiffElim* environment is still in a developmental stage, with many algorithms still needed (such as multivariate factorization). Much work remains to be done on interfacing and improving portability (currently only implemented under Linux).

Future work includes:

- Implementation of an algorithm in the *CDiffElim* environment to handle fully nonlinear systems;
- Modification of the *CDiffElim* environment to address the missing algorithm, interface, and portability issues;
- Significant extension of the benchmarks available on our web sites;
- Further work on the hybrid symbolic-numeric approach.

## References


[1] D. J. Benney and G. J. Roskes, *Wave Instabilities*, Stud. in Appl. Math., 48 (1969), pp. 377–385.

[2] A. L. Berkhoer and V. E. Zakharov, *Self excitation of waves with different polarizations in nonlinear media*, Sov. Phys. JETP, 31 (1970), pp. 486–490.

[3] G. W. Bluman and S. Kumei, *Symmetries and Differential equations*, vol. 81 of Applied Mathematical Sciences, Springer, 1989.

[4] F. Boulier, *Some improvements of a lemma of rosenfeld.* Preprint, 1998.

[5] F. Boulier, D. Lazard, F. Ollivier, and M. Petitot, *Representation for the radical of a finitely generated differential ideal*, in Proc. ISSAC 1995, ACM Press, 1995, pp. 158–166.

[6] K. Brenan, S. Campbell, and L. Petzold, *Numerical Solutions of Initial-Value Problems in Differential-Algebraic Equations*, Elsevier, 1989.

[7] W. Brown, *On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors*, J. of the ACM, 18 (1971), pp. 476–504.

[8] B. Buchberger, *Gröbner Bases: An algorithmic method in polynomial ideal theory*, in Multidimensional Systems Theory, D. Reidel Publ. Company, 1985, pp. 184–232.

[9] G. Carrà Ferro and W. Sit, *On term-orderings and rankings*, in Computational Algebra, vol. 151 of Lecture Notes in Pure and Applied Mathematics, Marcel Dekker, 1994, pp. 31–77.

[10] P. Clarkson, *Dimensional reductions and exact solutions of a generalized nonlinear Schrödinger equation*, Nonlinearity, 5 (1992), pp. 453–472.

[11] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, MIT Press, 1990.

[12] M. Fels and P. Olver, *Moving coframes I. A practical algorithm*, Acta Appl. Math., 51 (1998), pp. 161–213.

[13] L. Gagnon, *Self-similar solutions for a coupled system of nonlinear Schrödinger equations*, J. Phys. A: Math. and Gen., 25 (1992), pp. 2649–2667.

[14] L. Gagnon and P. Winternitz, *Lie symmetries of a generalised nonlinear Schrödinger equation: II. exact solutions*, J. Phys. A: Math. and Gen., 22 (1989), pp. 469–497.

[15] G. Golub and C. V. Loan, *Matrix Computations*, John Hopkins U. Press, 3rd ed., 1996.

[16] R. Grimshaw, *Slowly varying solitary waves. II. Nonlinear Schrödinger equation*, Proc. Roy. Soc. London. Ser. A, 368 (1979), pp. 377–388.

[17] D. Hartley and R. Tucker, *A constructive implementation of the Cartan–Kähler theory of exterior differential systems*, J. Symb. Comp., 12 (1991), pp. 655–667.

[18] N. H. Ibragimov, ed., *CRC Handbook of Lie Group Analysis of Differential Equations*, vol. I,II,III, CRC Press, Boca Raton, 1994,1995,1996.

[19] M. Janet, *Sur les systèmes d'équations aux dérivées partielles*, J. de Math. (8), 3 (1920), pp. 65–151.

[20] D. Knuth, *The Art of Computer Programming: Fundamental Algorithms*, vol. 1, Addison Wesley Longman, 3rd ed., 1997.

[21] ——, *The Art of Computer Programming: Seminumerical Algorithms*, vol. 2, Addison Wesley Longman, 3rd ed., 1997.

[22] E. Mansfield, *Differential Gröbner Bases*, PhD thesis, Univ. of Sydney, 1991.

[23] E. Mansfield, G. Reid, and P. Clarkson, *Nonclassical reductions of a 3+1-cubic nonlinear Schrödinger system*, Comp. Phys. Comm., 115 (1998), pp. 460–488.

[24] M. Monagan and A. Wittkopf, *On the design and implementation of Brown's algorithm over the integers and number fields*, in Proc. ISSAC '2000, St. Andrews, ACM Press, 2000, pp. 215–223.

[25] M. Oh and C. Pantelides, *A modelling and simulation language for combined lumped and distributed parameter systems*, Computers Chem. Eng., 20 (1996), pp. 611–633.

[26] P. Olver, *Applications of Lie Groups to Differential Equations*, vol. 107 of Graduate Texts in Mathematics, Springer, 2nd ed., 1993.

[27] ——, *Equivalence, Invariants, and Symmetry*, Graduate Texts in Mathematics, Cambridge University Press, 1995.

[28] O. M. Phillips, *The Dynamics of the Upper Ocean*, Cambridge University Press, 1977.

[29] J.-F. Pommaret, *Systems of Partial Differential Equations and Lie Pseudogroups*, Gordon and Breach, 1978.

[30] G. Reid, *Algorithms for reducing a system of PDEs to standard form, determining the dimension of its solution space and calculating its Taylor series solution*, European J. of Appl. Math., 2 (1991), pp. 293–318.

[31] G. Reid, I. Lisle, A. Boulton, and A. Wittkopf, *Algorithmic determination of commutation relations for Lie symmetry algebras of PDEs*, in Proc. ISSAC '92, Berkeley, P. Wang, ed., ACM Press, 1992, pp. 63–68.

[32] G. Reid and A. Wittkopf, *Determination of maximal symmetry groups of classes of differential equations*, in Proc. ISSAC '2000, St. Andrews, ACM Press, 2000, pp. 264–272.

[33] G. Reid, A. Wittkopf, and A. Boulton, *Reduction of systems of nonlinear partial differential equations to simplified involutive forms*, European J. of Appl. Math., 7 (1996), pp. 635–666.

[34] C. Riquier, *Les systèmes d'équations aux dérivées partielles*, Gauthier-Villars, 1910.

[35] C. Rust, *Rankings of Derivatives for Elimination Algorithms and Formal Solvability of Analytic Partial Differential Equations*, PhD thesis, Univ. Chicago, 1998. www-address: www.cecm.sfu.ca/~rust.

[36] C. Rust and G. Reid, *Rankings of partial derivatives*, in Proc. ISSAC '97, Maui, W. Küchlin, ed., acm Press, 1997, pp. 9–16.

[37] C. Rust, G. Reid, and A. Wittkopf, *Existence and uniqueness theorems for formal power series solutions of analytic differential systems*, in Proc. ISSAC '99, Vancouver, S. Dooley, ed., acm Press, 1999, pp. 105–112.

[38] F. Schwarz, *Reduction and completion algorithms for partial differential equations*, in Proc. ISSAC '92, Berkeley, P. Wang, ed., acm Press, 1992, pp. 49–56.

[39] A. Sedoglavic, *A mixed symbolic-numeric method to study prime ordinary differential ideal.* Preprint, www.medicis.polytechnique.fr/gage/notes/2000.html, 2000.

[40] W. Seiler, *Analysis and application of the formal theory of partial differential equations*, PhD thesis, Lancaster University, 1994.

[41] A. Sommese, J. Verschelde, and C. Wampler, *Numerical homotopies to compute generic points on positive dimensional algebraic sets*, J. of Complexity, 16 (2000), pp. 572–602.

[42] ———, *Numerical irreducible decomposition using projections from points on the components.* Preprint, Department of Mathematics, Statistics and Computer Science, University of Illinois at Chicago, 2000.

[43] C. Sulem and P. L. Sulem, *The Nonlinear Schrödinger Equation, Self-Focussing and Wave Collapse*, vol. 139 of Applied Mathematical Sciences, Springer, 1999.

[44] V. Talanov, *Focusing of light in cubic media*, JETP Lett, 11 (1970), pp. 199–201.

[45] L. Trefethen and D. B. III, *Numerical Linear Algebra*, SIAM, Philadelphia, 1997.

[46] A. Wittkopf, *Efficient implementation of differential elimination algorithms*, PhD thesis, Simon Fraer University, 2001.

[47] A. Wittkopf and G. Reid, *Fast differential elimination algorithms*, Technical Report, Ontario Research Centre for Computational Algorithms, (2000).

[48] T. Wolf, *A package for the analytic investigation and exact solutions of differential equations*, in Proc. EUROCAL '87, vol. 378, Springer, 1987, pp. 479–491.

[49] R. Zippel, *Probabilistic Algorithms for Sparse Polynomials*, PhD thesis, Massachusetts Inst. of Technology, 1979.