Vladimir P. Gerdt
Wolfram Koepf
Ernst W. Mayr
Evgenii V. Vorozhtsov (Eds.)

LNCS 6244

Computer Algebra in Scientific Computing

12th International Workshop, CASC 2010
Tsakhkadzor, Armenia, September 2010
Proceedings

Springer

Vladimir P. Gerdt Wolfram Koepf
Ernst W. Mayr Evgenii V. Vorozhtsov (Eds.)

Computer Algebra in Scientific Computing

12th International Workshop, CASC 2010
Tsakhkadzor, Armenia, September 6-12, 2010
Proceedings

Springer

Volume Editors

Vladimir P. Gerdt
Joint Institute for Nuclear Research
141980 Dubna, Moscow region, Russia
E-mail: gerdt@jinr.ru

Wolfram Koepf
Universität Kassel
34109 Kassel, Germany
E-mail: koepf@mathematik.uni-kassel.de

Ernst W. Mayr
Technische Universität München
85748 Garching, Germany
E-mail: mayr@in.tum.de

Evgenii V. Vorozhtsov
Russian Academy of Sciences
Novosibirsk, 630090, Russia
E-mail: vorozh@itam.nsc.ru

Library of Congress Control Number: 2010932428

CR Subject Classification (1998): I.1, G.2, I.3.5, F.2, G.1, F.2.2

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-642-15273-2 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-15273-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Dedicated to

Prof. Ernst W. Mayr

on the occasion of his 60th birthday

Preface

The CASC Workshops are traditionally held in turn in the Commonwealth of Independent States (CIS) and outside CIS (Germany in particular, but, at times, also other countries with lively CA communities). The previous CASC Workshop was held in Japan, and the 12th workshop was held for the first time in Armenia, which is one of the CIS republics. It should be noted that more than 35 institutes and scientific centers function within the National Academy of Sciences of Armenia (further details concerning the structure of the academy can be found `http://www.sci.am`). These institutions are concerned, in particular, with problems in such branches of natural science as mathematics, informatics, physics, astronomy, biochemistry, etc. It follows from the talks presented at the previous CASC workshops that the methods and systems of computer algebra may be applied successfully in all the above-listed branches of natural sciences. Therefore, the organizers of the 12th CASC Workshop hope that the present workshop will help the Armenian scientists to become even more familiar with the capabilities of advanced computer algebra methods and systems and to get in touch with specialists in computer algebra from other countries.

The 11 earlier CASC conferences, CASC 1998, CASC 1999, CASC 2000, CASC 2001, CASC 2002, CASC 2003, CASC 2004, CASC 2005, CASC 2006, CASC 2007, and CASC 2009 were held, respectively, in St. Petersburg (Russia), Munich (Germany), Samarkand (Uzbekistan), Konstanz (Germany), Yalta (Ukraine), Passau (Germany), St. Petersburg (Russia), Kalamata (Greece), Chişinău (Moldova), Bonn (Germany), and Kobe (Japan), and they all proved to be very successful.

The present volume contains revised versions of the papers submitted to the workshop by the participants and accepted by the Program Committee after a thorough reviewing process (each paper was reviewed by at least three referees).

The studies in Gröbner bases and their applications belong to traditional themes of the CASC Workshops. In particular, a new robust method is presented for an accurate floating-point computation of Gröbner bases, which is stable to error accumulation. The application of Gröbner bases to the solution of polynomial equations arising at the solution of the problem of KL-divergence minimization is presented.

The invited talk by E.W. Mayr surveys a number of relationships between computer algebra (in particular, polynomial ideals or, more specifically, binomial ideals) and concepts like Petri nets widely used in computer science for modeling and analyzing concurrent systems, and also presents some new complexity results and algorithms for restricted classes of polynomial ideals.

Another traditional topic of the CASC Workshop, polynomial algebra, is represented by contributions devoted to the construction of irreducible polynomials over finite fields, multivariate homogeneous polynomial decomposition, greatest common divisor (GCD) computations for finding universal denominators, iterative calculation of the approximate GCD for multiple univariate polynomials, and the REDUCE-based investigation of the convexity of rational functions over a polyhedral domain by reducing convexity questions to real quantifier elimination problems.

Two papers deal with the theory of matrices and its application. In one of them, the algorithms for a fast generalized Bruhat decomposition of the matrix and for the computation of the inverse matrix are presented. In the other paper, the minimal faithful matrix representation of filiform Lie algebras is computed with *Maple*.

Several papers are devoted to the investigation, with the aid of computer algebra, of various topics related to the ordinary differential equations (ODEs): symbolic solution of a third-order ODE, integrability of planar ODE systems near a degenerate stationary point, the use of differential resultants to investigate completely integrable dynamical systems, and derivation of new numerical methods for stiff ODE systems.

Investigating oscillations for parametric ODEs has many applications in science and engineering but is a very hard problem. The invited lecture by A. Weber presents a review of some recently developed criteria which give *sufficient conditions* to exclude oscillations by reducing them to problems on semi-algebraic sets—for polynomial vector fields. Some examples are given, and possible future work in the form of problems to be solved is discussed. Some of these problems might be rather immediate to be solved, some others might pose major challenges.

Two papers handle the topic of partial differential equations (PDEs): disjoint decomposition of nonlinear PDE systems of equations and inequations into so-called simple subsystems, and derivation of semigroup identities for evolution equations using CAS.

Several papers are devoted to software problems in computer algebra. One of them deals with the problem of achieving high performance when both symbolic and numerical computations are required, and it proposes using the Aldor programming language to solve this problem. Two other papers are devoted to the problem of the development of object-oriented computer algebra software and to functional parallelization of rational multiple-residue arithmetic.

A number of papers deal with the application of symbolic or symbolic-numerical computations in applied problems of physics, mechanics, and engineering: computer analysis of spheroidal quantum dot models, the use of symbolic computations in particle accelerator physics, reduction of nonlinear Lagrange systems with cyclic coordinates to the linear Routh systems with the aid of the Legendre transformation, and the use of geometric probabilities to model the self-healing process in concrete with the aid of capsules containing the healing agent.

The survey “Computational Science in Armenia” by H. Marandjian and Yu. Shoukourian is devoted to the development of informatics and computer science in Armenia. The results in theoretical computer science (algebraic models, solutions to systems of general form recursive equations, the methods of coding theory, pattern recognition, and image processing), constitute the theoretical basis for developing problem-solving-oriented environments. As examples can be mentioned: a synthesizer of optimized distributed recursive programs, software tools for cluster-oriented implementations of two-dimensional cellular automata, and a grid-aware Web interface with advanced service trading for linear algebra calculations. In the direction of solving scientific problems that require high-performance computing resources, examples of completed projects include the field of physics (parallel computing of complex quantum systems), astrophysics (Armenian virtual laboratory), biology (molecular dynamics study of human red blood cell membrane), and meteorology (implementing and evaluating the Weather Research and Forecast Model for the territory of Armenia). The overview also notes that the Institute for Informatics and Automation Problems of the National Academy of Sciences of Armenia has established a scientific and educational infrastructure uniting computing clusters of scientific and educational institutions of the country and provides the scientific community with access to local and international computational resources that is a strong support for computational science in Armenia.

Our particular thanks are due to the members of the CASC 2010 local Organizing Committee in Armenia, V. Sahakyan and M. Haroutunyan (The Institute for Informatics and Automation Problems; National Academy of Sciences of Armenia, Yerevan), who ably handled local arrangements in Yerevan and Tsakhkadzor. Furthermore, we want to thank the PC Committee for their invaluable work. Finally, we are grateful to W. Meixner for his technical help in the preparation of the camera-ready manuscript for this volume.

June 2010

V.P. Gerdt
W. Koepf
E.W. Mayr
E.V. Vorozhtsov

Organization

CASC 2010 was organized jointly by the Department of Informatics at the Technische Universität München, Germany, and the Institute for Informatics and Automation Problems (IIAP) at Yerevan, Armenia.

Workshop General Chairs

Vladimir Gerdt (JINR, Dubna)   Ernst W. Mayr (TU München)

Program Committee Chairs

Wolfram Koepf (Kassel, Chair)   Evgenii Vorozhtsov (Novosibirsk, Co-chair)

Program Committee

Sergei Abramov (Moscow)
Alkis Akritas (Volos)
Gerd Baumann (Cairo)
Hans-Joachim Bungartz (Munich)
Andreas Dolzmann (Saarbrücken)
Victor Edneral (Moscow)
Ioannis Emiris (Athens)
Jaime Gutierrez (Santander)
Richard Liska (Prague)
Marc Moreno Maza (London, Ontario)
Markus Rosenkranz (Canterbury)
Mohab Safey El Din (Paris)
Yosuke Sato (Tokyo)
Werner Seiler (Kassel)
Doru Stefanescu (Bucharest)
Serguei P. Tsarev (Krasnoyarsk)
Andreas Weber (Bonn)
Eva Zerz (Aachen)

Local Organization

Vladimir Sahakyan (IIAP, Yerevan)
Mariam Haroutunyan (IIAP, Yerevan)

Publicity Chair

Victor G. Ganzha (Munich)

Website

`http://wwwmayr.in.tum.de/CASC2010`

Table of Contents

Construction of Irreducible Polynomials over Finite Fields

Sergey Abrahamyan

Institute of Informatics and Automation Problems,
P. Sevak street 1, Yerevan 0014, Armenia
serj.abrahamyan@gmail.com

Abstract. The aim of this paper is to present an explicit construction of families of irreducible polynomials over finite fields by applying a polynomial composition method.

1 Introduction

The problem of irreducibility of polynomials over finite fields is a case of special interest and plays an important role in modern engineering [1,2,3]. One of the methods to construct irreducible polynomials is the polynomial composition method that allows constructions of irreducible polynomials of higher degree from given irreducible polynomials over finite fields. In this paper we state a theorem that enables explicit constructions of irreducible polynomials of degree pm starting from an irreducible polynomial of degree m over $\mathbb{F}_q$. As an example of the theorem we give an explicit construction of a family of irreducible polynomials over $\mathbb{F}_4$. We show that in the proposed construction the number of operations required to obtain an irreducible polynomial of degree $m \cdot p$ from a given monic irreducible polynomial of degree m over $\mathbb{F}_q$ is $O(m^2)$,where m is the degree of the starting polynomial, and p is the characteristic of the field.

2 Preliminaries

Let $\mathbb{F}_q$ be the Galois field of order $q = p^n$, where p is a prime and n is a natural number, and let $f(x)$ be an irreducible polynomial of degree m over $\mathbb{F}_q$. For $\alpha \in \mathbb{F}_{q^m}$ the trace $Tr_{\mathbb{F}_{q^m}/\mathbb{F}_q}(\alpha)$ of α over $\mathbb{F}_q$ is defined by

$$Tr_{\mathbb{F}_{q^m}/\mathbb{F}_q}(\alpha) = \alpha + \alpha^q + \cdots + \alpha^{q^{m-1}}.$$

If $\mathbb{F}_q$ is the prime subfield of $\mathbb{F}_{q^m}$, then $Tr_{\mathbb{F}_{q^m}/\mathbb{F}_q}(\alpha)$ is called the absolute trace of α and is simply denoted by $Tr_{\mathbb{F}_{q^m}}(\alpha)$.

We will begin with recalling some basic results on the irreducibility of polynomials that can be found, for example, in [2].

Proposition 1 ([2], Theorem 3.78). *Let $\alpha \in \mathbb{F}_q$ and let p be the characteristic of $\mathbb{F}_q$. Then the trinomial $x^p - x - \alpha$ is irreducible in $\mathbb{F}_q[x]$ if and only if it has no root in $\mathbb{F}_q$.*

Proposition 2 ([2], Corollary 3.79). *With the notation of Theorem 3.78, the trinomial $x^p - x - \alpha$ is irreducible in $\mathbb{F}_q[x]$ if and only if $Tr_{\mathbb{F}_q}(\alpha) \neq 0$.*

With these preliminaries, we state a theorem that yields an irreducible polynomial of degree pm over $\mathbb{F}_q$.

Theorem. *Let $f(x) = x^m + a_1x^{m-1} + \cdots + a_m$ be an irreducible polynomial of degree m over $\mathbb{F}_q$ and $q = p^n$. Then, if $a_1 + a_1^p + \cdots + a_1^{p^{n-1}} \neq 0$, the polynomial $a_m^{-1}(1 - x^{p-1})^m f^*\left(\frac{x^p}{1-x^{p-1}}\right)$, where $f^*(x) = x^m f\left(\frac{1}{x}\right)$, is an irreducible polynomial of degree pm over $\mathbb{F}_q$.*

Proof. Let α be a root of $f(x) = 0$ in some extension field $\mathbb{F}_{q^m}$ of $\mathbb{F}_q$. So the elements $\alpha, \alpha^q, \cdots, \alpha^{q^{m-1}}$ are the roots of $f(x) = 0$ and $Tr_{\mathbb{F}_{q^m}/\mathbb{F}_q}(\alpha) = \alpha + \alpha^q + \cdots + \alpha^{q^{m-1}} = -a_1$, $Tr_{\mathbb{F}_{q^m}/\mathbb{F}_p}(\alpha) = Tr_{\mathbb{F}_q/\mathbb{F}_p}\left(Tr_{\mathbb{F}_{q^m}/\mathbb{F}_q}(\alpha)\right) = Tr_{\mathbb{F}_q/\mathbb{F}_p}(-a_1) = -Tr_{\mathbb{F}_q/\mathbb{F}_p}(a_1) \neq 0$. Thus, according to Corollary 3.79 in [2] the polynomial $g(x) = x^p - x - \alpha$ is irreducible over $\mathbb{F}_q$. It is easy to see that the polynomial $-\alpha^{-1}g^*(x) = -\alpha^{-1}x^p g\left(\frac{1}{x}\right) = -\alpha^{-1}(1 - x^{p-1} - \alpha x^p)$ is also irreducible over $\mathbb{F}_{q^m}$. Because $\alpha, \alpha^q, \cdots, \alpha^{q^{m-1}}$ are the roots of $f(x) = 0$ in $\mathbb{F}_{q^m}$, we have that the elements $\alpha^{-1}, \alpha^{-q}, \cdots, \alpha^{-q^{m-1}}$ are the roots of $f^*(x)$ in $\mathbb{F}_{q^m}$. Hence

$$a_m^{-1} f^*(x) = \prod_{u=0}^{m-1} \left(x - \alpha^{-q^u}\right).$$

Substituting $\frac{x^p}{1-x^{p-1}}$ by x in the above expression and multiplying both sides of it by$(1 - x^{p-1})^m$, we have

$$a_m^{-1}\left(1 - x^{p-1}\right)^m \cdot f^*\left(\frac{x^p}{1 - x^{p-1}}\right) = \prod_{u=0}^{m-1}\left(x^p - \alpha^{-q^u}\left(1 - x^{p-1}\right)\right).$$

As $-\alpha^{-1}g^*(x) = x^p + \alpha^{-1}x^{p-1} - \alpha^{-1}$ is irreducible over $\mathbb{F}_{q^m}$, then, by Theorem 3.7 in [3], the polynomial $a_m^{-1}\left(1 - x^{p-1}\right)^m \cdot f^*\left(\frac{x^p}{1-x^{p-1}}\right)$ is irreducible over $\mathbb{F}_q$. □

Analytical results indicate that in the proposed construction, the number of operations needed to generate an irreducible polynomial $\sum_{i=0}^m a_m^{-1}a_i x^{ip}\left(1 - x^{p-1}\right)^{m-i} = \sum_{u=0}^{pm} b_u x^u$ from a given monic irreducible polynomial $\sum_{i=0}^m a_i x^{m-i}$ over $\mathbb{F}_q$ is $O(m^2)$.

The following example is an application of our theorem.

Example. Given $\mathbb{F}_4 = 0, 1, \alpha, \alpha + 1$, where α is a root of irreducible polynomial $x^2 + x + 1 \in \mathbb{F}_2[x]$ and $f(x) = x^3 + \alpha x^2 + 1$ is irreducible over $\mathbb{F}_4$. Since the polynomial $f(x) = x^3 + \alpha x^2 + 1$ satisfies the conditions of the theorem mentioned above, we may use $f(x)$ to generate a new irreducible polynomial over $\mathbb{F}_4$. So we have

$$\begin{aligned}(1+x)^3 f^*\left(\frac{x^2}{(1-x)}\right) &= (1+x)^3\left(\left(\frac{x^2}{(1+x)}\right)^3 + \alpha\left(\frac{x^2}{(1+x)}\right) + 1\right)\\ &= x^6 + \alpha x^2(1+x)^2 + (1+x)^3\\ &= x^6 + \alpha x^4 + x^3 + (\alpha+1)x^2 + x + 1.\end{aligned}$$

Thus,

$$g(x) = x^6 + \alpha x^4 + x^3 + (\alpha+1)x^2 + x + 1$$

is an irreducible polynomial over $\mathbb{F}_4$.

References

1. Albert, A.: Fundamental Concepts of Higher Algebra. University of Chicago Press, Chicago (1956)
2. Lidl, R., Niederreiter, H.: Finite Fields. Cambridge University Press, Cambridge (1987)
3. Menezez, A., Blake, I., Gao, X., Mullin, R., Vanstone, S., Yaghoobian, T.: Applications of Finite Fields. Kluwer Academic Publishers, Boston (1993)

Factorization of Polynomials and GCD Computations for Finding Universal Denominators*

S.A. Abramov[1], A. Gheffar[2], and D.E. Khmelnov[1]

[1] Computing Centre of the Russian Academy of Sciences, Vavilova, 40, Moscow 119991, GSP-1 Russia
sergeyabramov@mail.ru, dennis_khmelnov@mail.ru

[2] Institute XLIM, Université de Limoges, CNRS, 123, Av. A. Thomas, 87060 Limoges cedex, France
f_gheffar@yahoo.fr

Abstract. We discuss the algorithms which, given a linear difference equation with rational function coefficients over a field k of characteristic 0, compute a polynomial $U(x) \in k[x]$ (a universal denominator) such that the denominator of each of rational solutions (if exist) of the given equation divides $U(x)$. We consider two types of such algorithms. One of them is based on constructing a set of irreducible polynomials that are candidates for divisors of denominators of rational solutions, and on finding a bound for the exponent of each of these candidates (the full factorization of polynomials is used). The second one is related to earlier algorithms for finding universal denominators, where the computation of gcd was used instead of the full factorization. The algorithms are applicable to scalar equations of arbitrary orders as well as to systems of first-order equations.

A complexity analysis and a time comparison of the algorithms implemented in Maple are presented.

1 Introduction

In the early 1990s, computer algebra researchers and programmers tried not to use the complete (full) factorization of polynomials unless it was inevitable since this operation was very costly. Designing an algorithm everybody tried to find a suitable type of incomplete factorization based on computation of the greatest common divisors (gcd's) following classical samples of M.V.Ostrogradsky's and Ch.Hermite's algorithms for extracting the rational part of an indefinite integral of rational function. But later the situation with full factorization algorithms changed. Currently very fast and practical algorithms have become known, — see, e.g., [16]. Of course the complexity of the algorithms for the full factorization grows faster than the complexity of the algorithms for computing gcd when polynomial degrees tend to infinity. But when the degrees are of moderate size

* Supported by ECONET grant 21315ZF.

the full factorization is not costlier than the computation of gcd, e.g., in Maple system [22]. Thus, an interesting general problem arises, namely the problem of designing new alternative computer algebra algorithms based on the full factorization instead of numerous calls for the gcd subroutine. The appropriateness of such alternative algorithms has to be carefully investigated for any particular relevant computer algebra problem. Such investigation must be supported by suitable correct experiments.

In this paper, we revisit a problem related to the search for rational solutions of a linear difference equation with polynomial coefficients. Rational solutions may be a building block for other types of solutions, and more general, such algorithms may be a part of various computer algebra algorithms (see [21], [8], [9], [17], etc.). As a consequence, investigations of new ways to construct such solutions are quite valuable for computer algebra.

Let k be a field of characteristic 0. We consider systems of the form

$$Y(x+1) = A(x)Y(x), \tag{1}$$

$Y(x) = (Y_1(x), Y_2(x), \dots, Y_n(x))^T$, $A(x) = (a_{ij}(x)) \in \mathrm{Mat}_n(k(x))$. It is assumed that there exists the inverse matrix $A^{-1}(x) = (\tilde{a}_{ij}(x)) \in \mathrm{Mat}_n(k(x))$. If an inhomogeneous system $Y(x+1) = A(x)Y(x) + G(x)$ is given and $A(x)$ is as in (1), $G(x) \in k(x)^n$, then by adding to $Y(x)$ an $(n+1)$-st component with value 1, one can transform the given system into a homogeneous system with an invertible matrix $B(x) \in \mathrm{Mat}_{n+1}(k(x))$ (see, e.g., [15, Sect. 2.2]). For this reason we restrict our consideration to (1). At the same time we will consider scalar equations of the form

$$y(x+n) + a_{n-1}(x)y(x+n-1) + \dots + a_1(x)y(x+1) + a_0(x)y(x) = \varphi(x), \tag{2}$$

$\varphi(x), a_1(x), \dots, a_{n-1}(x) \in k(x)$, $a_0(x) \in k(x) \setminus \{0\}$, and such an equation is inhomogeneous if $\varphi(x)$ is a non-zero rational function. By clearing denominators we can rewrite (2) as

$$b_n(x)y(x+n) + \dots + b_1(x)y(x+1) + b_0(x)y(x) = \psi(x), \tag{3}$$

$\psi(x), b_1(x), \dots, b_{n-1}(x) \in k[x]$, $b_0(x), b_n(x) \in k[x] \setminus \{0\}$.

Currently, a few algorithms for finding rational (i.e., rational function) solutions of equations (2), (3) and systems (1) are known. The algorithms from [5,6,11,14] first construct a *universal denominator*, i.e., a polynomial $U(x)$ such that in the scalar case an arbitrary rational solution $y(x)$ of (2) or (3) can be represented as

$$y(x) = \frac{z(x)}{U(x)}, \tag{4}$$

where $z(x) \in k[x]$ (in other words, if (2) has a rational solution $\frac{f(x)}{g(x)}$ which is in the lowest terms then $g(x)|U(x)$). In the case of a system an arbitrary rational solution of (1) can be represented as

$$Y_i(x) = \frac{Z_i(x)}{U(x)}, \quad i = 1, 2, \dots, n, \tag{5}$$

where $Z_1(x), Z_2(x), \dots, Z_n(x) \in k[x]$.

The algorithm from [14] is based on constructing a set of irreducible polynomials that are candidates for divisors of denominators of rational solutions, and on finding in a quite simple way a bound for the exponent of each of these candidates. Such algorithms use the full factorization of polynomials. Experiments with the Maple system show that the full factorization makes some of computer algebra algorithms significantly faster in comparison with algorithms based on computations of gcd's and resultants ([20], [10] etc.).

When a universal denominator is constructed, one can substitute (4), (5) with undetermined $z(x)$ resp. $Z_i(x)$ into the initial equation resp. system to reduce the problem of searching for rational solutions to the problem of searching for polynomial solutions. After this, e.g., the algorithms from [2,7] (the scalar case; see also [13, Sect. 9]) and the corresponding algorithm from [6,11,18] (the case of a system) can be used.

The algorithm from [15] is applicable to the system (1) when $k = \mathbb{C}$. It finds n rational functions $R_1(x), R_2(x), \dots, R_n(x) \in \mathbb{C}(x)$ which are called *bounds for denominators* such that for any rational solution of (1) we have

$$Y_i(x) = Z_i(x)R_i(x), \quad i = 1, 2, \dots, n, \tag{6}$$

where $Z_1(x), Z_2(x), \dots, Z_n(x) \in \mathbb{C}[x]$ (the numerator of $R_i(x)$ is a factor of the numerator of the ith entry $Y_i(x)$ of any rational solution $Y(x)$). The substitution (6) is used instead of (4), (5). The algorithm from [15] can lead to a more "productive" substitution. But the general situation is not so simple. This algorithm is based on matrix operations (matrix entries are in $\mathbb{C}(x)$) which are costly. It is shown in [14, Th. 2] that there exist such examples when substitutions (5), (6) are identical, but the algorithm from [15], spends much more time than the algorithms from [5,6,11,14].

In this paper we concentrate on the approach discussed in [14].

The paper is organized as follows. Section 2 is devoted to a theoretical basis for algorithms for constructing universal denominators (a short review). Section 3 contains descriptions of the algorithm from [5,6,11,14]. In addition, we propose an improved version of the algorithm from [14]. In Section 4 we give some analysis of these algorithms and prove that all of them give the same universal denominator. A complexity analysis is given as well. In Section 5, we discuss our implementation of the proposed improved version of the algorithm from [14]. Section 6 contains a time comparison of this algorithm with the algorithms from [5,6] which are exploited in current versions of Maple. Finally in Section 7 we make some conclusion remarks.

2 The Dispersion Set

Working with polynomial and rational functions over k we will write $f(x) \perp g(x)$ for $f(x), g(x) \in k[x]$ to indicate that $f(x)$ and $g(x)$ are coprime; if $F(x) \in k(x)$, then $\operatorname{den} F(x)$ is the monic polynomial from $k[x]$ such that $F(x) = \frac{f(x)}{\operatorname{den} F(x)}$ for some $f(x) \in k[x], f(x) \perp \operatorname{den} F(x)$. In this case we write $\operatorname{num} F(x)$ for $f(x)$.

The set of monic irreducible polynomials of $k[x]$ will be denoted by $\mathrm{Irr}(k[x])$. If $p(x) \in \mathrm{Irr}(k[x])$, $f(x) \in k[x]$, then we define the valuation $\mathrm{val}_{p(x)} f(x)$ as the maximal $m \in \mathbb{N}$ such that $p^m(x) | f(x)$ ($\mathrm{val}_{p(x)} 0 = \infty$), and $\mathrm{val}_{p(x)} F(x) = \mathrm{val}_{p(x)}(\mathrm{num}\, F(x)) - \mathrm{val}_{p(x)}(\mathrm{den}\, F(x))$ for $F(x) \in k(x)$.

Let $A(x)$ be as in (1), then we define

$$\mathrm{den}\, A(x) = \operatorname*{lcm}_{i=1}^{n} \operatorname*{lcm}_{j=1}^{n} \mathrm{den}(a_{ij}(x)), \;\; \mathrm{den}\, A^{-1}(x) = \operatorname*{lcm}_{i=1}^{n} \operatorname*{lcm}_{j=1}^{n} \mathrm{den}(\tilde{a}_{ij}(x)).$$

If

$$F(x) = (F_1(x), F_2(x), \ldots, F_n(x))^T \in k(x)^n$$

then $\mathrm{den}\, F(x) = \mathrm{lcm}_{i=1}^{n} \mathrm{den}\, F_i(x)$, and $\mathrm{val}_{p(x)} F(x) = \min_{i=1}^{n} \mathrm{val}_{p(x)} F_i(x)$. A solution $F(x) = (F_1(x), F_2(x), \ldots, F_n(x))^T \in k(x)^n$ of (1) as well as a solution $F(x) \in k(x)$ of (2), (3) is a *rational* solution. If $\mathrm{den}\, F(x) \neq 1$ then this solution is *non-polynomial*, and *polynomial* otherwise.

If $p(x) \in \mathrm{Irr}(k[x])$, $f(x) \in k[x] \setminus \{0\}$ then we define the finite set

$$\mathcal{N}_{p(x)}(f(x)) = \{m \in \mathbb{Z} \,:\, p(x+m) | f(x)\}. \tag{7}$$

If $\mathcal{N}_{p(x)}(f(x)) = \emptyset$ then define $\max \mathcal{N}_{p(x)}(f(x)) = -\infty$, $\min \mathcal{N}_{p(x)}(f(x)) = +\infty$.

¿From now on we use the notation

$$V(x) = b_n(x-n), \;\; W(x) = b_0(x)$$

for equation (3), and

$$V(x) = u_1(x-1), \;\; W(x) = u_0(x),$$

where $u_1(x) = \mathrm{den}\, A(x)$, $u_0(x) = \mathrm{den}\, A^{-1}(x)$, for system (1).

The first computer algebra algorithm for finding solutions of (3) which belong to $k(x)$ was proposed in [3]. One of the statements proven in [3] (and later in [6] for the case of a system) can be formulated using notation (7) as follows:

Proposition 1. ([3,6]) *Let* $p(x)$ *divide the denominator of a rational solution of* (3) *or* (1), $p(x) \in \mathrm{Irr}(k[x])$. *Then* $\max \mathcal{N}_{p(x)}(V(x)) \geq 0$, *and* $\min \mathcal{N}_{p(x)}(W(x)) \leq 0$.

For $f(x), g(x) \in k[x] \setminus \{0\}$ we define their *dispersion set*:

$$\mathrm{ds}(f(x), g(x)) = \{h \in \mathbb{N} \,:\, \deg \gcd(f(x), g(x+h)) > 0\} \tag{8}$$

and their *dispersion*:

$$\mathrm{dis}(f(x), g(x)) = \max(\mathrm{ds}(f(x), g(x)) \cup \{-\infty\}). \tag{9}$$

The dispersion is equal to $-\infty$ iff $\deg \gcd(f(x), g(x+h)) = 0$ for all $h \in \mathbb{N}$, and belongs to $\mathbb{N}$ otherwise. The set $\mathrm{ds}(f(x), g(x))$ can be computed as the set of all integer non-negative roots of the polynomial $\mathrm{Res}_x(f(x), g(x+h)) \in k[h]$. This set can be also obtained from the full factorization of $f(x)$ and $g(x)$. Indeed, for given

$w(x), v(x) \in \mathrm{Irr}(k[x])$, $\deg w(x) = \deg v(x) = s$, one can easily recognize whether or not exists $h \in \mathbb{Z}$ such that $w(x+h) = v(x)$: if $w(x) = x^s + w_{s-1}x^{s-1} + \ldots$, $v(x) = x^s + v_{s-1}x^{s-1} + \ldots$, then $w(x+h) = x^s + (w_{m-1} + sh)x^{s-1} + \ldots$ and the only candidate for h is $\frac{v_{s-1} - w_{s-1}}{s}$, if this value belongs to $\mathbb{Z}$ ([20]). The computation is faster if one resorts to the approach from [20] based on the full factorization instead of computing integer roots of a resultant. This is successfully used, e.g., in Maple: LREtools[dispersion].

By Proposition 1, if a non-polynomial rational solution exists then the set $\mathrm{ds}(V(x), W(x))$ is not empty.

3 Algorithms for Constructing Universal Denominators

3.1 The Algorithm $\mathbf{A}_D$ from [5,6]

The algorithm is as follows:

Find $\mathcal{H} = \mathrm{ds}(V(x), W(x))$. If $\mathcal{H} = \emptyset$ then terminate the algorithm with the result $U(x) = 1$ (we suppose below that $\mathcal{H} = \{h_1, h_2, \ldots, h_s\}$ and $h_1 > h_2 > \ldots > h_s$, $s \geq 1$). Set $U(x) = 1$ and successively for $m = 1, 2, \ldots, s$ execute the following group of assignments:

$P(x) = \gcd(V(x), W(x + h_m))$
$V(x) = V(x)/P(x)$
$W(x) = W(x)/P(x - h_m)$
$U(x) = U(x)\prod_{i=0}^{h_m} P(x - i)$.

The final value of $U(x)$ is a universal denominator for equations (2), (3) or, resp., system (1).

We will refer to this algorithm as $\mathbf{A}_D$. This algorithm is exploited in current versions of Maple:

LREtools[ratpolysols], LinearFunctionSystems[UniversalDenominator].

3.2 The Algorithm from [11]

In [11] a more general problem than the search for rational solutions of system (1) was solved. However, the algorithm from [11, Prop. 3] can be used to compute a universal denominator $u(x)$ related to (1). Using our notation (setting in addition $h = \mathrm{dis}(V(x), W(x))$) this algorithm may be represented as follows.

Consider the sequence of polynomials $\{(V_j(x), W_j(x), P_j(x))\}$ defined inductively as:

$$V_0(x) = V(x), \;\; W_0(x) = W(x), \;\; P_0(x) = \gcd(V(x), W(x + h)),$$

and for $j = 1, 2, \ldots, h$,
$V_j(x) = V_{j-1}(x)/P_{j-1}(x)$,
$W_j(x) = W_{j-1}(x)/P_{j-1}(x - h + j - 1)$,
$P_j(x) = \gcd(V_j(x), W_j(x + h - j))$.

Then

$$u(x) = \prod_{j=0}^{h}\prod_{i=0}^{h-j} P_j(x - i).$$

3.3 The Algorithm $\mathbf{A}_U$ from [14]

An explicit formula for a lower bound of $\mathrm{val}_{p(x)}F(x)$ can be found in [14]: if $F(x)$ is a rational solution of equation (3) or system (1) then

$$\mathrm{val}_{p(x)}F(x) \geq -\min\left\{\sum_{l\in\mathbb{N}} \mathrm{val}_{p(x+l)}V(x),\ \sum_{l\in\mathbb{N}} \mathrm{val}_{p(x-l)}W(x)\right\} \tag{10}$$

for any $p(x) \in \mathrm{Irr}(k[x])$.

This formula was used in [14] as a base for the new algorithm $\mathbf{A}_U$ for computing a universal denominator. This algorithm can be divided into two steps. In the first step, $\mathbf{A}_U$ constructs a finite set M of irreducible polynomials that are candidates for divisors of denominators of rational solutions. At the second step, for each $p(x) \in M$ this algorithm computes the value

$$\gamma_{p(x)} = \min\left\{\sum_{l\in\mathbb{N}} \mathrm{val}_{p(x+l)}V(x),\ \sum_{l\in\mathbb{N}} \mathrm{val}_{p(x-l)}W(x)\right\}. \tag{11}$$

The product $\prod_{p(x)\in M} p^{\gamma_{p(x)}}(x)$ gives a universal denominator related to a given equation or system.

By Proposition 1 we can define

$$M = \left\{p(x) \in \mathrm{Irr}(k[x])\ :\ \min \mathcal{N}_{p(x)}(W(x)) \leq 0,\ \ \max \mathcal{N}_{p(x)}(V(x)) \geq 0\right\}.$$

For constructing this set the full factorization of polynomials $V(x), W(x)$ has to be found. Then we find the finite set $Q \subset \mathrm{Irr}(k[x])$ such that $q(x) \in Q$ iff

$$\min \mathcal{N}_{q(x)}(W(x)) = 0,\ \ \max \mathcal{N}_{q(x)}(V(x)) \geq 0.$$

Let $Q \neq \emptyset$ and $Q = \{q_1(x), q_2(x), \ldots, q_s(x)\}$, $s \geq 1$. For each $1 \leq i \leq s$ consider

$$M_{q_i(x)} = \{q_i(x), q_i(x+1), \ldots, q_i(x+h_i)\}, \tag{12}$$

where

$$h_i = \max \mathcal{N}_{q_i(x)}(V(x)). \tag{13}$$

We have $M = \bigcup_{i=1}^{s} M_{q_i(x)}$.

3.4 An Improved Version of the Algorithm $\mathbf{A}_U$ (the Algorithm $\mathbf{A}'_U$)

As it is described above the algorithm $\mathbf{A}_U$ contains two steps: the construction of the set M and the computation of $\gamma_{p(x)}$ using (11) for all $p(x) \in M$, which results in the universal denominator. Formula (11) contains the sums by $l \in \mathbb{N}$. In spite of the fact that $\mathbb{N}$ is infinite, the sums have only finite number of summands corresponding to the irreducible factors of $V(x)$ and $W(x)$, which are equal to non-negative and non-positive shifts of $p(x)$, respectfully (the corresponding valuations are equal to the exponents of such factors in the factorization of $V(x)$

and $W(x)$). No special way for time saving computing of the exponents $\gamma_{p(x)}$ was described in [14]. We propose below a possible way of this kind.

It is clear that when we compute (11) for $p(x) = q_i(x+j) \in M_{q_i(x)}$ (where $M_{q_i(x)}$ is as in (12)), the corresponding $\gamma_{q_i(x+j)}$ might be equal for many successive j. Indeed if we have computed $\gamma_{q_i(x)}$, and after that we compute $\gamma_{q_i(x+j)}$ for j from 1 to h_i, then the value can be changed only for those j for which there is an irreducible factor of $V(x)$ and/or $W(x)$ equal to $q_i(x+j)$ (such *critical* points can be computed in advance while constructing the set M). The consideration is a basis for the improved version of the algorithm $\mathbf{A}_U$; the new algorithm is presented below in details.

The first step is adjusted to compute the following:

– $\{q_i(x)\}_{i=1}^{s}$ and $\{h_i\}_{i=1}^{s}$ which correspond (12) and (13).
– The sets
$$C^{i,W} = \left\{c \in \mathbb{Z} : \mathrm{val}_{q_i(x+c)}(W(x)) > 0\right\},$$
$$C^{i,V} = \left\{c \in \mathbb{Z} : \mathrm{val}_{q_i(x+c-1)}(V(x)) > 0\right\}$$
of the critical points.
– $D^{i,W} = \left\{D_c^{i,W}\right\}$ and $D^{i,V} = \left\{D_c^{i,V}\right\}$ which are the sets of the valuations corresponding to the critical points: $D_c^{i,W} = \mathrm{val}_{q_i(x+c)}(W(x))$ for each $c \in C^{i,W}$ and $D_c^{i,V} = \mathrm{val}_{q_i(x+c-1)}(V(x))$ for each $c \in C^{i,V}$.

Note that all the data are computed simultaneously using the factorizations of $W(x)$ and $V(x)$.

The second step is performed as a loop by i from 1 to s. For each $q(x) = q_i(x)$ and $h = h_i$ execute the following:

– Construct the joint and sorted set of critical points:
$\left\{c_j : c_j \geq 0, c_j \leq h, c_j \in C^{i,W} \bigcup C^{i,V}\right\}_{j=1}^{n_i}$, with $c_1 < c_2 < \ldots < c_{n_i}$.
– Compute the intervals $\{l_0, \ldots, l_1 - 1\}$, $\{l_1, \ldots, l_2 - 1\}$, … $\{l_{k-1}, \ldots, h\}$ of the same exponents γ_1, γ_2, …, γ_k and the exponents themselves:
We initialize the computation with $k = 0$, $\gamma_0 = -1$ and $\gamma_w = \sum_{0 > c \in C^{i,W}} D_c^{i,W}$, $\gamma_v = \sum_{0 \geq c \in C^{i,V}} D_c^{i,V}$. Then for the critical points $c = c_1, c_2, \ldots c_{n_i}$ we compute the change of the values by $\gamma_w = \gamma_w + D_c^{i,W}$ (if $c \in C^{i,W}$) and/or $\gamma_v = \gamma_v - D_c^{i,V}$ (if $c \in C^{i,V}$), which gives a new $\gamma_{k+1} = \min(\gamma_v, \gamma_w)$. If $\gamma_{k+1} \neq \gamma_k$ then a new interval with the new exponent γ_{k+1} is started from $l_k = c$ (after that k is correspondingly increased by 1).
– Having added $l_k = h + 1$, compute the factor of the universal denominator that corresponds $q(x)$:
$U_i = \prod_{m=1}^{k} \prod_{j=l_{m-1}}^{l_m - 1} q(x+j)^{\gamma_m}$

The final universal denominator is the product of all U_i for $i = 1, 2, \ldots s$.

The algorithm is justified by considering the changes in (11) for computing $\gamma_{q(x+j)}$ with successive j. Note that γ_v (i.e. $\sum_{l \in \mathbb{N}} \mathrm{val}_{q(x+j+l)} V(x)$) and γ_w (i.e. $\sum_{l \in \mathbb{N}} \mathrm{val}_{q(x+j-l)} W(x)$) change a bit differently with the increase of j: the first one may only decrease and the second one may only increase. It leads to the corresponding differences in the algorithm in the definitions of $C^{i,W}$, $C^{i,V}$ and the formulas for the initial values of γ_w, γ_v and their changes.

We will refer to this detailed (improved) version of $\mathbf{A}_U$ as $\mathbf{A}'_U$.

4 Analysis of the Algorithms

4.1 Equivalence of Results

Proposition 2. *The universal denominators computed by the algorithms described in Section 3.2 coincide for any given $V(x), W(x)$. Intermediate polynomials computed by $\mathbf{A}_D$ are also computed as intermediate polynomials by the algorithm from [11].*

Proof. First show that the algorithm from [11] gives the same result and computes all the intermediate polynomials that $\mathbf{A}_D$ computes. Indeed, replace $\mathcal{H}$ by

$$\bar{\mathcal{H}} = \{h, h-1, \ldots, 0\}$$

$h = h_1 = \mathrm{dis}(V(x), W(x))$. This extension of $\mathcal{H}$ does not change the result (the additionally computed gcd's will be equal to 1). We also enumerate the values $V(x), W(x), P(x), U(x)$ in $\mathbf{A}_D$:

Set $U_0(x) = 1, V_0(x) = V(x), W_0(x) = W(x)$ and successively for $j = 0, 1, \ldots,$ $h-1$ execute the following group of assignments:

$$\begin{aligned}
&P_{j+1}(x) = \gcd(V_j(x), W(x+h-j))\\
&V_{j+1}(x) = V_j(x)/P_{j+1}(x)\\
&W_{j+1}(x) = W_j(x)/P_{j+1}(x-h+j)\\
&U_{j+1}(x) = U_j(x)\textstyle\prod_{i=0}^{h-j} P_{j+1}(x-i).
\end{aligned}$$

Evidently triples $(V_t(x), W_t(x), P_t(x))$ coincide for $t = 0, 1, \ldots h$ in both algorithms, and $u(x) = U_h(x)$.

It was proven in [11] that if $h = \mathrm{dis}(V(x), W(x))$ then

$$u(x) = \gcd\left(\prod_{i=0}^{h} V(x-i),\ \prod_{i=0}^{h} W(x+i)\right).$$

Therefore, the value $\mathrm{val}_{p(x)}u(x)$ is equal to the right-hand side of (11) for any $p(x) \in \mathrm{Irr}(k[x])$. This implies that the outputs of the algorithm from 3.2 and $\mathbf{A}_U$ coincide. Thus, the outputs of $\mathbf{A}_D$ and, resp. $\mathbf{A}_U$ coincide as well. The coincidence of the outputs of $\mathbf{A}_U$ and $\mathbf{A}'_U$ is evident. □

4.2 Complexity Comparison

We now give a complexity analysis of $\mathbf{A}_D$ and $\mathbf{A}'_U$. Let $n = \max\{\deg V(x),$ $\deg W(x)\}$ and $h = \mathrm{dis}(V(x), W(x))$. We compare the complexities $T_D(n,h)$ and $T_U(n,h)$ of $\mathbf{A}_D$ and $\mathbf{A}'_U$. In this context, the complexity is the number of the field operations in k in the worst case.

Both algorithms perform polynomial multiplications for getting $U(x)$. We do not specify the used polynomial multiplication algorithm, but suppose that the worst case is when it is necessary to multiply a big number (which is equal to $\deg U(x)$) of first degree polynomials.

Both algorithms spend the same time to find the full factorization of $V(x)$ and $W(x)$ and to compute their dispersion set. In addition, $\mathbf{A}'_U$ constructs the set Q as well as the set of corresponding h_i, the set of critical points, and the set of corresponding valuations. The cost of this computation in the worst case is $O(n)$ plus the cost of sorting critical points. This gives totally $O(n \log n)$.

On the other hand, $\mathbf{A}_D$ computes gcd's; if $h \geq n$ then in the worst case, the cost of this computation is $\sum_{i=0}^{n} T_{\mathrm{gcd}}(n-i)$, where $T_{\mathrm{gcd}}(n)$ is the complexity of the gcd computation for two polynomials whose maximal degree is n. If $0 < h < n$ then the cost in the worst case is $\sum_{i=0}^{h} T_{\mathrm{gcd}}(n-i)$. Obviously $\sum_{i=0}^{n} T_{\mathrm{gcd}}(n-i) = \sum_{i=0}^{n} T_{\mathrm{gcd}}(i)$, $\sum_{i=0}^{h} T_{\mathrm{gcd}}(n-i) = \sum_{i=n-h}^{n} T_{\mathrm{gcd}}(i)$, and we have the following proposition.

Proposition 3. *If $T_{\mathrm{gcd}}(n)/(n \log n) \to \infty$ then the difference $T_D(n,h)$–$T_U(n,h)$ is positive for almost all $n, h \in \mathbb{N}^+$ and*

$$T_D(n,h) - T_U(n,h) = \begin{cases} \sum_{i=0}^{n} T_{\mathrm{gcd}}(i) + O(n \log n), & \text{if } h \geq n, \\ \sum_{i=n-h}^{n} T_{\mathrm{gcd}}(i) + O(n \log n), & \text{if } h < n. \end{cases} \tag{14}$$

In the next proposition we use the Ω-notation which is very common in complexity theory ([19]). Unlike O-notation which is used for describing upper asymptotical bounds, the Ω-notation is used for describing lower asymptotical bounds.

Proposition 4. *Let $T_{\mathrm{gcd}}(n) = \Omega(n^d)$, $d > 1$. Then the difference $T_D(n,h) - T_U(n,h)$ is positive almost all $n, h \in \mathbb{N}^+$ and is $\Omega(R(n,h))$, where*

$$R(n,h) = \begin{cases} n^{d+1}, & \text{if } h \geq n, \\ hn^d, & \text{if } h < n. \end{cases}$$

Proof. The case $h \geq n$ follows from (14) and $d > 1$. In the case $h < n$ we can use the inequality

$$\sum_{i=m}^{n} i^d > \frac{n^d(n-m)}{d+1} \tag{15}$$

which is valid for any integer $0 < m \leq n$ and real $d \geq 1$. Taking $m = n-h$ we get the claimed. To prove (15) note that the function x^d is monotonically increasing when $x \geq 0$ and $d \geq 1$. This gives for $m < n$ (the case $m = n$ is trivial):

$$\sum_{i=m}^{n} i^d > \sum_{i=m+1}^{n} i^d > \sum_{i=m+1}^{n} \int_{i-1}^{i} x^d \, \mathrm{d}x = \int_{m}^{n} x^d \, \mathrm{d}x = \frac{n^{d+1}}{d+1} \left(1 - \left(\frac{m}{n}\right)^{d+1}\right).$$

Since in our case $1 - \left(\frac{m}{n}\right)^{d+1} \geq 1 - \frac{m}{n}$, we get (15). $\square$

To the authors' knowledge $T_{\mathrm{gcd}}(n) = \Omega(n^d)$, $d > 1$, for the algorithms now in use in actual practice for gcd computations.

The fast Euclidean algorithm [12, Ch. 11] has complexity $O(n \log^2 n \log \log n)$ if Fast Fourier Transform is used to multiply polynomials. But this version of the fast Euclidean algorithm is not practical due to a big constant hidden in O. Nevertheless, if we suppose that the fast Euclidean algorithm is used and the estimate $\Omega(n\ \log^2 n\ \log\log n)$ (or, even $\Omega(n\ \log^2 n)$) is valid for the complexity of this algorithm then by Proposition 3 the difference $T_D(n,h) - T_U(n,h)$ is positive (i.e., $T_U(n,h) < T_D(n,h)$) for almost all $n,h \in \mathbb{N}^+$.

5 Implementation

Below we consider an implementation in Maple of $\mathbf{A}'_U$ (Section 5) and demonstrate the corresponding time comparison with $\mathbf{A}_D$ (Section 6). As it was shown in Proposition 2 both algorithms give the same result, and the comparison is correct. The algorithm from [11] is similar to $\mathbf{A}_D$ by Proposition 2, and we do not involve this algorithm into the comparison.

As we mentioned in Section 3.1 the algorithm $\mathbf{A}_D$ implementation is available in Maple as an internal procedure of the package LREtools. We implemented our new algorithm $\mathbf{A}'_U$ and performed experimental comparison of the two algorithms.

The implementation has several peculiarities which are discussed below.

5.1 Full Factorization

The algorithm $\mathbf{A}'_U$ (and $\mathbf{A}_U$ as well) is based on the full factorization of the given polynomials $V(x)$ and $W(x)$. Our implementation uses the result of the factorization not only to construct the set M of irreducible polynomials, but also computes (11) using it. Note that it is not the case for the implementation of the algorithm $\mathbf{A}_D$ in Maple. It uses the procedure LREtools[dispersion] to compute the dispersion of polynomials which implements the algorithm [20], i.e., uses the full factorization. But the next steps of the algorithm $\mathbf{A}_D$ are implemented as presented not exploiting the result of the factorization of the previous step.

5.2 Shift Computation

Our implementation uses vastly the auxiliary procedure, which given $p(x), r(x) \in \mathrm{Irr}(k[x])$ computes the shift $s \in \mathbb{Z}$ such that $p(x) = r(x+s)$ or defines that no such s exists (actually it is a particular case of computing $\mathcal{N}_{p(x)}(r(x))$ when $r(x) \in \mathrm{Irr}(k[x])$). The procedure is used both to compute the set M and to compute $C^{i,W}$, $C^{i,V}$, $D^{i,W}$, $D^{i,V}$ for further computations of the exponents γ. The shift computation is implemented efficiently using the main idea of the algorithm [20] presented in the end of Section 2.

5.3 Computing Universal Denominator

Though we compute the values γ successively, it is better to compute the universal denominator at once for all $q_i(x+j)$, rather than compute the universal denominator also successively. In the latter case, the intermediate computations of the preliminary results might be costly at least in Maple.

6 Three Experiments

Using our implementation of the algorithm $\mathbf{A}'_U$ and the implementation of the algorithms $\mathbf{A}_D$ that is embedded in Maple, we have performed three experiments to compare the algorithms.

6.1 Experiment 1

We have applied both algorithms to the following similar inputs:

(a) $V(x) = W(x) = \prod_{i=1}^{l}(x + m + 1/i)(x - m + 1/i)$ for $m = 20, 100, 500, 2500$, $l = 1, 15, 30, 45, 60$;
(b) $V(x)=W(x)=\prod_{i=1}^{l}(x+m+i+1/i)(x-m-i+1/i)$ for $m=20, 100, 500, 2500$, $l = 1, 15, 30, 45, 60$.

The corresponding universal denominators found by both algorithm for the inputs are, respectfully, the following:

(a) $\prod_{i=1}^{l}\prod_{j=-m}^{m}(x - j - 1 + 1/i)$;
(b) $\prod_{i=1}^{l}\prod_{j=-m-i}^{m+i}(x - j + 1/i)$.

The experiment is based on the example from [15], which is transformed to be more complicated by using l similar pair factors instead of the only one pair. Tables 1 and 2 show the CPU time[1] needed to compute the corresponding universal denominators by three implementations for each of the pair $V(x)$ and $W(x)$. The input polynomials are expanded before calling the implementations. The expansion is needed to create equal conditions for both algorithms (otherwise the factored input definitely simplifies the work for $\mathbf{A}'_U$). In addition, it has been found that the implementation of the algorithm [20] in Maple for computing the dispersion of two polynomials uses some additional preprocessing which leads to inefficiency for the inputs in the factored form at least in our experiments, so the expanded input allows eliminating this question in our comparison.

Table 1. Results of the experiment 1(a), in seconds

	m=20		m=100		m=500		m=2500	
	$\mathbf{A}'_U$	$\mathbf{A}_D$	$\mathbf{A}'_U$	$\mathbf{A}_D$	$\mathbf{A}'_U$	$\mathbf{A}_D$	$\mathbf{A}'_U$	$\mathbf{A}_D$
l=1	0.000	0.016	0.000	0.015	0.015	0.015	0.016	0.031
l=15	0.079	0.141	0.094	0.141	0.172	0.203	0.546	0.438
l=30	0.375	0.547	0.407	0.562	0.547	0.656	1.266	1.109
l=45	0.719	1.140	0.828	1.235	1.172	1.531	3.015	2.531
l=60	2.032	2.875	2.390	3.344	3.000	4.516	5.063	5.704

The results show that the algorithms behave differently with the growth of m and l. The results of $\mathbf{A}_D$ are getting relatively worse with the growth of l if we fix any m, and the results of $\mathbf{A}'_U$ are getting relatively worse with the growth of m if

[1] For all the experiments: Maple 13, Windows XP, Pentium 4 1.7 GHz, 512 MB RAM.

Table 2. Results of the experiment 1(b), in seconds

	m=20		m=100		m=500		m=2500	
	$\mathbf{A}'_U$	$\mathbf{A}_D$	$\mathbf{A}'_U$	$\mathbf{A}_D$	$\mathbf{A}'_U$	$\mathbf{A}_D$	$\mathbf{A}'_U$	$\mathbf{A}_D$
l=1	0.016	0.015	0.000	0.000	0.000	0.016	0.031	0.031
l=15	0.078	0.375	0.109	0.422	0.172	0.531	0.578	1.032
l=30	0.359	2.890	0.407	3.063	0.531	3.484	1.266	5.344
l=45	0.860	10.641	0.796	11.547	1.516	13.234	3.078	17.656
l=60	2.406	31.187	2.719	33.484	2.657	37.125	4.766	44.797

we fix any l. The latter observation may be explained if we analyze the structure of the algorithms in respect to the particular problem in hand: actually, for the fixed m and given l $\mathbf{A}'_U$ performs similar set of operations l times, but $\mathbf{A}_D$ needs to perform gcd computations with the polynomials of l times higher degrees.

It is easy to see that the inputs (a) are more convenient for the algorithm $\mathbf{A}_D$: the gcd is computed only once for each input, the number of multiplied polynomials is $2m + 1$, while for $\mathbf{A}'_U$ this number is $(2m + 1)l$. In spite of this handicap the timing of $\mathbf{A}'_U$ looks better for the whole experiment (Table 1).

The input (b) corresponds near to the worst case for both algorithms $\mathbf{A}'_U$ and $\mathbf{A}_D$ (the input size is a pair of numbers as in Section 4.2), and an advantage of $\mathbf{A}'_U$ is evident (Table 2).

We have noted in Section 3.4 that no special way for time saving computing of the exponents $\gamma_{p(x)}$ was proposed in the description of $\mathbf{A}_U$ given in [14]. If one uses formula (11) for each $p(x) \in M$ then the total computation time increases dramatically. We have implemented a straightforward version of $\mathbf{A}_U$ as well for the preliminary experiments and, for example, the result of $\mathbf{A}_U$ for the input of type (a) with $m = 2500$, $l = 60$ is 350 seconds.

6.2 Experiment 2

We have also applied the algorithms to several sets of randomly generated pairs of polynomials $V(x)$ and $W(x)$. Each set contains 500 pairs, and each polynomial is generated using Maple command randpoly(x,degree=d,terms=l), i.e., it is a polynomial of degree up to d and it contains up to l terms. Note that given such generated polynomials, the universal denominator found by the considered algorithms is most probably x^n for some $n \in \mathbb{N}$, and moreover it is just 1 for most of the cases. Still the experiment is meaningful, since if we try to search for the rational solution of absolutely arbitrary equations, it would be exactly like this. 9 sets are generated for $d = 10, 20, 30$ and $l = 2, d/2, d$. Table 3 shows the CPU time needed to compute the corresponding universal denominators by the implementations for each of the set. We do not need to expand the input polynomials before calling the implementations in the experiment since the polynomials are expanded by construction.

The results show that $\mathbf{A}'_U$ is better than $\mathbf{A}_D$ in this experiment for all the sets.

Table 3. Results of the experiment 2, in seconds

	l=2		l=d/2		l=d	
	$\mathbf{A}'_U$	$\mathbf{A}_D$	$\mathbf{A}'_U$	$\mathbf{A}_D$	$\mathbf{A}'_U$	$\mathbf{A}_D$
d=10	1.578	6.016	5.953	9.578	6.734	10.157
d=20	1.750	8.094	8.594	12.938	9.828	13.969
d=30	1.922	10.422	12.235	17.234	13.985	19.375

6.3 Experiment 3

We have also applied the algorithms to several sets of other randomly generated pairs of polynomials $V(x)$ and $W(x)$. Each set contains again 500 pairs, but the polynomials are generated differently. Each polynomial is generated as a product of at most l factors of the form $(x - r_i)^{d_i}$, where r_i is a random integer between -10 and 10, d_i is a random integer between 0 and d. Such method of generation ensures that the found universal denominators will be non trivial. 9 sets are generated for $d = 2, 4, 6$ and $l = 1, 5, 10$. Table 4 shows the CPU time needed to compute the corresponding universal denominators by the implementations for each of the set. The input polynomials are expanded before calling the implementations.

Table 4. Results of the experiment 3, in seconds

	l=1		l=5		l=10	
	$\mathbf{A}'_U$	$\mathbf{A}_D$	$\mathbf{A}'_U$	$\mathbf{A}_D$	$\mathbf{A}'_U$	$\mathbf{A}_D$
d=1	0.219	0.890	1.094	2.453	2.672	6.265
d=3	0.390	1.390	2.953	6.500	5.844	14.937
d=5	0.437	1.609	4.328	9.750	8.313	23.250

The results show that $\mathbf{A}'_U$ is better than $\mathbf{A}_D$ in this experiment for all the sets.

7 Conclusion

Our investigation presented in the paper has confirmed that it might be useful to revisit the problems which were solved earlier by the algorithms which use incomplete factorization based on computation of the greatest common divisors as a result of the desire to avoid the use of the full factorization. The full factorization based algorithm $\mathbf{A}_U$ (and its new improved version $\mathbf{A}'_U$) for the universal denominator construction is proved to deliver the same results as the old gcd computation based algorithm $\mathbf{A}_D$, but the implementation of $\mathbf{A}'_U$ is shown to be more efficient. It is especially logical to switch to the new approach, in particular, in Maple, since the existing Maple implementation of the algorithm $\mathbf{A}_D$ uses already the factorization based auxiliary algorithm for computing the dispersion, i.e., the required factorizations are computed already.

Note that new algorithms should not be necessary obtained out of the old ones just by substituting gcd computations with the corresponding computations using the results of factorizations. It might be more useful to re-think the whole algorithm over again based on the new approach. In this way, $\mathbf{A}'_U$ utilizes the new computations based on the new formula (11), and they are implemented efficiently, e.g., taking into account the fact that when we compute (11) for $p(x) = q_i(x+j) \in M_{q_i(x)}$ from (12), the corresponding $\gamma_{q_i(x+j)}$ might be equal for many successive j.

Logically if the basic operations are significantly changed then concepts for algorithms designing have to be updated.

Acknowledgments. The authors are grateful to M. Barkatou and M. Petkovšek for interesting discussions, and to anonymous referees for their helpful comments.

References

1. Abramov, S.: On the summation of rational functions. USSR Comput. Math. Phys. 11, 324–330 (1971); Transl. from Zh. vychisl. mat. mat. fyz. 11, 1071–1075 (1971)
2. Abramov, S.: Problems of computer algebra involved in the search for polynomial solutions of linear differential and difference equations. Moscow Univ. Comput. Math. Cybernet. 3, 63–68 (1989); Transl. from Vestn. MGU. Ser. 15. Vychisl. mat. i kibernet. 3, 53–60 (1989)
3. Abramov, S.: Rational solutions of linear difference and differential equations with polynomial coefficients. USSR Comput. Math. Phys. 29, 7–12 (1989); Transl. from Zh. vychisl. mat. mat. fyz. 29, 1611–1620 (1989)
4. Abramov, S.: Rational solutions of linear difference and q-difference equations with polynomial coefficients. In: ISSAC 1998 Proceedings, pp. 303–308 (1995)
5. Abramov, S.: Rational solutions of linear difference and q-difference equations with polynomial coefficients. Programming and Comput. Software 21, 273–278 (1995); Transl. from Programmirovanie 6, 3–11 (1995)
6. Abramov, S., Barkatou, M.: Rational solutions of first order linear difference systems. In: ISSAC 1998 Proceedings, pp. 124–131 (1998)
7. Abramov, S., Bronstein, M., Petkovšek, M.: On polynomial solutions of linear operator equations. In: ISSAC 1995 Proceedings, pp. 290–295 (1995)
8. Abramov, S., van Hoeij, M.: A method for the integration of solutions of Ore equations. In: ISSAC 1997 Proceedings, pp. 172–175 (1997)
9. Abramov, S., van Hoeij, M.: Integration of solutions of linear functional equations. Integral Transforms and Special Functions 8, 3–12 (1999)
10. Abramov, S., Ryabenko, A.: Indicial rational functions of linear ordinary differential equations with polynomial coefficients. Fundamental and Applied Mathematics 14(4), 15–34 (2008); Transl. from Fundamentalnaya i Prikladnaya Matematika 14(4), 15–34 (2008)
11. Barkatou, M.: Rational solutions of matrix difference equations: problem of equivalence and factorization. In: ISSAC 1999 Proceedings, pp. 277–282 (1999)
12. Von zur Gathen, J., Gerhard, J.: Modern Computer Algebra, 2nd edn. Cambridge University Press, Cambridge (2003)
13. Gerhard, J.: Modular Algorithms in Symbolic Summation and Symbolic Integration. LNCS, vol. 3218. Springer, Heidelberg (2004)

14. Gheffar, A., Abramov, S.: Valuations of rational solutions of linear difference equations at irreducible polynomials. Adv. in Appl. Maths (submitted 2010)
15. van Hoeij, M.: Rational solutions of linear difference equations. In: ISSAC 1998 Proceedings, pp. 120–123 (1998)
16. van Hoeij, M.: Factoring polynomials and the knapsack problem. J. Number Theory 95, 167–189 (2002)
17. van Hoeij, M., Levy, G.: Liouvillian solutions of irreducible second order linear difference equations. In: ISSAC 2010 Proc. (2010)
18. Khmelnov, D.E.: Search for polynomial solutions of linear functional systems by means of induced recurrences. Programming and Comput. Software 30, 61–67 (2004); Transl. from Programmirovanie 2, 8–16 (2004)
19. Knuth, D.E.: Big omicron and big omega and big theta. ACM SIGACT News 8(2), 18–23 (1976)
20. Man, Y.K., Wright, F.J.: Fast polynomial dispersion computation and its application to indefinite summation. In: ISSAC 1994 Proceedings, pp. 175–180 (1994)
21. Petkovšek, M.: Hypergeometric solutions of linear recurrences with polynomial coefficients. J. Symbolic Computation 14, 243–264 (1992)
22. Maple online help, http://www.maplesoft.com/support/help/

A Role of Symbolic Computations in Beam Physics

Serge N. Andrianov

Faculty of Applied Mathematics and Control Processes,
Saint Petersburg State University,
198504, Saint Petersburg, Russian Federation
sandrianov@yandex.ru

Abstract. It is known that accelerator physics technology has made essential contributions to other branches of science and technology. Experiments realized on linear and circular accelerators have made remarkable discoveries about the basic nature of matter. In particular, there are now two accelerator projects. The first of them is already realized — the Large Hadron Collider, the second — the pilot project for future dedicated EDM machine. These and other similar projects (i. e., the project NICA, JINR, Dubna) demand some special requirements for simulation methods and technologies. Indeed, the successful functioning of these accelerators requires essential advancement in theory and technology leading to new particle accelerators capabilities. The complexity of accelerator physics problems makes comprehensive use of modern analytical, numerical, and symbolic methods. Only if we integrate these approaches the corresponding computational technologies will be effective. In the present report, we discuss some problems of correlation between symbolic and numerical manipulation. The main approach for beam dynamics is based on Lie algebraic methods and corresponding matrix formalism as presentation tools. All suggested approaches are realized using symbolic algorithms, and the corresponding symbolic formulae are assumed as a basis of numerical algorithms. This approach allows to realize the necessary numerical modeling using parallel and distributed computational systems for some practical problems.

Keywords: Symbolic algebra, beam physics, code generation, Lie algebraic methods, parallel and distributed computing.

1 Introduction

One of the main goals of modern computational accelerator physics can be formulated as to give researchers a powerful and efficient tools for nonlinear beam dynamics modeling.

The use of the Lie algebraic tools [1], in combination with a symbolic algebra system (i. e. MAPLE, MATHEMATICA), enables us to carry out the entire modeling on a high level. These systems easily handle the elaborate necessary computation procedures required to form the equations of motion and then generate them into such languages as FORTRAN, C, Java, Visual Basic. Besides, the last version of these packages can export resulting expressions into the MATLAB package. It should be noted that MATLAB is a well suited package for running small computational simulations followed by various post processing activities including graphics. Knowledge of analytical representations for most preliminary and final expressions gives a researcher very powerful tools for both next numerical computation and qualitative investigation. Indeed, firstly symbolic algebra systems opened possibility to generate code automatically from formulas, and secondly — to investigate qualitative properties of model-based systems. The preliminary scheme of symbolic and numerical modules is presented in Fig. 1.

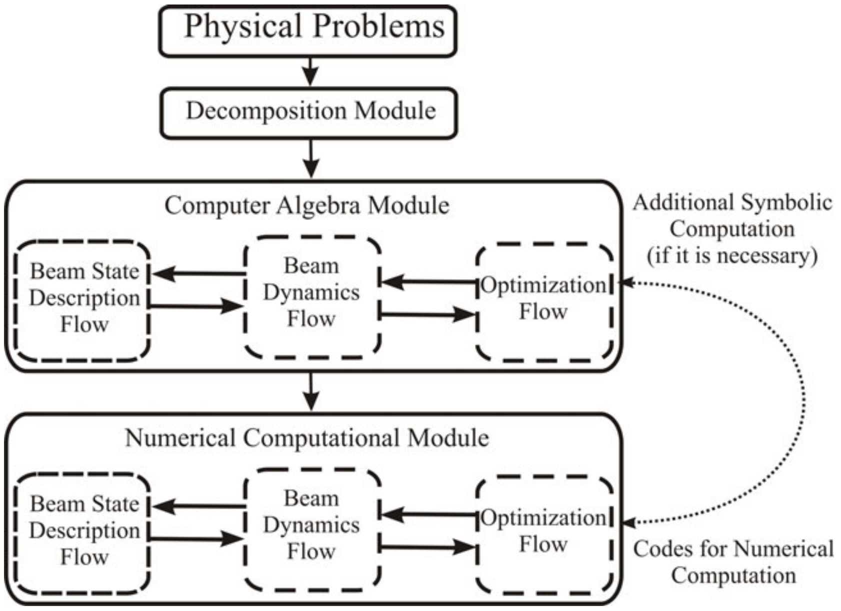

Physical Problems
Decomposition Module
Computer Algebra Module
Beam State Description Flow
Beam Dynamics Flow
Optimization Flow
Additional Symbolic Computation (if it is necessary)
Numerical Computational Module
Beam State Description Flow
Beam Dynamics Flow
Optimization Flow
Codes for Numerical Computation

Fig. 1. The scheme of usage of computer algebra tools

2 Mathematical Background

In this paper, dynamics of beam particles is presented in the form of an operator form using the Lie nonlinear transformation $\mathcal{M}$ (see, for example, [2])

$$\mathcal{M}(t|t_0) = \mathrm{T}\exp\left(\int\limits_{t_0}^{t} \mathcal{L}_{\mathbf{F}(\tau)} d\tau\right), \tag{1}$$

where $\mathcal{L}_{\mathbf{F}(\tau)}$ is a Lie operator associated with some vector function $\mathbf{F}(\tau) = \mathbf{F}(\tau; \mathbf{X}, \mathbf{U}, \mathbf{B})$ defining the motion equation for beam particles

$$\frac{d\mathbf{X}}{dt} = \mathbf{F}(t; \mathbf{X}, \mathbf{U}, \mathbf{B}). \tag{2}$$

Here $\mathbf{X} \in \mathcal{X}$, $\mathbf{U} \in \mathfrak{U}$, $\mathbf{B} \in \mathfrak{B}$ are the phase vector, the vector of control functions, and the vector of control parameters, respectively. In eq. (1), the beam propagator $\mathcal{M}(t|t_0)$ is presented in the form of the so-called time-ordered exponential operator (see, for example, [1,2]). Eqs. (1) and (2) define a dynamical system with control functions $\mathbf{U} = (U_1, \ldots, U_n)^{\mathrm{T}}$ (U_j are electromagnetic field components corresponding to an external (control) electromagnetic field), control parameters $\mathbf{B} = (B_1, \ldots, B_m)^{\mathrm{T}}$ (here B_k are parameters which can not be verified during a work-session), and the operator $\mathcal{M}$ can be identified with the dynamical system itself.

For practical modeling the time-ordered operator $\mathcal{M}(t|t_0)$ in (1) is presented using the so-called Magnus's presentation

$$\mathcal{M}(t|t_0) = \exp\left(\mathcal{L}_{\mathbf{G}(t|t_0)}\right). \tag{3}$$

The new vector-function $\mathbf{G}(t|t_0)$ in eq. (3) for the Lie operator $\mathcal{L}_{\mathbf{G}(t|t_0)}$ is calculated using the continuous analogue of the well known CBH-formula [3,6] symbolically.

2.1 The Symmetry and Explicit Solutions for Dynamical Systems

The above introduced Lie transformation $\mathcal{M}(t|t_0)$ maps the initial phase manifold (occupied by beam particles) $\mathfrak{M}(t_0)$ onto a current manifold $\mathfrak{M}(t)$. It is natural that every dynamical system has some qualitative properties, and these properties should be preserved during the whole propagation process. Here we should mention such properties as invariants (kinematic and dynamic, see, for example, [4] and [5,6]), symmetries. We should especially mention the possibility of finding explicit solutions for some special types of dynamical systems [7].

In general the retrieval of invariants and symmetries is a very complicated problem. Taking into account that usually the Lie map can be found using perturbation presentation we redefine the conception of invariants and symmetry. In other words, a researcher finds so-called approximating invariants and symmetries. But, in the case of kinematic invariants we should tell about explicit invariants (compare, for example, with symplectic property, which is inherent in any Hamiltonian system). For many problems of accelerators physics one has to accurately track the orbits of beam particles during many thousand turns. Traditional numerical methods for motion equation integration result in several artificial effects, which are not allied to real behavior of beam particles. This urges us to use special numerical methods which guarantee necessary conservation lows with necessary precision. First of all, here we mean the conservation problem for symplecticity property when one replaces an exactly symplectic map (produced by any Hamiltonian system) with approximating map.

At this step, the role of symbolic computation is very important. Indeed, the approaches suggested in [5,6] can be realized up to necessary order of approximation (for approximating invariants and symmetries) and the algorithm for explicit solutions in a closed form as described in [7].

2.2 Symbolic Computation Algorithm for the Matrix Formalism

The above approach is based on the so-called matrix formalism for Lie algebraic tools [6,8] up to necessary order of approximation. Indeed, under the assumption that the function in Eq. (2) $\mathbf{F}(\mathbf{X},t)$ admits the expansion as a Taylor series $\mathbf{F}(\mathbf{X},t) = \sum_{k=0}^{\infty} \mathbb{P}^{1k}(t)\mathbf{X}^{[k]}$, we can rewrite required solution of Eq. (2) in terms of matrices $\mathbb{M}^{1k}$:

$$\mathbf{X}(t) = \mathcal{M}(t|t_0) \circ \mathbf{X}_0 = \sum_{k=0}^{\infty} \mathbb{M}^{1k}(t|t_0)\mathbf{X}_0^{[k]}. \qquad (4)$$

Here $\mathbf{X}^{[k]}$ is the Kronecker power of the kth order for the phase vector $\mathbf{X}^{[k]}$ and $\mathbb{P}^{1k}(t)$, $\mathbb{M}^{1k}(t)$ are two-dimensional matrices. Matrices $\mathbb{M}^{1k}$ are named aberration matrices (matrices $\mathbb{M}^{1k}$ accumulate all abberations of the kth order). The main goal of the matrix formalism is to calculate these matrices up to some approximating order N. This order usually is defined taking into account information about used control elements (for example, usually for quadrupole magnet lenses it is necessary to know $\mathbb{M}^{1k}$ up to third or fifth order).

The Dragt–Finn factorization for the Lie transformations (1) or (3) allows us to rewrite the corresponding exponential operators as an infinite product of exponential operators generated by corresponding Lie operators

$$\mathcal{M} = \ldots \cdot \exp\{\mathcal{L}_{H_2}\} \cdot \exp\{\mathcal{L}_{H_1}\} = \exp\{\mathcal{L}_{V_1}\} \cdot \exp\{\mathcal{L}_{V_2}\} \cdot \ldots, \qquad (5)$$

where $H_k = \mathbf{H}_k^{\mathrm{T}}\mathbf{X}^{[k]}$, $V_k = \mathbf{V}_k^{\mathrm{T}}\mathbf{X}^{[k]}$ are homogeneous polynomials of the kth order. The vectors $\mathbf{H}_k$ or $\mathbf{V}_k$ can be calculated with the help of the continuous analogue of the CBH- and Zassenhauss formulae and by using the Kronecker product and Kronecker sum technique for matrices [8]. Moreover, using the matrix representation for the Lie operators one can write a matrix representation for the Lie map generated by these Lie operators. Here we consider homogeneous equations of particle motion in a reference orbit neighborhood. So, for any beam propagator one can write the following matrix representation in the well known Poincaré–Birkhoff–Witt basis

$$\mathcal{M} \cdot \mathbf{X} = \mathbb{M} \circ \mathbf{X}^{\infty} = (\mathbb{M}^{11}\, \mathbb{M}^{12}\, \ldots\, \mathbb{M}^{1k} \ldots)\mathbf{X}^{\infty} = \sum_{k=1}^{\infty} \mathbb{M}^{1k}\mathbf{X}^{[k]}, \qquad (6)$$

where $\mathbf{X}^{\infty} = (\mathbf{X}\,\mathbf{X}^{[2]} \ldots \mathbf{X}^{[k]} \ldots)^{\mathrm{T}}$, and the matrices $\mathbb{M}^{1k}$ (*solution matrices*) can be calculated according to the recurrence sequence of following formulas [8]:

$$\mathcal{M}_k \circ \mathbf{X}^{[l]} = \exp\{\mathcal{L}_{G_k}\} \circ \mathbf{X}^{[l]} = \mathbf{X}^{[l]} + \\ + \sum_{m=1}^{\infty} \frac{1}{m!} \prod_{j=1}^{m} \mathbb{G}_m^{\oplus((j-1)(k-1)+l)} \mathbf{X}^{[m(k-1)+l]}, \qquad (7)$$

where $\mathbb{G}^{\oplus l} = \mathbb{G}^{\oplus(l-1)} \otimes \mathbb{E} + \mathbb{E}^{[l-1]} \otimes \mathbb{G}$ denotes the Kronecker sum of the lth order. The algebraical expressions similar to (7) are used for evaluation of aberration matrices up to an approximating order N. For this purpose a researcher has to realize one of the concatenation procedures. For example, we can suggest the following expressions.

Let us introduce a notation $\mathcal{M}_{\leq k} = \mathcal{M}_k \circ \mathcal{M}_{k-1} \circ \ldots \circ \mathcal{M}_2 \circ \mathcal{M}_1$. Then we write for nonlinearities of the third order

$$\mathcal{M}_{\leq 3} \circ \mathbf{X} = \mathbb{M}^{11} \left(\mathbf{X} + \sum_{m=2}^{3} \sum_{k=1}^{\infty} \frac{\mathbb{P}_m^{k1}}{k!} \mathbf{X}^{[k(m-1)+1]} + \right. \\ \left. + \sum_{l=1}^{\infty} \sum_{k=1}^{\infty} \frac{1}{k!l!} \mathbb{P}_2^{kl} \, \mathbb{P}_3^{l\,(k+1)} \, \mathbf{X}^{[2l+k+1]} \right). \quad (8)$$

Repeating described procedure we can evaluate $\mathcal{M}_{\leq k} \circ \mathbf{X}$ for any k with necessary calculation accuracy. But, as $\mathcal{M}_{\leq k}$ is a truncated form of the map $\mathcal{M}$ (we use information about generating function up to the kth order only) then we have to evaluate up to terms of the kth order for obtaining desired result for $\mathcal{M}_{\leq k}$. So we can write

$$\mathcal{M}_{\leq 2} \circ \mathbf{X} \approx \mathbb{M}^{11}\mathbf{X} + \mathbb{M}^{12}\mathbf{X}^{[2]}, \quad \mathbb{M}^{12} = \mathbb{M}^{11}\, \mathbb{P}_2^{11},$$

$$\mathcal{M}_{\leq 3} \circ \mathbf{X} \approx \mathbb{M}^{11} \left(\mathbf{X} + \mathbb{P}_2^{11}\, \mathbf{X}^{[2]} + \left(\mathbb{P}_3^{11} + \frac{1}{2} \mathbb{P}_2^{21} \right) \mathbf{X}^{[3]} \right) =$$

$$= \mathbb{M}^{11}\mathbf{X} + \mathbb{M}^{12}\mathbf{X}^{[2]} + \mathbb{M}^{13}\mathbf{X}^{[3]},$$

$$\mathbb{M}^{13} = \mathbb{M}^{11} \left(\mathbb{P}_3^{11} + \frac{1}{2!} \mathbb{P}_2^{21} \right),$$

and so on. In the following we act step-by-step by operators $\mathcal{M}_k$, $k < m$ on the vector $\mathbf{X}$. Here we have to keep terms up to the kth order, and as a result we obtain a matrix representation of the beam propagator in the basis $\left(\mathbf{X}, \mathbf{X}^{[2]}, \ldots, \mathbf{X}^{[k]}\right)$. In other words, one can write expression (6) up to some kth order.

For some special problems one has evaluated above described procedures for a phase vector moment of the kth order $\mathbf{X}^{[m]}$. Using the Lie map and Kronecker product properties we can proceed necessary evaluations and write [6]:

$$\mathcal{M} \circ \mathbf{X}^{[m]} = (\mathcal{M} \circ \mathbf{X})^{[m]} = \left(\sum_{k=1}^{\infty} \mathbb{M}^{1k}\, \mathbf{X}^{[k]} \right)^{[m]} =$$

$$= \sum_{\substack{k_1=1 \\ k_1+\ldots+k_m=m}} \ldots \sum_{k_m=1} \mathbb{M}^{1k_1}\, \mathbf{X}^{[k_1]} \otimes \ldots \otimes \mathbb{M}^{1k_m}\, \mathbf{X}^{[k_m]} =$$

$$= \sum_{\substack{k_1=1 \\ k_1+\ldots+k_m=m}} \ldots \sum_{k_m=1} \left(\mathbb{M}^{1k_1} \otimes \ldots \otimes \mathbb{M}^{1k_m} \right) \mathbf{X}^{[m]} = \sum_{\substack{k_1=1 \\ k_1+\ldots+k_m=m}} \ldots \sum_{k_m=1} \mathbb{M}^{ml}\mathbf{X}^{[l]}, \quad (9)$$

where $\mathbb{M}^{ml} = \sum\limits_{\substack{k_1+\ldots+k_m=l \\ k_i \geq 1}} \bigotimes\limits_{i=1}^{m} \mathbb{M}^{1k_i}, \quad l \geq m$ (compare with Eq. (6)).

For the inverse map $\mathcal{M}^{-1}\colon \mathbf{X} \to \mathbf{X}_0 = \mathcal{M}^{-1} \circ \mathbf{X}$, one can compute the corresponding block matrices using the generalized Gauss algorithm. We must note here that one should evaluate the inverse matrix for $\mathbb{M}^{11}$ only. The remaining block matrices for the inverse matrix demand algebraic operations only.

As one can see, the starting point for all described evaluations are matrices $\mathbb{P}^{1k}$. Using these matrices (without their content) one can compute aberration matrices $\mathbb{M}^{1k}$ according to the following (or similar) formulae. Here we should mention that before symbolic or numerical evaluation, a researcher must define correlations between physical and their mathematical (in our case matrix) representation. It is necessary to note that symbolic expressions for above described matrices can be evaluated in symbolic form for some representative class of time dependent functions entering linear matrices $\mathbb{P}^{11}(t)$ (see, for example, [9]). These symbolic expressions for elements of $\mathbb{P}^{11}(t)$ can be embedded in a data base and used in the numerical computations on-demand.

2.3 The Numerical Computation Algorithm for the Matrix Formalism

Above described approach is used for symbolic computation of aberration matrices (and additional formulae which are necessary according to the physical problem). In the case of more complex time dependence of $\mathbb{P}^{11}(t)$, one can use the numerical solution methods using the following algorithm.

It is necessary to note that we can consider eq. (2) as a starting point for our beam evolution consideration in the following form

$$\frac{d\mathbf{X}}{dt} = \mathbf{F}(t; \mathbf{X}, \mathbf{U}, \mathbf{B}) = \sum_{k=1}^{\infty} \mathbb{P}^{1k}(t)\mathbf{X}^{[k]}. \tag{10}$$

If we present the solution of (10) using (4), we can write

$$\frac{d\mathbf{X}}{dt} = \sum_{k=1}^{\infty} \mathbb{P}^{1k}(t)\mathbf{X}^{[k]}. \tag{11}$$

Following [8] one can write the solution of (11) in the following form:

$$\mathbf{X}(t) = \sum_{k=1}^{\infty} \mathbb{M}^{1k}(t|t_0)\mathbf{X}_0^{[k]}, \tag{12}$$

$$\frac{d\mathbb{M}^{ik}(t|t_0)}{dt} = \sum_{j=i}^{k} \mathbb{P}^{ij}(t)\mathbb{M}^{jk}(t|t_0),\ 1 \le i < k, \quad \mathbb{P}^{ij} = \mathbb{P}^{1(j-i+1)}\mathbb{P}^{(i-1)(j-1)},$$

$$\mathbb{M}^{11}(t_0|t_0) = \mathbb{E},\ \mathbb{M}^{kk}(t_0|t_0) = \mathbb{E}^{[k]}, \quad \mathbb{M}^{jk}(t_0|t_0) = \mathbb{O}^{jk},\ \forall j \ne k,$$

where $\mathbb{E}$ is the identity matrix, $\mathbb{O}^{jk}$ is a null matrix of corresponding order. Equations (12) can be solved numerically up to necessary order of nonlinearities. This approach can be applied when independent variable t dependence is complex enough, and corresponding symbolic computation can not be produced.

3 Symplectification Problem and Invariants Computation

The Hamiltonian formalism is known to be very popular in physics problems, in particular in accelerator physics. This leads us to the necessity of Hamiltonian nature requirement. As a consequence this requirement leads us to symplectic property of our beam propagator $\mathcal{M}(t|t_0)$ for all $t \geq t_0$.

It is known that symplectic integrators are very powerful tools implemented in most of the tracking codes in accelerator physics. The property of area conservation is particularly suitable for integrating the equations of particle motion over thousands and millions of turns for modern accelerators. The symplectic maps arise from Hamiltonian dynamics because these preserve the loop action. Thus, for example, the time t map of any Hamiltonian flow is symplectic, as is a Poincaré return map defined on a cross section. For example, a circular particle accelerator (storage ring or collider) has a sequence of accelerating and focusing elements that can be modeled by a composition of symplectic maps.

The most popular approach for the dynamics of the particle is based on representing each magnet by a separate Hamiltonian. Its flow $\mathcal{M}$ be computed, i. e., the map linking the particle coordinates at the entrance, $\mathbf{X}(t_0)$ and the current, $\mathbf{X}(t)$: $\mathbf{X}(t) = \mathcal{M}(t|t_0)\mathbf{X}(t_0)$. So the flow of the full ring is then obtained by concatenating the flows of each single element of the ring. Above described matrix formalism is adapted to similar description.

3.1 Two Solution Schemes

Numerical integration algorithms play an essential role in investigation of the long term beam particle evolution, stability of similar process, and nonlinear nonintegrable Hamiltonian systems. Transfer maps generated by Hamiltonian systems obey the symplectic condition, which can be in the following form

$$\mathbb{M}^{\mathrm{T}}(t|t_0)\mathbb{J}_0\mathbb{M}(t|t_0) = \mathbb{J}_0, \quad \forall\, t \geq t_0, \tag{13}$$

where $\mathbb{J}_0 = \begin{pmatrix} \mathbb{O} & \mathbb{E} \\ -\mathbb{E} & \mathbb{O} \end{pmatrix}$ is the so-called canonical symplectic matrix, and $\mathbb{M}$ denotes the Jacobian matrix:

$$\mathbb{M}(t|t_0) = \frac{\partial}{\partial \mathbf{X}_0^{\mathrm{T}}}\Big(\mathcal{M}(t|t_0) \circ \mathbf{X}_0\Big), \tag{14}$$

Unfortunately, standard numerical integration methods are not symplectic, and this violation of the symplectic condition (14) can lead to some false effects, for example, spurious chaotic, dissipative behavior and so on.

There are several approaches devoted to the development of numerical methods preserving some qualitative structure, which is inherent in the dynamical system under study. These schemes will be noted as conservative integration schemes. It is necessary to distinguish two similar types of integration schemes.

The first direction of beam propagator evaluation is based on the universal exponential identities or relations among Lie algebra because these maps have all requisite properties. But its numerical realization loses these properties, and it is necessary to restore desired properties for a numerical variant of this map too. In this report, we take after the second approach.

The second direction is based on the universal schemes, such as different symplectic variants for traditional numerical integration schemes (see, for example, works by J.M. San-Serna [10] and others).

As mentioned above, the beam evolution is defined completely by a sequence of aberration matrices $\mathbb{M}^{1k}$. But for practical calculations, we have to break off for some aberration order N. After this manipulation, the resulting map — the restricted map $\mathcal{M}^N$ loses the symplectic property, which should be recovered. In other words, we should replace the full map $\mathcal{M}$ generated by the sequence of matrices $\left(\mathbb{M}^{11}, \mathbb{M}^{12}, \ldots, \mathbb{M}^{1k}, \ldots\right)$ with reduced map $\mathcal{M}_N$ generated by finite sequence $\left(\mathbb{M}^{11}, \mathbb{M}^{12}, \ldots, \mathbb{M}^{1N}\right)$.

So we have two approaches for map evaluations. The first is based on symbolic representation for $\mathbb{M}^{1k}$, which can be found for some restricted models of $\mathbb{P}^{1j}$. This set of models is defined by functions family, for which there are symbolic solutions appropriate for fringe field distribution. Among them the most familiar is a step function. The symbolic representation for aberration matrices $\mathbb{M}^{1k}$ enables a parametric investigation of the beam propagator. This property is especially useful for optimization problems (see, e. g., [11]). The second approach is based on the numerical solution of differential equations for aberration matrices (see (14)), which can be evaluated for arbitrary forms of fringe fields in the control elements. Here we lose not only the symbolic-specific flexibility, but possibility to keep corresponding solutions in data knowledge as LEGO-objects [12].

3.2 Symplectification Procedures

The both (above described) cases demand symplectic property for the restricted beam propagator $\mathcal{M}^N$. Several symplectic integration methods have been proposed in the literature (see, for example, [13]–[15]). A concept of invariants allows to use the tools of symmetry theory permitting to write any invariant property as an appropriate symmetry condition [6].

In this paper, we shortly discuss a problem of symplectification procedure for restricted map generated by Hamiltonian dynamical systems. For this purpose let us consider the symplectic condition (13) for restricted matrix representation

$$\mathcal{M}_N \mathbf{X}_0 = \sum_{k=1}^{N} \mathbb{M}^{1k} \mathbf{X}_0^{[k]}. \tag{15}$$

In contrast to full series (see (12)), the symplectic condition (13) applied to a short map $\mathcal{M}_N$ will be violated. As an example let us consider one-dimensional motion equations with second-order nonlinearities

$$\mathbf{X}(t) = \mathbb{M}^{11}\mathbf{X}_0 + \mathbb{M}^{12}\mathbf{X}_0^{[2]} = \mathbb{M}^{11}\left(\mathbf{X}_0 + \mathbb{Q}^{12}\mathbf{X}_0^{[2]}\right), \ \dim \mathbf{X} = 2. \tag{16}$$

The symplectic condition leads to linear algebraic homogeneous equations for matrix elements $m_{ij}^k : \mathbb{M}^{1k} = \left\{m_{ij}^k\right\}$. Note that the matrix $\mathbb{M}^{11}$ in (16) is

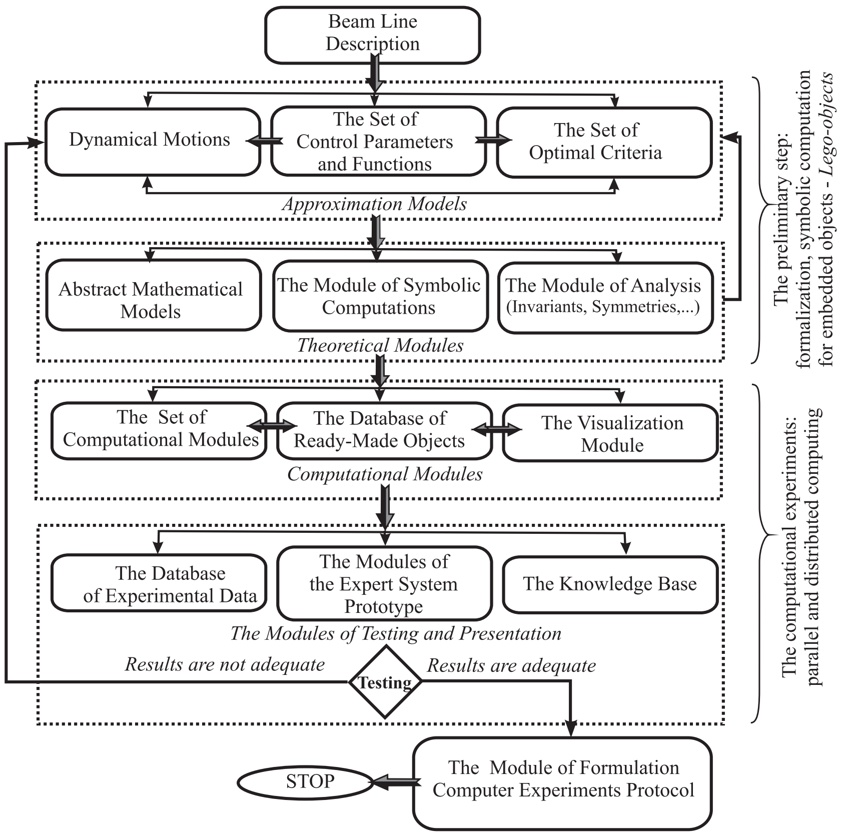

Beam Line Description
Dynamical Motions
The Set of Control Parameters and Functions
The Set of Optimal Criteria
Approximation Models
Abstract Mathematical Models
The Module of Symbolic Computations
The Module of Analysis (Invariants, Symmetries,...)
Theoretical Modules
The Set of Computational Modules
The Database of Ready-Made Objects
The Visualization Module
Computational Modules
The Database of Experimental Data
The Modules of the Expert System Prototype
The Knowledge Base
The Modules of Testing and Presentation
Results are not adequate
Testing
Results are adequate
The Module of Formulation Computer Experiments Protocol
STOP
The preliminary step: formalization, symbolic computation for embedded objects - Lego-objects
The computational experiments: parallel and distributed computing

Fig. 2. The total cycle of the computational experiment

symplectic automatically, and following [8] one can evaluate these equations and write for matrix $\mathbb{Q}^{[2]} = \{q_{ij}\}$ the following:

$$\mathbb{Q}^{12} = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} & q_{15} & q_{16} & q_{17} & q_{18} & q_{19} & q_{110} \\ q_{21} & q_{22} & q_{23} & q_{15} & q_{25} & q_{17} & q_{27} & q_{19}/2 & 2\,q_{110} & q_{210} \\ q_{31} & q_{32} & q_{33} & -2\,q_{11} & -2\,q_{21} & -q_{12} & -q_{22} & q_{14}/2 & -q_{15} & q_{25}/2 \\ q_{32}/2 & 2\,q_{33} & q_{43} & -q_{12} & -q_{22} & q_{32}/2 & 2\,q_{33} & q_{43} & -q_{12} & -q_{22} \end{bmatrix}. \qquad (17)$$

Similar formulas can be obtained for any order of nonlinearities N and dimension of the phase vector $\mathbf{X}$. In addition, these relations reduce the computational costs, indeed, in the case of dim $\mathbf{X} = 4$, for 40 elements of $\mathbb{Q}^{12}$ we obtain 24 restrictions, and for 80 elements for $\mathbb{Q}^{13}$ we obtain 60 restrictions of type (17).

3.3 Kinematic and Dynamic Nonlinear Invariants

Any numerical computational process leads to distortion of qualitative properties (for example, some dynamic and kinematical invariants [4]). These quantities can be evaluated using, for example, Casimirs operators. According to this theory for Lie groups generated by dynamical systems we can construct invariants using special forms and use these data for computational process controlling [8].

The basic types of computational processes are presented in Fig. 2. All presented modules are not only necessary for correct realization of computational experiments, but give a researcher efficient tools for these experiments setup. Usage of symbolic algebra tools give us flexibility and efficiency of the computational process.

3.4 Additional Procedures for Optimization of the Beam Dynamical System

The matrix formalism allows us presenting a wide set of optimal demands in the matrix elements of $\mathbb{M}^{1k}$ up to some kth order [6]. The necessary expressions can easily be evaluated by some symbolic procedures and embedded into computational processes. Moreover, at an initial step of corresponding investigations, the corresponding computational processes can be realized using `Maple` or `Mathematica` tools.

4 Parallel and Distributed Computing in Accelerator Physics

The choice of a matrix formalism as the basic tool for the beam evaluation process allows the use of databases of matrix objects prepared in symbolic and/or numerical modes. Moreover, this aids the construction of efficient numerical and symbolic codes for computational experiments. Here we describe two concepts: *parallel and distributed computing.*

The first type of computational process involves the implementation of homogeneous operations on a set of homogeneous processors. In the second type, the operations having different structure can be computed using a heterogeneous set of processors. This requires to distinguish two types of computational operations: the first of them corresponds to *matrix operations* and the second one to *computational flows* (see, for example, Fig. 3). This separation allows us to distribute a computational experiment over several clusters. Every cluster solves the problems intrinsic in one of the flows. This approach has a bottleneck problem connected with the synchronization of these flows. This problem can be solved using a base of homogeneous mathematical tools — the matrix formalism for Lie methods. The matrix form of practically all required objects allows us to realize parallel computing in a natural way. First of all, parallel computing is realized at the numerical stage when the matrix representation of the current map is built. The second parallelization process is connected with the phase beam portrait construction stage. For this stage, there are several possible approaches

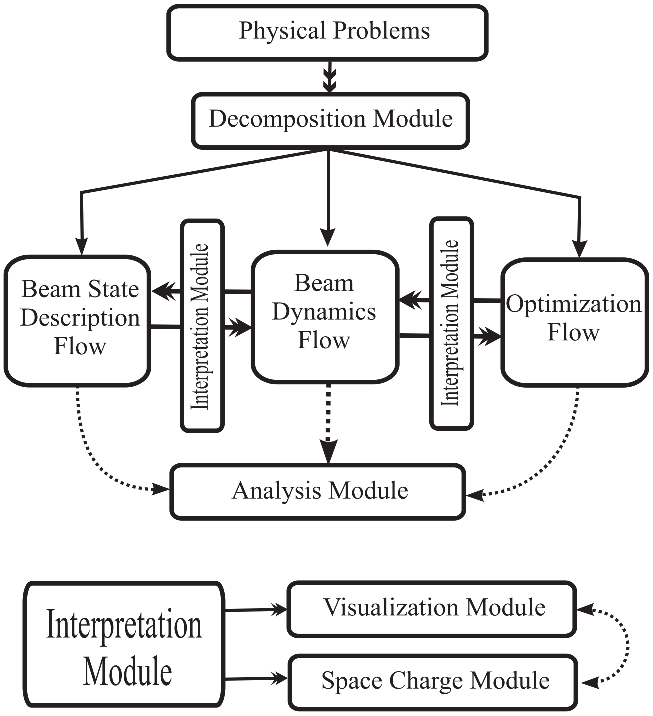


Fig. 3. The types of basic calculational flows

(see, for example, [8]. Moreover, the matrix formalism and symbolic formulae for some matrix operations lead us to more efficient computational procedures for the distributed and parallel computational process.

Additional computational flows are connected with the next two procedures. The first of them is devoted to visualization of all necessary information including auxiliary procedures (for example, analysis of images using differential geometry methods) and space-charge force computing. Here we can use the methods proposed in our previous publications (see, for example, [16,17,18]).

5 Conclusions

The most of described computational flows are realized using Message-Passing Interface (MPI) and deployed on the Linux clusters (placed into the Faculty of Applied Mathematics and Control Processes, SPbSU). It gives students and scientists of the Faculty the affordable time for the circle accelerator complex beam dynamics studies.

References

1. Dragt, A.J.: Lectures on Nonlinear Orbit Dynamics. In: AIP Conf. Proc., vol. 87, pp. 147–313 (1987)
2. Agrachev, A.A., Gamkrelidze, R.V.: Exponential presentation of flows and chronological calculus. Matematicheskii Sbornik 107(149), 467–532 (1978) (in Russian)
3. Magnuss, W.: On the exponential solution of differential equations for a linear operator. Comm. Pure Appl. Math. 7(4), 649–679 (1954)
4. Dragt, A.J., Gluckstern, R.L., Neri, F., Rangarajan, G.: Theory of emittance invariants. Lectures Notes Phys. 343, 94–121 (1989)
5. Andrianov, S.N.: Symbolic computation of approximate symmetries for ordinary differential equations. Mathematics and Computers in Simulation 57(3-5), 147–154 (2001)
6. Andrianov, S.N.: Dynamical Modeling of Control Systems for Particle Beams. In: SPbSU, Saint Petersburg (2004) (in Russian)
7. Andrianov, S.N.: The explicit form for Lie transformations. In: Proc. Fifth European Particle Accelerator Conference EPAC 1996. SITGES, Barcelona, Spain, pp. 998–1000 (1996)
8. Andrianov, S.N.: Matrix representation of the Lie algebraic methods for design of nonlinear beam lines. In: AIP Conf. Proc., New York, vol. 391, pp. 355–360 (1997)
9. Antone, T.A., AL-Maaitah, A.A.: Analytical solutions to classes of linear oscillator equations with time varying frequencies. J. Math. Phys. 33(10), 3330–3339 (1992)
10. Sanz-Serna, J.M., Calvo, M.: Numerical Hamiltonian Problems. Chapman and Hall, London (1994)
11. Andrianov, S., Edamenko, N., Chernyshev, A., Tereshonkov, Y.: Synthesis of optimal nanoprobe. In: EPAC 2008, Genoa, Italy, pp. 2969–2971 (2008)
12. Andrianov, S.N.: LEGO-technology approach for beam line design. In: Proc. Eighth European Particle Accelerator Conference — EPAC 2002, Paris (France), pp. 1607–1609 (2002)
13. Ruth, R.D.: A canonical integration technique. IEEE Trans. Nucl. Sci. 30, 2669 (1983)
14. Yoshida, H.: Construction of high order symplectic integrators. Phys. Lett. A 150, 262 (1990)
15. Forest, É.: Canonical integration as tracking codes (or How to integrate pertubation theory with tracking). In: AIP Conf. Proc., New York, Part 1, vol. 184, p. 2 (1989)
16. Andrianov, S.N.: Parallel computing in beam physics problems. In: Proc. Seventh European Particle Accelerator Conference — EPAC 2000, Vienna (Austria), pp. 1459–1461 (2000)
17. Andrianov, S.N., Edamenko, N.S., Dyatlov, A.A.: Algebraic modeling and parallel computing. Nuclear Instruments and Methods. Ser. A. 558, 150–153 (2006)
18. Andrianov, S., Edamenko, N., Podzivalov, E.: Some problems of global optimization for beam lines. In: Proc. PHYSCON 2009, Catania, Italy, September 1-4, p. 6 (2009)

Thomas Decomposition of Algebraic and Differential Systems

Thomas Bächler[1], Vladimir Gerdt[2], Markus Lange-Hegermann[1], and Daniel Robertz[1]

[1] Lehrstuhl B für Mathematik, RWTH-Aachen University, Germany
thomas@momo.math.rwth-aachen.de, markus@momo.math.rwth-aachen.de, daniel@momo.math.rwth-aachen.de
[2] Joint Institute for Nuclear Research, Dubna, Russia
gerdt@jinr.ru

Abstract. In this paper we consider disjoint decomposition of algebraic and non-linear partial differential systems of equations and inequations into so-called simple subsystems. We exploit Thomas decomposition ideas and develop them into a new algorithm. For algebraic systems simplicity means triangularity, squarefreeness and non-vanishing initials. For differential systems the algorithm provides not only algebraic simplicity but also involutivity. The algorithm has been implemented in Maple.

1 Introduction

Nowadays, triangular decomposition algorithms, which go back to the characteristic set method by Ritt [Rit50] and Wu [Wu00], and software implementing them have become powerful tools for investigating and solving systems of multivariate polynomial equations. In many cases these methods are computationally more efficient than those based on construction of Gröbner bases. As an example of such problems one can indicate Boolean polynomial systems arising in cryptanalysis of stream ciphers. For those systems triangular decomposition algorithms based on the characteristic set method revealed their superiority over the best modern algorithms for the construction of Gröbner bases [sGH09].

For terminology, literature, definitions and basic proofs on triangular-decomposition algorithms for polynomial and differential-polynomial systems we refer to the excellent tutorial papers [Hub03a, Hub03b] and to the bibliographical references therein.

Among numerous triangular decompositions the Thomas one stands by itself. It was suggested by the American mathematician J.M.Thomas in his books [Tho37, Tho62] and decomposes a finite system of polynomial equations and inequations into finitely many triangular subsystems that he called *simple*. Unlike other decomposition algorithms it yields a *disjoint* zero decomposition, that is, it decomposes the affine variety or quasi-affine variety defined by the input into a finite number of disjoint quasi-affine varieties determined by the output simple systems. Every simple system is a regular chain.

V.P. Gerdt et al. (Eds.): CASC 2010, LNCS 6244, pp. 31–54, 2010.

During his research on triangular decomposition, Thomas was motivated by the Riquier-Janet theory [Riq10, Jan29], extending it to non-linear systems of partial differential equations. For this purpose he developed a theory of (Thomas) monomials, which generate the involutive monomial division called Thomas division in [GB98a]. He gave a recipe for decomposing a non-linear differential system into algebraically simple and passive subsystems [Tho37].

Differential Thomas decomposition differs noticeably from that computed by the famous Rosenfeld-Gröbner algorithm [BLOP09, BLOP95] which forms a basis of the diffalg and BLAD libraries [BH04, Bou09] as well as from other differential decompositions (e.g. [BKRM01]). We found that diffalg and BLAD are optimized and well-suited for ordinary differential equations. However, all other known methods give a zero decomposition which, unlike that in Thomas decomposition, is not necessarily disjoint.

A first implementation of the Thomas decomposition was done by Teresa Gómez-Díaz in AXIOM under the name "dynamic constructible closure" which later turned out to be the same as the Thomas decomposition [Del00]. Wang later designed and implemented an algorithm constructing the Thomas decomposition [Wan98, Wan01, LW99]. For polynomial and ordinary differential systems Wang's algorithm was implemented by himself in Maple [Wan04] as part of the software package ϵpsilon [Wan03], which also contains implementations of a number of other triangular decomposition algorithms. A modified algorithmic version of the Thomas decomposition was considered in [Ger08] with its link to the theory of involutive bases [GB98a, Ger05, Ger99]. The latter theory together with some extensions is presented in detail in the recent book [Sei10].

In the given paper we present a new algorithmic version of the Thomas decomposition for polynomial and (partial) differential systems. In the differential case the output subsystems are Janet involutive in accordance to the involutivity criterion from [Ger08], and hence they are coherent. Moreover, for every output subsystem the set of its equations is a minimal Janet basis of the radical differential ideal generated by this set. The algorithm has been implemented in Maple for both the algebraic and differential case. For a linear differential system it constructs a Janet basis of the corresponding differential ideal and for this case works similarly to the Maple package Janet (cf. [BCG$^+$03]).

This paper is organized as follows. In §2 we sketch the algebraic part of our algorithm for the Thomas decomposition with its main objects defined in §2.1. The algorithm itself together with its subalgorithms is considered in §2.2. Decomposition of differential systems is described in §3. Here we briefly introduce some basic notions and concepts from differential algebra (§3.1) and from the theory of involutive bases specific to Janet division (§3.2) together with one of the two extra subalgorithms that extend the algebraic decomposition to the differential one. The second such subalgorithm is considered in §3.3 along with the definition of differential simple systems. Subsection §3.4 contains a description of the differential Thomas decomposition algorithm. Some implementation issues are discussed in §4, where we also demonstrate the Maple implementation for the differential decomposition using the example of a system related to control theory.

We omit the proofs for compactness. They will be published elsewhere.

2 Algebraic Systems

The algebraic THOMAS decomposition deals with systems of polynomial equations and inequations. This section introduces the concepts of simple systems and disjoint decompositions based on properties of the set of solutions of a system. A pseudo reduction procedure and several splitting algorithms on the basis of polynomial remainder sequences are introduced as tools for the main algorithm, which is presented at the end of the section.

2.1 Preliminaries

Let F be a computable field of characteristic 0 and $R := F[x_1, \ldots, x_n]$ the polynomial ring in n variables. A total order $<$ on the indeterminates of R is called a **ranking**. The notation $R = F[x_1, \ldots, x_n]$ shall implicitly define the ranking $x_1 < \ldots < x_n$. The indeterminate x is called **leader** of $p \in R$ if x is the $<$-largest variable occurring in p and we write $\mathrm{ld}(p) = x$. If $p \in F$, we define $\mathrm{ld}(p) = 1$ and $1 < x$ for all indeterminates x. The degree of p in $\mathrm{ld}(p)$ is called **rank** of p and the leading coefficient $\mathrm{init}(p) \in F[\, y \mid y < \mathrm{ld}(p)\,]$ of $\mathrm{ld}(p)^{\mathrm{rank}(p)}$ in p is called **initial** of p.

For $\mathbf{a} \in \overline{F}^n$, where $\overline{F}$ denotes the algebraic closure of F, define the following evaluation homomorphisms:

$$\phi_{\mathbf{a}} : F[x_1, \ldots, x_n] \to \overline{F} : x_i \mapsto a_i$$

$$\phi_{<x_k,\mathbf{a}} : F[x_1, \ldots, x_n] \to \overline{F}[x_k, \ldots, x_n] : \begin{cases} x_i \mapsto a_i,\ i < k \\ x_i \mapsto x_i, \text{ otherwise} \end{cases}$$

For a polynomial $p \in R$, the symbols $p_=$ and $p_{\neq}$ shall denote the equation $p = 0$ and inequation $p \neq 0$, respectively. A finite set of equations and inequations is called an **(algebraic) system** over R. Abusing notation, we sometimes treat $p_=$ or $p_{\neq}$ as the underlying polynomial p. A **solution** of a system S is a tuple $\mathbf{a} \in \overline{F}^n$ such that $\phi_{\mathbf{a}}(p) = 0$ for all equations $p_= \in S$ and $\phi_{\mathbf{a}}(p) \neq 0$ for all inequations $p_{\neq} \in S$. The set of all solutions of S is denoted by $\mathfrak{Sol}(S)$.

Define $S_x := \{p \in S \mid \mathrm{ld}(p) = x\}$. In a situation where it is clear that $|S_x| = 1$, we also use S_x to denote the unique element of S_x. The subset $S_{<x} := \{p \in S \mid \mathrm{ld}(p) < x\}$ can be considered a system over $F[\, y \mid y < x\,]$. Furthermore, the sets of all equations $p_= \in S$ and all inequations $p_{\neq} \in S$ are denoted by $S^=$ and $S^{\neq}$, respectively.

The general idea of the THOMAS methods is to use the homomorphism $\phi_{<x,\mathbf{a}}$ to treat each polynomial $p \in S_x$ as the *univariate* polynomial $\phi_{<x,\mathbf{a}}(p) \in \overline{F}[x]$ for all $\mathbf{a} \in \mathfrak{Sol}(S_{<x})$ *simultaneously.* This idea forms the basis of our central object, the **simple system**:

Definition 2.1 (Simple Systems). Let S be a system.

1. S is **triangular** if $|S_{x_i}| \leq 1 \ \forall\ 1 \leq i \leq n$ and $S \cap \{c_=, c_{\neq} \mid c \in F\} = \emptyset$.
2. S has **non-vanishing initials** if $\phi_{\mathbf{a}}(\mathrm{init}(p)) \neq 0 \ \forall\ \mathbf{a} \in \mathfrak{Sol}(S_{<x_i})$ and $p \in S_{x_i}$ for $1 \leq i \leq n$.
3. S is **square-free** if the univariate polynomial $\phi_{<x_i,\mathbf{a}}(p) \in \overline{F}[x_i]$ is square-free $\forall\ \mathbf{a} \in \mathfrak{Sol}(S_{<x_i})$ and $p \in S_{x_i}$ for $1 \leq i \leq n$.
4. S is called **simple** if it is *triangular*, has *non-vanishing initials* and is *square-free.*

Although all required properties are characterized via solutions of lower-ranking equations and inequations, the Thomas decomposition algorithm does not calculate solutions of polynomials. Instead, it uses polynomial equations and inequations to *partition* the set of solutions of the lower-ranking system to ensure the above properties.

Remark 2.2. Simplicity of a system guarantees the existence of solutions: If $\mathbf{b} \in \mathfrak{Sol}(S_{<x})$ and S_x is not empty, then $\phi_{<x,\mathbf{b}}(S_x)$ is a univariate polynomial with exactly $\mathrm{rank}(S_x)$ *distinct* roots. When extending $\mathbf{b}$ to a solution $(\mathbf{b}, a)$ of $S_{\leq x}$, for an equation in S_x there are $\mathrm{rank}(S_x)$ choices for a, whereas for an inequation or empty S_x all but finitely many $a \in \overline{F}$ give an extension.

To transform a system into a simple system, it is in general necessary to partition the set of solutions. Instead of an equivalent simple system, this leads to a so-called decomposition into simple systems.

Definition 2.3. A family $(S_i)_{i=1}^m$ is called **decomposition** of S if $\mathfrak{Sol}(S) = \bigcup_{i=1}^m \mathfrak{Sol}(S_i)$. A decomposition is called **disjoint** if $\mathfrak{Sol}(S_i) \cap \mathfrak{Sol}(S_j) = \emptyset \ \forall\ i \neq j$. A *disjoint* decomposition of a system into *simple systems* is called **(algebraic) Thomas decomposition**.

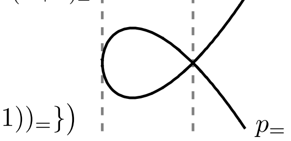
For any algebraic system S, there exists a Thomas decomposition (cf. [Tho37], [Tho62], [Wan98]). The algorithm presented in the following section provides another proof of this fact. First, we give an easy example of a Thomas decomposition.

Example 2.4. Consider the equation

$$p = y^2 - x^3 - x^2 \ .$$

A Thomas decomposition of $\{p_=\}$ is given by:

$$\left(\{(y^2 - x^3 - x^2)_=, (x \cdot (x+1))_{\neq}\}, \{y_=, (x \cdot (x+1))_=\}\right)$$



2.2 Decomposition Algorithms

Our version of the decomposition algorithm in each round treats one system, potentially splitting it into several subsystems. For this purpose, one polynomial is chosen from a list of polynomials to be processed. This polynomial is pseudo-reduced modulo the system and afterwards combined with the polynomial in the system having the same leader. To ensure that all polynomials are square-free and their initials do not vanish, the system may be split into several ones by initials of polynomials or subresultants.

From now on, a system S is presented as a pair of sets (S_T, S_Q), where S_T represents a candidate for a simple system while S_Q is the queue of elements to be processed. S_T is always triangular and $(S_T)_x$ denotes the unique equation or inequation of leader x in S_T, if any. S_T also fulfills a weaker form of the other two simplicity conditions, i.e., for any solution $\mathbf{a}$ of $(S_T)_{<x} \cup (S_Q)_{<x}$, we have $\phi_{\mathbf{a}}(\mathrm{init}((S_T)_x)) \neq 0$ and $\phi_{<x,\mathbf{a}}((S_T)_x)$ is square-free.

From now on, let prem be a **pseudo remainder algorithm**[1] in R and pquo the corresponding **pseudo quotient algorithm**, i.e., for p and q with $\mathrm{ld}(p) = \mathrm{ld}(q) = x$

$$m \cdot p = \mathsf{pquo}(p, q, x) \cdot q + \mathsf{prem}(p, q, x) \tag{1}$$

where $\deg_x(q) > \deg_x(\mathsf{prem}(p, q, x))$ and $m \in R \setminus \{0\}$ with $\mathrm{ld}(m) < x$ and $m \mid \mathrm{init}(q)^k$ for some $k \in \mathbb{Z}_{\geq 0}$. Note that if the initials of p and q are non-zero, the initial of $\mathsf{pquo}(p, q, x)$ is also non-zero. Equation (1) only allows us to replace p with $\mathsf{prem}(p, q, x)$ if m does not vanish on any solution. The below Algorithm (2.5) and Remark (2.6) require the last property, which, by definition, holds in simple systems.

The following algorithm employs pseudo remainders and the triangular structure to reduce a polynomial modulo S_T:

Algorithm 2.5 (Reduce).
Input: A system S, a polynomial $p \in R$
Output: A polynomial q with $\phi_{\mathbf{a}}(p) = 0$ if and only if $\phi_{\mathbf{a}}(q) = 0$ for each $\mathbf{a} \in \mathfrak{Sol}(S)$.
Algorithm:

```
 1: $x \leftarrow \mathrm{ld}(p)$; $q \leftarrow p$
 2: while $x > 1$ and $(S_T)_x$ is an equation and $\mathrm{rank}(q) \geq \mathrm{rank}((S_T)_x)$ do
 3:     $q \leftarrow \mathsf{prem}(q, (S_T)_x, x)$
 4:     $x \leftarrow \mathrm{ld}(q)$
 5: end while
 6: if $x > 1$ and $\mathsf{Reduce}(S, \mathrm{init}(q)) = 0$ then
 7:     return $\mathsf{Reduce}(S, q - \mathrm{init}(q)x^{\mathrm{rank}(q)})$
 8: else
 9:     return $q$
10: end if
```

A polynomial p is called **reduced modulo** S_T if $\mathsf{Reduce}(S, p) = p$. A polynomial p **reduces to** q **modulo** S_T if $\mathsf{Reduce}(S, p) = q$.

The result of the Reduce algorithm does not need to be a canonical normal form. It only needs to detect polynomials that vanish on all solutions of a system:

[1] In our context prem does not necessarily have to be the classical pseudo remainder, but any sparse pseudo remainder with property (1) will suffice.

Remark 2.6. Let $p \in R$ with $\mathrm{ld}(p) = x$. $\mathsf{Reduce}(S, p) = 0$ implies $\phi_{\mathbf{a}}(p) = 0 \ \forall \ \mathbf{a} \in \mathfrak{Sol}(S_{\leq x})$.

The converse of this remark only holds if $(S_Q)_{\leq x} = \emptyset$, i.e., $(S_T)_{\leq x}$ is simple. If it is not simple, but $\mathrm{ld}(p) = x$ and $(S_Q)^{=}_{<x} = \emptyset$ hold, we still have some information. In particular, $\mathsf{Reduce}(S, p) \neq 0$ implies that either $\mathfrak{Sol}(S_{<x})$ is empty or there exists $\mathbf{a} \in \mathfrak{Sol}(S_{<x} \cup \{(S_T)_x\})$ such that $\phi_{\mathbf{a}}(p) \neq 0$.

We now direct our attention to the methods we use to produce disjoint decompositions. Since $(S \cup \{p_{\neq}\}, S \cup \{p_{=}\})$ is a disjoint decomposition of S, we will use the following one-line subalgorithm as the basis of all the splitting algorithms described below.

Algorithm 2.7 (Split). *Input:* A system S, a polynomial $p \in R$
Output: The disjoint decomposition $(S \cup \{p_{\neq}\}, S \cup \{p_{=}\})$ of S.
Algorithm:

```
1: return  $((S_T, S_Q \cup \{p_{\neq}\}), (S_T, S_Q \cup \{p_{=}\}))$
```

The output of the following splitting algorithms is not yet a disjoint decomposition of the input. However, the main algorithm $\mathsf{Decompose}$ will use this output to construct a disjoint decomposition. We single out these algorithms to make the main algorithm more compact and readable. For details we refer to the input and output specifications of the algorithms in question.

The algorithm $\mathsf{InitSplit}$ ensures that in one of the returned systems the property 2 in Definition (2.1) holds for the input polynomial. In the other system the initial of that polynomial vanishes.

Algorithm 2.8 ($\mathsf{InitSplit}$). *Input:* A system S, an equation or inequation q with $\mathrm{ld}(q) = x$.
Output: Two systems S_1 and S_2, where $(S_1 \cup \{q\}, S_2)$ is a disjoint decomposition of $S \cup \{q\}$. Moreover, $\phi_{\mathbf{a}}(\mathrm{init}(q)) \neq 0$ holds for all $\mathbf{a} \in \mathfrak{Sol}(S_1)$ and $\phi_{\mathbf{a}}(\mathrm{init}(q)) = 0$ for all $\mathbf{a} \in \mathfrak{Sol}(S_2)$. *Algorithm:*

```
1: $(S_1, S_2) \leftarrow \mathsf{Split}(S, \mathrm{init}(q))$
2: if $q$ is an equation then
3:     $(S_2)_Q \leftarrow (S_2)_Q \cup \{(q - \mathrm{init}(q) x^{\mathrm{rank}(q)})_{=}\}$
4: else if $q$ is an inequation then
5:     $(S_2)_Q \leftarrow (S_2)_Q \cup \left\{(q - \mathrm{init}(q) x^{\mathrm{rank}(q)})_{\neq}\right\}$
6: end if
7: return  $(S_1, S_2)$
```

In Definition (2.1) we view a multivariate polynomial p as the univariate polynomial $\phi_{<\mathrm{ld}(p), \mathbf{a}}(p)$. For ensuring triangularity and square-freeness, we often compute the gcd of two polynomials, which generally depends on the inserted value $\mathbf{a}$. Subresultants provide a generalization of the Euclidean algorithm useful in our context and their initials distinguish the cases of different degrees of gcds.

Definition 2.9. Let $p, q \in R$ with $\mathrm{ld}(p) = \mathrm{ld}(q) = x$, $\deg_x(p) = d_p > \deg_x(q) = d_q$. We denote by $\mathrm{PRS}(p, q, x)$ the **subresultant polynomial remainder sequence** (see [Hab48], [Mis93, Chap. 7], [Yap00, Chap. 3]) of p and q w.r.t. x, and by $\mathrm{PRS}_i(p,q,x)$, $i < d_q$ the regular polynomial of degree i in $\mathrm{PRS}(p,q,x)$ if it exists, or 0 otherwise. Furthermore, $\mathrm{PRS}_{d_p}(p,q,x) := p$, $\mathrm{PRS}_{d_q}(p,q,x) := q$ and $\mathrm{PRS}_i(p,q,x) := 0$, $d_q < i < d_p$.

Define $\mathrm{res}_i(p,q,x) := \mathrm{init}\,(\mathrm{PRS}_i\,(p,q,x))$ for $0 < i < d_p$, whereas $\mathrm{res}_{d_p}(p,q,x) := 1$ and $\mathrm{res}_0(p,q,x) := \mathrm{PRS}_0\,(p,q,x)$. Note that $\mathrm{res}(p,q,x) := \mathrm{res}_0(p,q,x)$ is the usual resultant.

Definition 2.10. Let S be a system and $p_1, p_2 \in R$ with $\mathrm{ld}(p_1) = \mathrm{ld}(p_2) = x$. If $|\mathfrak{Sol}(S_{<x})| > 0$, we call

$$i := \min\{i \in \mathbb{Z}_{\geq 0} \mid \exists\, \mathbf{a} \in \mathfrak{Sol}(S_{<x}) \text{ such that } \deg_x(\gcd(\phi_{<x,\mathbf{a}}(p_1), \phi_{<x,\mathbf{a}}(p_2))) = i\}$$

the **fiber cardinality** of p_1 and p_2 w.r.t. S. Moreover, if $(S_Q)^{=}_{<x} = \emptyset$, then

$$\begin{aligned} i' := \min\{i \in \mathbb{Z}_{\geq 0} \mid\ & \mathsf{Reduce}(\mathrm{res}_j(p_1,p_2,x), S_T) = 0\ \forall\ j < i \\ & \text{and } \mathsf{Reduce}(\mathrm{res}_i(p_1,p_2,x), S_T) \neq 0\} \end{aligned}$$

is the **quasi fiber cardinality** of p_1 and p_2 w.r.t. S. A disjoint decomposition (S_1, S_2) of S such that

1. $\deg_x(\gcd(\phi_{<x,\mathbf{a}}(p_1), \phi_{<x,\mathbf{a}}(p_2))) = i\ \forall\ \mathbf{a} \in \mathfrak{Sol}((S_1)_{<x})$
2. $\deg_x(\gcd(\phi_{<x,\mathbf{a}}(p_1), \phi_{<x,\mathbf{a}}(p_2))) > i\ \forall\ \mathbf{a} \in \mathfrak{Sol}((S_2)_{<x})$

is called the i-th **fibration split** of p_1 and p_2 w.r.t. S. A polynomial $r \in R$ with $\mathrm{ld}(r) = x$ such that $\deg_x(r) = i$ and

$$\phi_{<x,\mathbf{a}}(r) \sim \gcd(\phi_{<x,\mathbf{a}}(p_1), \phi_{<x,\mathbf{a}}(p_2))\ \forall\ \mathbf{a} \in \mathfrak{Sol}((S_1)_{<x})$$

is called the i-th **conditional greatest common divisor** of p_1 and p_2 w.r.t. S, where $p \sim q$ if and only if $p \in \overline{K}^* q$. Furthermore, $q \in R$ with $\mathrm{ld}(q) = x$ and $\deg_x(q) = \deg_x(p_1) - i$ such that

$$\phi_{<x,\mathbf{a}}(q) \sim \frac{\phi_{<x,\mathbf{a}}(p_1)}{\gcd(\phi_{<x,\mathbf{a}}(p_1), \phi_{<x,\mathbf{a}}(p_2))}\ \forall\ \mathbf{a} \in \mathfrak{Sol}((S_1)_{<x})$$

is called the i-th **conditional quotient** of p_1 by p_2 w.r.t. S. By replacing $\phi_{<x,\mathbf{a}}(p_2)$ in the above definition with $\frac{\partial}{\partial x}(\phi_{<x,\mathbf{a}}(p_1))$, we get an i-th **square-free split** and i-th **conditional square-free part** of p_1 w.r.t. S.

The fiber cardinality is often not immediately available, as we may be unable to take inequations into account. However, we can use all information contained in the equations using reduction, if all equations are contained in S_T. Thus we require $(S_Q)^{=}_{<x} = \emptyset$ before doing any reduction.

In this situation, the quasi fiber cardinality is easy to calculate and in many cases will be identical to the fiber cardinality. Furthermore, if we consider the system S_2 from an i-th fibration split of some polynomials for a system S and ensure that $((S_2)_Q)^{=}_{<x} = \emptyset$, then the quasi fiber cardinality of the same polynomials for S_2 will be $i + 1$. Therefore and due to the following lemma, the quasi fiber cardinality is good enough for our purposes.

Lemma 2.11. Let $|\mathfrak{Sol}(S_{<x})| > 0$ and $(S_Q)^{=}_{<x} = \emptyset$. For p_1, p_2 as in Definition (2.10) with $\phi_{\mathbf{a}}(\mathrm{init}(p_1)) \neq 0 \;\forall\; \mathbf{a} \in \mathfrak{Sol}(S_{<x})$ and $\mathrm{rank}(p_1) > \mathrm{rank}(p_2)$, let i be the fiber cardinality of p_1 and p_2 w.r.t. S and i' the corresponding quasi fiber cardinality. Then

$$i' \leq i$$

where the equality holds if and only if $|\mathfrak{Sol}\,(S_{<x} \cup \{\mathrm{res}_{i'}(p_1, p_2, x)_{\neq}\})| > 0$.

Corollary 2.12. Let $|\mathfrak{Sol}(S_{<x})| > 0$ and $(S_Q)^{=}_{<x} = \emptyset$. For polynomials p_1, p_2 as in Definition (2.10) with $\phi_{\mathbf{a}}(\mathrm{init}(p_1)) \neq 0$ and $\phi_{\mathbf{a}}(\mathrm{init}(p_2)) \neq 0 \;\forall\; \mathbf{a} \in \mathfrak{Sol}(S_{<x})$, let i be the fiber cardinality of p_1 and p_2 w.r.t. S and i' the quasi fiber cardinality of p_1 and $\mathsf{prem}(p_2, p_1, x)$ w.r.t. S. Then

$$i' \leq i$$

with equality if and only if $|\mathfrak{Sol}\,(S_{<x} \cup \{\mathrm{res}_{i'}(p_1, \mathsf{prem}(p_2, p_1, x), x)_{\neq}\})| > 0$.

The following algorithm calculates the quasi fiber cardinality of two polynomials. It is used as the basis for all algorithms that calculate a greatest common divisor or a least common multiple.

Algorithm 2.13 (ResSplit). *Input:* A system S with $(S_Q)^{=}_{<x} = \emptyset$, two polynomials $p, q \in R$ with $\mathrm{ld}(p) = \mathrm{ld}(q) = x$, $\mathrm{rank}(p) > \mathrm{rank}(q)$ and $\phi_{\mathbf{a}}(\mathrm{init}(p)) \neq 0$ for all $\mathbf{a} \in \mathfrak{Sol}(S_{<x})$.
Output: The quasi fiber cardinality i of p and q w.r.t. S and an i-th fibration split (S_1, S_2) of p and q w.r.t. S.
Algorithm:

```
1: $i \leftarrow \min\{i \in \mathbb{Z}_{\geq 0} \mid \mathsf{Reduce}(\mathrm{res}_j(p, q, x), S_T) = 0 \;\forall\; j < i$
                      and $\mathsf{Reduce}(\mathrm{res}_i(p, q, x), S_T) \neq 0\}$
2: return $(i, S_1, S_2) := (i, \mathsf{Split}(S, \mathrm{res}_i(p, q, x)))$
```

Similarly to the InitSplit algorithm (2.8), the following algorithm does not return a disjoint decomposition, but Decompose uses it to construct one.

Algorithm 2.14 (ResSplitGCD). *Input:* A system S with $(S_Q)^{=}_{<x} = \emptyset$, where $(S_T)_x$ is an equation, and an equation $q_=$ with $\mathrm{ld}(q) = x$. Furthermore $\mathrm{rank}(q) < \mathrm{rank}((S_T)_x)$.
Output: Two systems S_1 and S_2 and an equation $\widetilde{q}_=$ such that:

a) $S_2 = \widetilde{S_2} \cup \{q\}$ where $\left(S_1, \widetilde{S_2}\right)$ is an i-th fibration split of $(S_T)_x$ and q w.r.t. S
b) $\widetilde{q}$ is an i-th conditional gcd of $(S_T)_x$ and q w.r.t. S.

where i is the quasi fiber cardinality of p and q w.r.t. S.

Algorithm:

```
1: $(i, S_1, S_2) \leftarrow \mathsf{ResSplit}\,(S, (S_T)_x, q)$
2: $(S_2)_Q \leftarrow (S_2)_Q \cup \{q\}$
3: return $S_1, S_2, \mathrm{PRS}_i((S_T)_x, q, x)_=$
```

The following algorithm is similar, but instead of the gcd, it returns the first input polynomial divided by the gcd. It is used to assimilate an inequation into a system where there already is an equation with the same leader, or to calculate the least common multiple of two inequations.

Algorithm 2.15 (ResSplitDivide). *Input:* A system S with $(S_Q)^{=}_{<x} = \emptyset$ and two polynomials p, q with $\mathrm{ld}(p) = \mathrm{ld}(q) = x$ and $\phi_{\mathbf{a}}(\mathrm{init}(p)) \neq 0$ for all $\mathbf{a} \in \mathfrak{Sol}(S_{<x})$. Furthermore, if $\mathrm{rank}(p) \leq \mathrm{rank}(q)$ then $\phi_{\mathbf{a}}(\mathrm{init}(q)) \neq 0$.
Output: Two systems S_1 and S_2 and a polynomial $\widetilde{p}$ such that:

a) $S_2 = \widetilde{S_2} \cup \{q\}$ where $\left(S_1, \widetilde{S_2}\right)$ is an i-th fibration split p and q' w.r.t. S
b) $\widetilde{p}$ is an i-th conditional quotient of p by q' w.r.t. S

where i is the quasi fiber cardinality of p and q' w.r.t. S, with $q' = q$ for $\mathrm{rank}(p) > \mathrm{rank}(q)$ and $q' = \mathsf{prem}(q, p, x)$ otherwise.

Algorithm:

```
 1: if $\mathrm{rank}(p) \leq \mathrm{rank}(q)$ then
 2:     return ResSplitDivide$(S, p, \mathsf{prem}(q, p, x))$
 3: else
 4:     $(i, S_1, S_2) \leftarrow$ ResSplit$(S, p, q)$
 5:     if $i > 0$ then
 6:         $\widetilde{p} \leftarrow \mathsf{pquo}(p, \mathrm{PRS}_i(p, \mathsf{prem}(q, p, x), x), x)$
 7:     else
 8:         $\widetilde{p} \leftarrow p$
 9:     end if
10:     $(S_2)_Q \leftarrow (S_2)_Q \cup \{q\}$
11:     return $S_1, S_2, \widetilde{p}$
12: end if
```

Applying the last algorithm to a polynomial p and its partial derivative by its leader yields an algorithm to make polynomials square-free.

In the above ResSplit-based algorithms, we had the requirement that $(S_Q)^{=}_{<x} = \emptyset$. This ensures that all information contained in any equation of a smaller leader than x will be respected by reduction modulo S_T and thus avoids creating redundant systems. It will also be necessary for termination of the Decompose algorithm. This motivates the definition of a selection strategy as follows.

Definition 2.16 (Select). Let $\mathbb{P}_{\mathrm{finite}}(M)$ be the set of all finite subsets of a set M. A **selection strategy** is a map

$$\begin{aligned}\mathsf{Select} : \mathbb{P}_{\mathrm{finite}}(\{p_{=}, p_{\neq} \mid p \in R\}) &\longrightarrow \{p_{=}, p_{\neq} \mid p \in R\} : \\ Q &\longmapsto q \in Q\end{aligned}$$

with the following properties:

1. If $\mathsf{Select}(Q) = q_{=}$ is an equation, then $Q^{=}_{<\mathrm{ld}(q)} = \emptyset$.
2. If $\mathsf{Select}(Q) = q_{\neq}$ is an inequation, then $Q^{=}_{\leq\mathrm{ld}(q)} = \emptyset$.

The second property of Select could be weakened further, i.e., $Q^{=}_{<\mathrm{ld}(q)} = \emptyset$. However, this would result in redundant calculations in the Decompose algorithm, thus we want all equations of the same leader to be treated first.

The following algorithm is trivial. However, it will be replaced with a more complicated algorithm in §3 when the differential Thomas decomposition is treated.

Algorithm 2.17 (InsertEquation). *Input:* A system S and an equation $r_=$ with $\mathrm{ld}(r) = x$ satisfying $\phi_{\mathbf{a}}(\mathrm{init}(r)) \neq 0$ and $\phi_{<x,\mathbf{a}}(r)$ square-free for all $\mathbf{a} \in \mathfrak{Sol}(S_{<x})$.
Output: A system S where $r_=$ is inserted into S_T.
Algorithm:

```
1: if $(S_T)_x$ is not empty then
2:     $S_T \leftarrow (S_T \setminus \{(S_T)_x\})$
3: end if
4: $S_T \leftarrow S_T \cup \{r_=\}$
5: return $S$
```

Now we present the main algorithm. It is based on all above algorithms and yields an algebraic Thomas decomposition. This algorithm forms the basis of the differential Thomas decomposition to be discussed in detail in §3.

The general structure is as follows: In each iteration, a system S is selected from a list P of unfinished systems. An equation or inequation q is chosen from S_Q according to the selection strategy and reduced modulo S_T. The algorithm assimilates q into S_T, potentially adding inequations of lower leader to S_Q and adding new systems S_i to P that contain a new equation of lower leader in $(S_i)_Q$. This process works differently depending on whether q and $(S_T)_{\mathrm{ld}(q)}$ are equations or inequations, but it is based on the InitSplit, ResSplitGcd and ResSplitDivide methods in all cases. As soon as the algorithm yields an equation $c = 0$ for $c \in F \setminus \{0\}$ or an inequation $0 \neq 0$ in a system, this system is inconsistent and thus discarded.

Algorithm 2.18 (Decompose). The algorithm is printed on page 41.

In the next section, we consider an extension of this algorithm to partial differential systems. Both algorithms have been implemented, and their implementation aspects are considered in §4.

3 The Differential Thomas Decomposition

The differential Thomas decomposition is concerned with manipulations of polynomial differential equations. The basic idea for a construction of this decomposition is twofold. On the one hand a combinatorial calculus developed by Janet takes care of finding unique reductors and all differential consequences by completing systems to involution. On the other hand the algebraic Thomas decomposition makes the necessary splits for regularity of initials during the computation and ensures disjointness.

Algorithm 2.18 (Decompose)

Input: A system S' with $(S')_T = \emptyset$.
Output: A Thomas decomposition of S'.
Algorithm:

```
1: $P \leftarrow \{S'\}$; $Result \leftarrow \emptyset$
2: while $|P| > 0$ do
3:    Choose $S \in P$; $P \leftarrow P \setminus \{S\}$
4:    if $|S_Q| = 0$ then
5:       $Result \leftarrow Result \cup \{S\}$
6:    else
7:       $q \leftarrow \mathsf{Select}(S_Q)$; $S_Q \leftarrow S_Q \setminus \{q\}$
8:       $q \leftarrow \mathsf{Reduce}(q, S_T)$; $x \leftarrow \mathrm{ld}(q)$
9:       if $q \notin \{0_{\neq}, c_= \mid c \in F \setminus \{0\}\}$ then
10:         if $x \neq 1$ then
11:            if $q$ is an equation then
12:               if $(S_T)_x$ is an equation then
13:                  if $\mathsf{Reduce}(\mathrm{res}_0((S_T)_x, q, x), S_T) = 0$ then
14:                     $(S, S_1, p) \leftarrow \mathsf{ResSplitGCD}(S, q, x)$; $P \leftarrow P \cup \{S_1\}$
15:                     $S \leftarrow \mathsf{InsertEquation}(S, p_=)$
16:                  else
17:                     $S_Q \leftarrow S_Q \cup \{q_=, \mathrm{res}_0((S_T)_x, q, x)_=\}$
18:                  end if
19:               else
20:                  if $(S_T)_x$ is an inequation[a] then
21:                     $S_Q \leftarrow S_Q \cup \{(S_T)_x\}$; $S_T \leftarrow S_T \setminus \{(S_T)_x\}$
22:                  end if
23:                  $(S, S_2) \leftarrow \mathsf{InitSplit}(S, q)$; $P \leftarrow P \cup \{S_2\}$
24:                  $(S, S_3, p) \leftarrow \mathsf{ResSplitDivide}\left(S, q, \frac{\partial}{\partial x} q\right)$; $P \leftarrow P \cup \{S_3\}$
25:                  $S \leftarrow \mathsf{InsertEquation}(S, p_=)$
26:               end if
27:            else if $q$ is an inequation then
28:               if $(S_T)_x$ is an equation then
29:                  $(S, S_4, p) \leftarrow \mathsf{ResSplitDivide}(S, (S_T)_x, q)$; $P \leftarrow P \cup \{S_4\}$
30:                  $S \leftarrow \mathsf{InsertEquation}(S, p_=)$
31:               else
32:                  $(S, S_5) \leftarrow \mathsf{InitSplit}(S, q)$; $P \leftarrow P \cup \{S_5\}$
33:                  $(S, S_6, p) \leftarrow \mathsf{ResSplitDivide}\left(S, q, \frac{\partial}{\partial x} q\right)$; $P \leftarrow P \cup \{S_6\}$
34:                  if $(S_T)_x$ is an inequation then
35:                     $(S, S_7, r) \leftarrow \mathsf{ResSplitDivide}(S, (S_T)_x, p)$; $P \leftarrow P \cup \{S_7\}$
36:                     $(S_T)_x \leftarrow (r \cdot p)_{\neq}$
37:                  else if $(S_T)_x$ is empty then
38:                     $(S_T)_x \leftarrow p_{\neq}$
39:                  end if
40:               end if
41:            end if
42:         end if
43:         $P \leftarrow P \cup \{S\}$
44:      end if
45:   end if
46: end while
47: return $Result$
```

[a] Remember that $(S_T)_x$ might be empty, and thus neither an equation nor an inequation.

We start by giving the basic definitions from differential algebra needed for the algorithms. Afterwards we summarize the JANET division and its combinatorics. The combinatorics give us a new algorithm InsertEquation to add equations into systems. Afterwards we review the differential implications of the algebraic decomposition algorithm and present the algorithm Reduce utilized for differential reduction. Replacing the insertion and reduction methods from the algebraic case with these differential methods yields the differential decomposition algorithm.

3.1 Preliminaries from Differential Algebra

Let $\Delta = \{\partial_1, \ldots, \partial_n\}$ be the set of derivations ($n > 0$) and F be a computable **Δ-differential field** of characteristic zero. This means any $\partial_j \in \Delta$ is a linear operator $\partial_j : F \to F$ fulfilling the LEIBNIZ rule. For a **differential indeterminate** u consider the **Δ-differential polynomial ring** $F\{u\} := F\left[\, u_{\mathbf{i}} \mid \mathbf{i} \in \mathbb{Z}^n_{\geq 0} \,\right]$, a polynomial ring infinitely generated by the algebraically independent set $\langle u \rangle_\Delta := \{u_{\mathbf{i}} \mid \mathbf{i} \in \mathbb{Z}^n_{\geq 0}\}$. The operation of $\partial_j \in \Delta$ on $\langle u \rangle_\Delta$ by $\partial_j u_{\mathbf{i}} = u_{\mathbf{i}+e_j}$ is extended linearly and by the LEIBNIZ rule to $F\{u\}$. Let $U = \{u^{(1)}, \ldots, u^{(m)}\}$ be the set of differential indeterminates. The multivariate Δ-differential polynomial ring is given by $F\{U\} := F\{u^{(1)}\} \ldots \{u^{(m)}\}$. The elements of $\langle U \rangle_\Delta := \left\{ u^{(j)}_{\mathbf{i}} \mid \mathbf{i} \in \mathbb{Z}^n_{\geq 0}, j \in \{1, \ldots, m\} \right\}$ are called **differential variables**.

We remark, that the algebraic closure $\overline{F}$ of F is a differential field with a differential structure uniquely defined by the differential structure of F (cf. [Kol73, §II.2, Lemma 1]). Let

$$E := \bigoplus_{j=1}^{m} \overline{F}[[z_1, \ldots, z_n]] \cong \overline{F}^{\langle U \rangle_\Delta}$$

with indeterminates $z_1, \ldots, z_n$, where $\overline{F}[[z_1, \ldots, z_n]]$ denotes the ring of formal power series. The isomorphism maps coefficients of the power series to function values of differential variables, i.e.,

$$\alpha \colon \bigoplus_{i=1}^{m} \overline{F}[[z_1, \ldots, z_n]] \to \overline{F}^{\langle U \rangle_\Delta} : \left(\sum_{\mathbf{i} \in \mathbb{Z}^n_{\geq 0}} a^{(1)}_{\mathbf{i}} \frac{z^{\mathbf{i}}}{\mathbf{i}!}, \ldots, \sum_{\mathbf{i} \in \mathbb{Z}^n_{\geq 0}} a^{(m)}_{\mathbf{i}} \frac{z^{\mathbf{i}}}{\mathbf{i}!} \right) \mapsto \left(u^{(j)}_{\mathbf{i}} \mapsto a^{(j)}_{\mathbf{i}} \right)$$

where $\mathbf{i}! := i_1! \cdot \ldots \cdot i_n!$ defines the factorial of a multi-index.

In the formulation of the algorithm the direct sum of formal power series E suffices to give a notion of solutions coherent to the algebraic case: For $e \in E$ we define the F-algebra homomorphism

$$\phi_e : F\{U\} \to \overline{F} : u^{(j)}_{\mathbf{i}} \mapsto \alpha(e)(u^{(j)})$$

evaluating all differential variables of a differential polynomial at the power series e. A **differential equation** or **inequation** for m functions $U = \{u^{(1)}, \ldots, u^{(m)}\}$ in n indeterminates is an element $p \in F\{U\}$ written $p_=$ or $p_{\neq}$, respectively. A

solution of $p_=$ or $p_{\neq}$ is an $e \in E$ with $\phi_e(p) = 0$ or $\phi_e(p) \neq 0$, respectively. More generally $e \in E$ is called a solution of a set P of equations and inequations, if it is a solution of each element in P. The set of solutions of P is denoted by $\mathfrak{Sol}_E(P) = \mathfrak{Sol}(P) \subseteq E$. Since we substitute elements of $\overline{F}$ algebraically for the differential indeterminates, Remark (2.2), which guarantees the continuation of solutions from lower ranking variables to higher ranking ones, also holds here.

Any differential F-algebra R with a differential embedding of $E \hookrightarrow R$ might be chosen as universal set of solutions, for example a universal differential field containing F: Clearly $\overline{F}[[z_1, \dots, z_n]]$ embeds into its field of quotients $\overline{F}((z_1, \dots, z_n))$, and thus $\overline{F}[[z_1, \dots, z_n]]$ also embeds into a universal differential field containing F, since $\overline{F}((z_1, \dots, z_n))$ is a finitely generated differential field extension of $\overline{F}$ (cf. [Kol73, §II.2 and §III.7]). We denote the set of solutions in R by $\mathfrak{Sol}_R(P) \subseteq R$.

A finite set of equations and inequations is called a **(differential) system** over $F\{U\}$. We will be using the same notation for systems as in the algebraic THOMAS decomposition introduced in §2.1 and §2.2, in particular a system S is represented by a pair (S_T, S_Q). However, the candidate simple system S_T will also reflect a differential structure using combinatorial methods. We will elaborate on the combinatorics in the next section.

3.2 The Combinatorics of Janet Division

In this subsection we will focus on the combinatorics of equations, enabling us to control the infinite set of differential variables appearing as partial derivatives of differential indeterminates. For this purpose we use JANET division [GB98a] which defines these combinatorics and also automatizes construction of integrability conditions. An overview of modern development can be found in [Ger05, Sei10] and the original ideas by JANET are formulated in [Jan29]. This is achieved by partitioning the set of differential variables into finitely many "cones" and "free" variables. For creating this partition we present an algorithm for inserting new equations into an existing set of equations and adjusting the cone decomposition. Apart from this insertion algorithm the only other adaptation of the algebraic Decompose algorithm (2.18) will be the reduction algorithm presented in §3.3.

We fix a (differential) **ranking** $<$, which is defined as a total order on the differential variables such that $u^{(k)} < \partial_j u^{(k)}$ and $u^{(k)} < u^{(l)}$ implies $\partial_j u^{(k)} < \partial_j u^{(l)}$ for all $u^{(k)}, u^{(l)} \in U$, $\partial_j \in \Delta$. For any finite set of differential variables, a differential ranking is a ranking as defined for the algebraic case in §2.1. This allows us to define the largest differential variable $\mathrm{ld}(p)$ appearing in a differential polynomial $p \in F\{U\}$ as **leader**, which is set to 1 for $p \in F$. Furthermore, define $\mathrm{rank}(p)$ and $\mathrm{init}(p)$ as the degree in the leader and the coefficient of $\mathrm{ld}(p)^{\mathrm{rank}(p)}$, respectively. Again we will assume $1 < u_{\mathbf{i}}^{(j)}$ for all $j \in \{1, \dots, m\}$ and $\mathbf{i} \in \mathbb{Z}^n_{\geq 0}$.

A set W of differential variables is **closed** under the action of $\Delta' \subseteq \Delta$ if $\partial_i w \in W \ \forall \partial_i \in \Delta', w \in W$. The smallest such closed set containing a differential variable w denoted by $\langle w \rangle_{\Delta'}$ is called a **cone** and the elements of Δ' we shall call (Janet) **admissible derivations**[2]. The Δ'-closed set generated by a set W of differential variables is defined to be

$$\langle W\rangle_{\Delta'} := \bigcap_{\substack{W_i \supseteq W \\ W_i\ \Delta'\text{-closed}}} W_i \;\subseteq\; \langle U\rangle_\Delta \ .$$

For a finite set $W = \{w_1, \dots, w_r\}$, the Janet **division** algorithmically assigns admissible derivations to the elements of W such that the cones generated by the $w \in W$ are disjoint. The derivation $\partial_l \in \Delta$ is assigned to the cone generated by $w = u_{\mathbf{i}}^{(j)} \in W$ as admissible derivation, if and only if

$$\mathbf{i}_l = \max\left\{\mathbf{i}'_l \mid u_{\mathbf{i}'}^{(j)} \in W, \mathbf{i}'_k = \mathbf{i}_k \text{ for all } 1 \le k < l\right\}$$

holds. We remark, that j is fixed in this definition, i.e., when constructing cones we only take into account other differential variables belonging to the same differential indeterminate. The admissible derivations assigned to w are denoted by $\Delta_W(w) \subseteq \Delta$ and we call the cone $\langle w\rangle_{\Delta_W(w)}$ the Janet **cone** of w with respect to W. This construction ensures disjointness of cones but not necessarily that the union of cones equals $\langle W\rangle_\Delta$. For the Janet **completion** a finite set $\widetilde{W} \supset W$ is successively created by adding any $\tilde{w} = \partial_i w_j \notin \biguplus_{w\in\widetilde{W}} \langle w\rangle_{\Delta_{\widetilde{W}}(w)}$ to $\widetilde{W}$, where $w_j \in \widetilde{W}$ and $\partial_i \in \Delta \setminus \Delta_{\widetilde{W}}(w_j)$. This leads to the disjoint Janet **decomposition**

$$\langle W\rangle_\Delta = \biguplus_{w\in\widetilde{W}} \langle w\rangle_{\Delta_{\widetilde{W}}(w)}$$

that separates a Δ-closed set W into finitely many cones $\langle w\rangle_{\Delta_{\widetilde{W}}(w)}$ after finitely many steps. For details see [Ger05, Def. 3.4] and [GB98a, Corr. 4.11].

With the Janet decomposition being defined for sets of differential variables, we will assign admissible derivations to differential polynomials according to their leaders. In particular, we extend the definitions of $\Delta_W(w)$ for finite $W \subset F\{U\}$ and $w \in W$ by defining $\Delta_W(w) := \Delta_{\mathrm{ld}(W)}(\mathrm{ld}(w))$.

A differential polynomial $q \in F\{U\}$ is called **reducible** with respect to $p \in F\{U\}$, if there exists $\mathbf{i} \in \mathbb{Z}^n_{\ge 0}$ such that $\partial_1^{\mathbf{i}_1} \cdot \ldots \cdot \partial_n^{\mathbf{i}_n} \mathrm{ld}(p) = \mathrm{ld}(\partial_1^{\mathbf{i}_1} \cdot \ldots \cdot \partial_n^{\mathbf{i}_n} p) = \mathrm{ld}(q)$ and $\mathrm{rank}(\partial_1^{\mathbf{i}_1} \cdot \ldots \cdot \partial_n^{\mathbf{i}_n} p) \le \mathrm{rank}(q)$. We call a derivative of an equation by an admissible derivation an **admissible prolongation**. When restricting ourselves to admissible prolongation, we get the following concept: For a finite set $T \subset F\{U\}$ we call a differential polynomial $q \in F\{U\}$ Janet **reducible** with respect to $p \in T$, if there exists $\mathbf{i} \in \mathbb{Z}^n_{\ge 0}$ such that $\partial_1^{\mathbf{i}_1} \cdot \ldots \cdot \partial_n^{\mathbf{i}_n} \mathrm{ld}(p) = \mathrm{ld}(q)$ with all proper derivatives being admissible and $\mathrm{rank}(\partial_1^{\mathbf{i}_1} \cdot \ldots \cdot \partial_n^{\mathbf{i}_n} p) \le \mathrm{rank}(q)$ holds. We shall also say that q is Janet **reducible** modulo T if there is a $p \in T$, such that q is Janet reducible with respect to $p \in T$.

A set of differential variables $T \subset \langle U\rangle_\Delta$ is called **minimal**, if for any set $S \subset \langle U\rangle_\Delta$ with $\biguplus_{t\in T} \langle t\rangle_{\Delta_T(t)} = \biguplus_{s\in S} \langle s\rangle_{\Delta_S(s)}$ the condition $T \subseteq S$ holds (cf.

[2] In [Ger99] and [Sei10, Chap. 7] the admissible derivations are called (Janet) multiplicative variables.

[GB98b, Def. 4.2]). We also call a set of differential polynomials minimal, if the corresponding set of leaders is minimal.

In addition to the non-zero initials and square-freeness of polynomials in the candidate set S_T for a simple system (as defined in §2.2), the equations in $(S_T)^=$ are required to have admissible derivations assigned to them. When an equation p is not reducible modulo $(S_T)^=$ it is added to $(S_T)^=$ and all polynomials in S_T with a leader being a derivative of $\operatorname{ld}(p)$ are removed from S_T, ensuring minimality. Furthermore, all non-admissible prolongations are created to be processed. This is formulated in the following algorithm:

Algorithm 3.1 (InsertEquation).
Input: A system S' and a polynomial $p_= \in F\{U\}$ not reducible modulo $(S'_T)^=$.
Output: A system S, where $(S_T)^= \subseteq (S'_T)^= \cup \{p_=\}$ is maximal satisfying

$$\{\operatorname{ld}(q) \mid q \in (S_T) \setminus \{p\}\} \cap \langle \operatorname{ld}(p)\rangle_\Delta = \emptyset,$$

$$S_Q = S'_Q \cup (S'_T \setminus S_T) \cup \{(\partial_i q)_= \mid q \in (S_T)^=, \partial_i \notin \Delta_{((S_T)^=)}(q)\} \ .$$

Algorithm:

```
1: $S \leftarrow S'$
2: $S_T \leftarrow S_T \cup \{p_=\}$
3: for $q \in S_T \setminus \{p\}$ do
4:    if $\operatorname{ld}(q) \in \langle \operatorname{ld}(p)\rangle_\Delta$ then
5:       $S_Q \leftarrow S_Q \cup \{q\}$
6:       $S_T \leftarrow S_T \setminus \{q\}$
7:    end if
8: end for
9: Reassign admissible derivations to $(S_T)^=$
10: $S_Q \leftarrow S_Q \cup \{(\partial_i q)_= \mid q \in (S_T)^=, \partial_i \notin \Delta_{((S_T)^=)}(q)\}$
11: return $S$
```

We remark that a non-admissible prolongations might be added to S_Q again each step, even though it has been added before. This can be prevented by simply storing all previously generated non-admissible prolongations.

3.3 Differential Simple Systems

This section goes on reducing the differential decomposition algorithm to the algebraic one. We start by introducing partial solutions in order to algebraically evaluate differential polynomials at them yielding univariate differential polynomials. Then we present a differential reduction algorithm, as the second distinction from the algebraic decomposition algorithm. At last this section defines differential simple systems.

For a differential variable $x \in \langle U\rangle_\Delta$ and a power series $e \in E$ define the F-algebra homomorphism

$$\phi_{<x,e} : F\{U\} \to \overline{F}[\ v \mid v \in \langle U\rangle_\Delta, v \geq x\ ] : \begin{cases} u_{\mathbf{i}}^{(j)} \mapsto \alpha(e)(u^{(j)}), & \text{for } u_{\mathbf{i}}^{(j)} < x \\ u_{\mathbf{i}}^{(j)} \mapsto u_{\mathbf{i}}^{(j)}, & \text{for } u_{\mathbf{i}}^{(j)} \geq x \end{cases}$$

evaluating all differential variables of a differential polynomial at e which are $<$-smaller than x.

For differential reduction the JANET partition of differential variables provides the mechanism to get a unique reductor in a fast way (for an algorithm see [GYB01]) by restricting to admissible prolongations. After finding a reductor we apply a pseudo remainder algorithm (see Eq. (1)).

We need to ensure that initials (and initials of the derivatives) of equations are non-zero. Let $p \in F\{U\}$ with $x = \mathrm{ld}(p)$ and define the **separant** $\mathrm{sep}(p) := \frac{\partial p}{\partial x}$. One easily checks (cf. [Kol73, §I.8, lemma 5] or [Hub03b, §3.1]) that the initial of any derivative of p is $\mathrm{sep}(p)$ and the separant of any square-free equation p is nonzero on $\mathfrak{Sol}(p)$. So by making the equations square-free, it is ensured that pseudo reductions are not only possible modulo p, but also modulo its derivatives. This allows us to formulate the differential reduction algorithm:

Algorithm 3.2 (Reduce).
Input: A differential system S and a polynomial $p \in F\{U\}$.
Output: A polynomial q that is not JANET reducible modulo S_T with $\phi_e(p) = 0$ if and only if $\phi_e(q) = 0$ for each $e \in \mathfrak{Sol}(S)$.
Algorithm:

```
 1: $x \leftarrow \mathrm{ld}(p)$
 2: **while** exists $q_= \in (S_T)^=$ and $i_1, \ldots, i_n \in \mathbb{Z}_{\geq 0}$ with $i_j = 0$ for $\partial_j \notin \Delta_{(S_T)^=}(q)$
    such that $\partial_1^{i_1} \cdot \ldots \cdot \partial_n^{i_n} \mathrm{ld}(q) = \mathrm{ld}(p)$ and $\mathrm{rank}(\partial_1^{i_1} \cdot \ldots \cdot \partial_n^{i_n} p) \geq \mathrm{rank}(q)$ hold
    **do**
 3:   $p \leftarrow \mathsf{prem}(p, \partial_1^{i_1} \cdot \ldots \cdot \partial_n^{i_n} q, x)$
 4:   $x \leftarrow \mathrm{ld}(p)$
 5: **end while**
 6: **if** $\mathsf{Reduce}(S, \mathrm{init}(p)) = 0$ **then**
 7:   **return** $\mathsf{Reduce}(S, p - \mathrm{init}(p) x^{\mathrm{rank}(p)})$
 8: **else**
 9:   **return** $p$
10: **end if**
```

A polynomial $p \in F\{U\}$ is called **reduced**[3] **modulo** S_T if $\mathsf{Reduce}(S, p) = p$. A polynomial $p \in F\{U\}$ **reduces to** q **modulo** S_T if $\mathsf{Reduce}(S, p) = q$.

Usually in differential algebra, one distinguishes a (full) differential reduction as used here and a partial (differential) reduction. Partial reduction only employs *proper* derivations of equations for reduction (cf. [Kol73, §I.9] or [Hub03b, §3.2]). This is useful for separation of differential and algebraic parts of the algorithm and for the use of ROSENFELD's Lemma (cf. [Ros59]).

Definition 3.3 (Differential Simple Systems). A differential system S is (JANET) **involutive**, if all non-admissible prolongations in $(S_T)^=$ reduce to zero by $(S_T)^=$.

[3] There is a fine difference between not being reducible and being reduced. In the case of not being reducible the initial of a polynomial can still reduce to zero and iteratively the entire polynomial.

A system S is called **differentially simple** or **simple**, if S is

a) algebraically simple in the finite set of differential variables appearing in it,
b) involutive,
c) $S^{=}$ is minimal,
d) no inequation is reducible modulo $S^{=}$.

A disjoint decomposition of a system into differentially simple subsystems is called **(differential)** THOMAS **decomposition**.

3.4 The Differential Decomposition Algorithm

The differential THOMAS decomposition algorithm is a modification of the algebraic THOMAS decomposition algorithm. We have already introduced the new algorithms InsertEquation (3.1) for adding new equations into the systems and Reduce (3.2) for reduction, that can replace their counterparts in the algebraic algorithm.

Algorithm 3.4 (DifferentialDecompose).
Input: A differential system S' with $(S')_T = \emptyset$.
Output: A differential THOMAS decomposition of S'.
Algorithm: The algorithm is obtained by replacing the two subalgorithms InsertEquation and Reduce in (2.18) with their differential counterparts (3.1) and (3.2), respectively.

We give an example taken from [BC99, pp. 597-600]:

Example 3.5 (Cole-Hopf Transformation). For $F := \mathbb{R}(x,t)$, $\Delta = \{\frac{\partial}{\partial x}, \frac{\partial}{\partial t}\}$, and $U = \{\eta, \zeta\}$ consider the heat equation $h = \eta_t + \eta_{xx} \in F\{U\}^{=}$ and BURGER's equation $b = \zeta_t + \zeta_{xx} + 2\zeta_x \cdot \zeta \in F\{U\}^{=}$. To improve readability, leaders of polynomials are underlined below.

First we claim that any power series solution for the heat equation with a non-zero constant term can be transformed to a solution of BURGER's equation by means of the COLE-HOPF transformation $\lambda : \eta \mapsto \frac{\eta_x}{\eta}$. The differential THOMAS decomposition for an orderly ranking with $\zeta_x > \eta_t$ of

$$\{h_{=}, \underbrace{(\eta \cdot \zeta - \eta_x)_{=}}_{\Leftrightarrow \zeta = \lambda(\eta)}, \eta_{\neq}\}$$

consists of the single system

$$S = \{(\underline{\eta_x} - \eta \cdot \zeta)_{=}, (\eta \cdot \underline{\zeta_x} + \eta_t + \eta \cdot \zeta^2)_{=}, \underline{\eta}_{\neq}\}$$

and one checks that $\mathsf{Reduce}(S, b) = 0$ holds. This implies that any non-zero solution of the heat equation is mapped by the COLE-HOPF transformation to a solution of BURGER's equation.

In addition we claim that λ is surjective. For the proof we choose an elimination ranking (cf. [Hub03b, §8.1] or [Bou07]) with $\eta \gg \zeta$, i.e., $\eta_{\mathbf{i}} > \zeta_{\mathbf{j}}$ for all

$\mathbf{i}, \mathbf{j} \in (\mathbb{Z}_{\geq 0})^2$. We compute the differential THOMAS decomposition of $\{h_=, b_=, (\eta \cdot \zeta - \eta_x)_=, \eta_{\neq}\}$ which again consists of a single system

$$S = \{(\underline{\eta_x} - \eta \cdot \zeta)_=, (\eta \cdot \zeta_x + \underline{\eta_t} + \eta \cdot \zeta^2)_=, \underline{b}_=, \underline{\zeta}_{\neq}\} \ .$$

The properties of a simple system ensure that for any solution of lower ranking equations there exists a solution of the other equations (cf. (2.2)). The elimination ordering guarantees that the only constraint for ζ is BURGER's equation $b_=$ and thus for any solution $f \in \mathfrak{Sol}(b_=)$ there exists a solution $(g, f) \in \mathfrak{Sol}(S)$. Furthermore, since $h_=$ was added to the input system, $g \in \mathfrak{Sol}(h_=)$ holds and finally the equation $(\eta \cdot \zeta - \eta_x)_=$ implies $\lambda(g) = f$.

Remark 3.6. Elements of the differential field are not subjected to splittings, unless they are modelled as differential indeterminates. For example to model a differential field $F = \mathbb{C}(x)$ with $\Delta = \{\frac{\partial}{\partial x}\}$, we add an extra differential indeterminate X to U and replace x by X in all equations and inequations. We subject X to the relation $\frac{\partial}{\partial x} X = 1$ for X being "generic" or $(\frac{\partial}{\partial x} X - 1) \cdot \frac{\partial}{\partial x} X = 0$, if we allow X to degenerate to a point. This will be subject of further study.

4 Implementation

4.1 Algorithmic Optimizations

In the Decompose algorithm, pseudo remainder sequences for the same pairs of polynomials are usually needed several times. As these calculations are expensive in general, for *avoiding repeated calculations*, it is important that the results are kept in memory and will be reused when the same sequence is requested again.

If a polynomial admits *factorization*, we can use the it to save computation time. More precisely, a disjoint decomposition of the system $S \uplus \{(p \cdot q)_=\}$ is given by $(S \cup \{p_=\}, S \cup \{p_{\neq}, q_=\})$ and the system $S \uplus \{(p \cdot q)_{\neq}\}$ is equivalent to $S \cup \{p_{\neq}, q_{\neq}\}$. Let $Y_i := \left\{x_j \mid x_j < x_i, (S_T)^{=}_{x_j} \neq \emptyset\right\}$ and $Z_i := \left\{x_j \mid x_j < x_i, (S_T)^{=}_{x_j} = \emptyset\right\}$. If $(S_T)^{=}_{x_i}$ is irreducible over the field $F_i := F(Z_i)[Y_i]/\langle (S_T)^{=}_{<x_i} \rangle_{F(Z_i)[Y_i]}$ for all $i \in \{1, \ldots, n\}$, where $\langle (S_T)^{=}_{<x_i} \rangle_{F(Z_i)[Y_i]}$ is the ideal generated by $(S_T)^{=}_{<x_i}$ in the polynomial ring $F(Z_i)[Y_i]$, factorization of polynomials can be performed over F_n instead of F.

Coefficient growth is a common problem in elimination. If possible, polynomials should be represented as compact as possible. Once it is known that the initial cannot vanish, the *content* (in the univariate sense) cannot vanish either. Thus, every time an initial has been added as an inequation to the system, one can divide the polynomial by its content.

If the ground field F is represented as a field of fractions of a domain D (like the rationals or a rational function field over the rationals), it also makes sense to remove the multivariate content, which is an element of F.

When reducing, in addition to reduction modulo the polynomial of the same leader, reducing the coefficients modulo the polynomials of lower leader can be considered. In some cases this leads to a reduction of sizes of coefficients, in other cases sizes increase. The latter is partly due to whole polynomials being multiplied with initials of the reductors. Finding a good heuristic for this *coefficient reduction* is crucial for efficiency.

In the algebraic algorithm, polynomials don't necessarily have to be square-free when they are inserted into the candidate simple system. Efficiency is sometimes improved greatly by postponing the calculation of the square-free split as long as possible.

In the differential case application of *criteria* to avoid useless reduction of non-admissible prolongations can decrease computation time. The combinatorial approach used in this paper already avoids many reductions of so-called Δ-polynomials, as used in other approaches (see [GY06]). Nonetheless, using the involutive criteria 2-4 (cf. [GB98a, Ger05, AH05] and [BLOP09, §4, Prop. 5]) which together are equivalent to the chain criterion, is valid and helpful.

Another possible improvement is parallelization, since the main loop in line 2 of Decompose (2.18) can naturally be used in parallel for different systems.

4.2 Implementation in Maple

Both algorithms have been implemented in the Maple computer algebra system. Packages can be downloaded from [BLH10], documentation and example worksheets are available there.

The main reason for choosing Maple for the implementation is the collection of solvers for polynomial equations, ODEs, and PDEs already present. Furthermore, fast algorithms exist for polynomial factorization over finitely generated field extensions of $\mathbb{Q}$ and for gcd computation. Computation of subresultants is not available in Maple, therefore an algorithm based on [Duc00] is implemented for that purpose.

Features for the differential package include arbitrary differential rankings, using functions implemented in Maple as differential field, computation of power series solutions, and a direct connection to the solvers of Maple for differential equations.

Example 4.1. Start by loading the current version of our package:

```
> with(DifferentialThomas):
> ComputeRanking([t],[x2,x1,y,u],"EliminateFunction");
```

This creates the differential polynomial ring $\mathbb{Q}\{x^{(2)}, x^{(1)}, y, u\}$ for $\Delta = \{\frac{\partial}{\partial t}\}$. Here u indicates the input, $x^{(1)}$ and $x^{(2)}$ the state, and y the output of the system. The chosen ranking "$<$" is the elimination ranking with $x^{(2)} \gg x^{(1)} \gg y \gg u$, i.e., $x^{(2)}_{\mathbf{i}} > x^{(1)}_{\mathbf{j}} > y_{\mathbf{k}} > u_{\mathbf{l}}$ for all $\mathbf{i}, \mathbf{j}, \mathbf{k}, \mathbf{l} \in \mathbb{Z}_{\geq 0}$.

```
> L:=[x1[1]-u[0]*x2[0],x2[1]-x1[0]-u[0]*x2[0],y[0]-x1[0]];
```

$$L := [x1_1 - u_0\, x2_0,\; x2_1 - x1_0 - u_0\, x2_0,\; y_0 - x1_0]$$

We follow [Dio92, Ex. 1] and want to compute the external trajectories of a differential ideal generated by L, i.e. intersect this differential ideal with $\mathbb{Q}\{y, u\}$.

```
> res:=DifferentialThomasDecomposition(L,[]);
```

$$res := [DifferentialSystem, DifferentialSystem]$$

We show the equations and inequations of the differential systems not involving $x^{(1)}$ and not involving $x^{(2)}$. The chosen ranking guarantees that for each differential system of the output, all constraints holding for lower ranking differential indeterminates can be read off the equations and inequations only involving these differential indeterminates, i.e., the systems shown determine the external trajectories of the system:

```
> PrettyPrintDifferentialSystem(res[1]):
> remove(a->has(a,x2) or has(a,x1),%);
```

$$[-\mathrm{u}(t)\,(\tfrac{d^2}{dt^2}\,\mathrm{y}(t)) + (\tfrac{d}{dt}\,\mathrm{y}(t))\,\mathrm{u}(t)^2 + (\tfrac{d}{dt}\,\mathrm{y}(t))\,(\tfrac{d}{dt}\,\mathrm{u}(t)) + \mathrm{y}(t)\,\mathrm{u}(t)^2 = 0,\ \mathrm{u}(t) \neq 0]$$

```
> PrettyPrintDifferentialSystem(res[2]):
> remove(a->has(a,x2) or has(a,x1),%);
```

$$[\tfrac{d}{dt}\,\mathrm{y}(t) = 0,\ \mathrm{u}(t) = 0]$$

These systems, having disjoint solution sets, are identical to the ones found in [Dio92].

4.3 Implementations of Similar Decomposition Algorithms

The RegularChains package [LMX05], which is shipped with recent versions of Maple, implements a decomposition of a polynomial ideal into ideals represented by regular chains and a radical decomposition of an ideal into square-free regular chains. The solution sets of this decomposition are in general not disjoint. However, there is an extension called comprehensive triangular decomposition (cf. [CGL+07]) that provides disjointness on the parameters of a parametric system. The systems of the parameters are not simple systems though. The RegularChains package contains FastArithmeticTools as a subpackage implementing asymptotically fast polynomial arithmetic for the modular case.

The ϵpsilon package ([Wan03]) by Dongming Wang implements different kinds of triangular decompositions (including a decomposition into regular chains like above) in Maple. It is the only software package besides our own that implements the Thomas decomposition. It uses the simpler "top-down" approach that Thomas (cf. [Tho37, Tho62]) suggested, i.e., polynomials of higher leader are considered first. All polynomials of the same leader are combined into one common consequence. New systems, which contain conditions on initials of polynomials and subresultants, are created by splitting subalgorithms similar to ours. All these new conditions of lower leader are not taken into account for now and will be treated in a later step. Contrary to our approach, one cannot reduce modulo an *unfinished* system and hence inconsistency checks are less natural and more complicated. It is conceivable that this strategy spends too much time on computations with inconsistent systems. Therefore, ϵpsilon implements highly sophisticated heuristics for early detection of inconsistent systems. It achieves similar performance to our implementation.

The MAPLE package diffalg [BH04] deals with ordinary and partial differential equations as described in [BLOP09]. Its functionalities are used by symbolic differential equations solvers in MAPLE. For an input of equations and inequations it computes a radical decomposition of the differential ideal generated by the equations and saturated by the inequations. I.e., a description of the vanishing ideal of the KOLCHIN closure (cf. [Kol73, §IV.1]) of the solutions is computed. The output are differential characteristic sets as introduced by RITT [Rit50, §I.5]. Computation of differential consequences is driven by reduction of Δ-polynomials, which are the analogon of s-polynomials in differential algebra. We found the system being optimized and well-suited for computations with ordinary differential equations.

Similar algorithms as in diffalg are used in the BLAD-libraries [Bou09]. It is designed as a stand-alone C-library with an emphasis on usability for non-mathematicians and extensive documentation. As it is written in C, BLAD is expected to outperform diffalg for relevant examples.

For future publications, we plan to compare the THOMAS decomposition and our implementation with other decompositions and implementations. We also plan to further examine applications that benefit from the properties of simple systems.

Acknowledgements

The contents of this paper profited very much from numerous useful comments and remarks from Wilhelm Plesken. The authors thank him as well as Dongming Wang and François Boulier for fruitful discussions. Furthermore, our gratitude goes to the anonymous referees for valuable comments and for pointing out informative references. The second author (V.P.G.) acknowledges the Deutsche Forschungsgemeinschaft for the financial support that made his stay in Aachen possible. The presented results were obtained during his visits.

References

[AH05] Apel, J., Hemmecke, R.: Detecting unnecessary reductions in an involutive basis computation. J. Symbolic Comput. 40(4-5), 1131–1149 (2005), MR MR2169107 (2006j:13026)

[BC99] Buium, A., Cassidy, P.J.: Differential algebraic geometry and differential algebraic groups: from algebraic differential equations to diophantine geometry. In: [Kol99], pp. 567–636 (1999)

[BCG$^+$03] Blinkov, Y.A., Cid, C.F., Gerdt, V.P., Plesken, W., Robertz, D.: The MAPLE Package JANET: I. Polynomial Systems. II. Linear Partial Differential Equations. In: Proc. 6th Int. Workshop on Computer Algebra in Scientific Computing, Passau, Germany, pp. 31–54 (2003), http://wwwb.math.rwth-aachen.de/Janet

[BH04] Boulier, F., Hubert, E.: *DIFFALG*: description, help pages and examples of use, Symbolic Computation Group, University of Waterloo, Ontario, Canada (1996-2004), http://www-sop.inria.fr/members/Evelyne.Hubert/diffalg/

[BKRM01] Bouziane, D., Rody, A.K., Maârouf, H.: Unmixed-dimensional decomposition of a finitely generated perfect differential ideal. J. Symbolic Comput. 31(6), 631–649 (2001), MR MR1834002 (2002c:12007)

[BLH10] Bächler, T., Lange-Hegermann, M.: *AlgebraicThomas* and *DifferentialThomas*: Thomas decomposition for algebraic and differential systems (2008-2010),
`http://wwwb.math.rwth-aachen.de/thomasdecomposition/`

[BLOP95] Boulier, F., Lazard, D., Ollivier, F., Petitot, M.: Representation for the radical of a finitely generated differential ideal. In: ISSAC, pp. 158–166 (1995)

[BLOP09] Boulier, F., Lazard, D., Ollivier, F., Petitot, M.: Computing representations for radicals of finitely generated differential ideals. Appl. Algebra Engrg. Comm. Comput. 20(1), 73–121 (2009), MR MR2496662 (2010c:12005)

[Bou07] Boulier, F.: Differential elimination and biological modelling, Gröbner bases in symbolic analysis. Radon Ser. Comput. Appl. Math. 2, 109–137 (2007), MR MR2394771 (2009f:12005)

[Bou09] Boulier, F.: *BLAD*: Bibliothèques lilloises d'algèbre différentielle (2004-2009),
`http://www.lifl.fr/~boulier/BLAD/`

[CGL$^+$07] Chen, C., Golubitsky, O., Lemaire, F., Maza, M.M., Pan, W.: Comprehensive triangular decomposition. In: Ganzha, V.G., Mayr, E.W., Vorozhtsov, E.V. (eds.) CASC 2007. LNCS, vol. 4770, pp. 73–101. Springer, Heidelberg (2007)

[Del00] Dellière, S.: D.m. wang simple systems and dynamic constructible closure, Rapport de Recherche No. 2000–16 de l'Université de Limoges (2000)

[Dio92] Diop, S.: On universal observability. In: Proc. 31st Conference on Decision and Control, Tucaon, Arizona (1992)

[Duc00] Ducos, L.: Optimizations of the subresultant algorithm. J. Pure Appl. Algebra 145(2), 149–163 (2000), MR MR1733249 (2000m:68187)

[GB98a] Gerdt, V.P., Blinkov, Y.A.: Involutive bases of polynomial ideals. Math. Comput. Simulation 45(5-6), 519–541 (1998), Simplification of systems of algebraic and differential equations with applications. MR MR1627129 (99e:13033)

[GB98b] Gerdt, V.P., Blinkov, Y.A.: Minimal involutive bases. Math. Comput. Simulation 45(5-6), 543–560 (1998), Simplification of systems of algebraic and differential equations with applications. MR MR1627130 (99e:13034)

[Ger99] Gerdt, V.P.: Completion of linear differential systems to involution. In: Computer Algebra in Scientific Computing—CASC 1999, Munich, pp. 115–137. Springer, Berlin (1999), MR MR1729618 (2001d:12010)

[Ger05] Gerdt, V.P.: Involutive algorithms for computing Gröbner bases. In: Computational Commutative and Non-Commutative Algebraic Geometry, NATO Sci. Ser. III Comput. Syst. Sci., vol. 196, pp. 199–225. IOS, Amsterdam (2005), MR MR2179201 (2007c:13040)

[Ger08] Gerdt, V.P.: On decomposition of algebraic PDE systems into simple subsystems. Acta Appl. Math. 101(1-3), 39–51 (2008), MR MR2383543 (2009c:35003)

[GY06] Gerdt, V.P., Yanovich, D.A.: Investigation of the effectiveness of involutive criteria for computing polynomial Janet bases. Programming and Computer Software 32(3), 134–138 (2006), MR MR2267374 (2007e:13036)

[GYB01] Gerdt, V.P., Yanovich, D.A., Blinkov, Y.A.: Fast search for the Janet divisor. Programming and Computer Software 27(1), 22–24 (2001), MR MR1867717

[Hab48] Habicht, W.: Eine Verallgemeinerung des Sturmschen Wurzelzählverfahrens. Comment. Math. Helv. 21, 99–116 (1948), MR MR0023796 (9,405f)

[Hub03a] Hubert, E.: Notes on triangular sets and triangulation-decomposition algorithms. I. Polynomial systems. In: Winkler, F., Langer, U. (eds.) SNSC 2001. LNCS, vol. 2630, pp. 1–39. Springer, Heidelberg (2003), MR MR2043699 (2005c:13034)

[Hub03b] Hubert, E.: Notes on triangular sets and triangulation-decomposition algorithms. II. Differential systems. In: Winkler, F., Langer, U. (eds.) SNSC 2001. LNCS, vol. 2630, pp. 40–87. Springer, Heidelberg (2003), MR MR2043700 (2005c:13035)

[Jan29] Janet, M.: Leçons sur les systèmes des équationes aux dérivées partielles. In: Cahiers Scientifiques IV, Gauthiers-Villars, Paris (1929)

[Kol73] Kolchin, E.R.: Differential algebra and algebraic groups. Pure and Applied Mathematics, vol. 54. Academic Press, New York (1973), MR MR0568864 (58 #27929)

[Kol99] Kolchin, E.R.: Selected works of Ellis Kolchin with commentary. American Mathematical Society, Providence (1999); Commentaries by Borel, A., Singer, M.F., Poizat, B., Buium, A., Cassidy, P.J. (eds.) with a preface by Hyman Bass, Buium and Cassidy. MR MR1677530 (2000g:01042)

[LMX05] Lemaire, F., Moreno Maza, M., Xie, Y.: The RegularChains library in Maple. SIGSAM Bull. 39(3), 96–97 (2005)

[LW99] Li, Z., Wang, D.: Coherent, regular and simple systems in zero decompositions of partial differential systems. System Science and Mathematical Sciences 12, 43–60 (1999)

[Mis93] Mishra, B.: Algorithmic algebra. Texts and Monographs in Computer Science. Springer, New York (1993), MR MR1239443 (94j:68127)

[Riq10] Riquier, F.: Les systèmes d'équations aux dérivées partielles (1910)

[Rit50] Ritt, J.F.: Differential Algebra. In: American Mathematical Society Colloquium Publications, vol. XXXIII. American Mathematical Society, New York (1950), MR MR0035763 (12,7c)

[Ros59] Rosenfeld, A.: Specializations in differential algebra. Trans. Amer. Math. Soc. 90, 394–407 (1959), MR MR0107642 (21 #6367)

[Sei10] Seiler, W.M.: Involution. In: Algorithms and Computation in Mathematics, vol. 24. Springer, Berlin (2010), The formal theory of differential equations and its applications in computer algebra. MR MR2573958

[sGH09] shan Gao, X., Huang, Z.: Efficient characteristic set algorithms for equation solving in finite fields and application in analysis of stream ciphers, Cryptology ePrint Archive, Report 2009/637 (2009), http://eprint.iacr.org/

[Tho37] Thomas, J.M.: Differential systems, vol. XXI. AMS Colloquium Publications (1937)

[Tho62] Thomas, J.M.: Systems and roots. The William Byrd Press, Inc., Richmond Virginia (1962)

[Wan98] Wang, D.: Decomposing polynomial systems into simple systems. J. Symbolic Comput. 25(3), 295–314 (1998), MR MR1615318 (99d:68130)

[Wan01] Wang, D.: Elimination methods. In: Texts and Monographs in Symbolic Computation. Springer, Vienna (2001), MR MR1826878 (2002i:13040)

[Wan03] Wang, D.: *ϵpsilon*: description, help pages and examples of use (2003), `http://www-spiral.lip6.fr/~wang/epsilon/`

[Wan04] Wang, D.: Elimination practice. Imperial College Press, London (2004), Software tools and applications, With 1 CD-ROM (UNIX/LINUX, Windows). MR MR2050441 (2005a:68001)

[Wu00] Wu, W.-T.: Mathematics mechanization. In: Mathematics and its Applications, vol. 489, Kluwer Academic Publishers Group, Dordrecht (2000), Mechanical geometry theorem-proving, mechanical geometry problem-solving and polynomial equations-solving. MR MR1834540 (2003a:01005)

[Yap00] Yap, C.K.: Fundamental problems of algorithmic algebra. Oxford University Press, New York (2000), MR MR1740761 (2000m:12014)

On Multivariate Homogeneous Polynomial Decomposition

Paula Bustillo and Jaime Gutierrez

Universidad de Cantabria, Santander, Spain

Abstract. An algorithm for *decomposing* a list of homogeneous polynomials in several variables of the same degree was given in [2]. We show that there is a bijective relation among these decompositions and intermediate $\mathbb{K}$-algebras of a special kind, but the relation cannot be extended to intermediate fields. We also try to find the *dimension* of the decomposable lists over an algebraically closed field.

1 Introduction

In [9], the authors proposed a new cryptosystem called 2R-scheme inspired by the C^*-cryptosystem, see [7]. In a 2R-scheme the space of plain texts and ciphertexts is $\mathbb{F}_q^m$, where $\mathbb{F}_q$ is a finite field of q elements. The secret key items are three affine bijections $r, s, t : \mathbb{F}_q^m \longrightarrow \mathbb{F}_q^m$ and two applications $\phi, \psi : \mathbb{F}_q^m \longrightarrow \mathbb{F}_q^m$ given by m quadratic equations over $\mathbb{F}_q$. The public key is the polynomial representation of the application $t \circ \psi \circ s \circ \phi \circ r : \mathbb{F}_q^m \longrightarrow \mathbb{F}_q^m$. This representation consists of m polynomials of degree 4.

The applications ϕ and ψ are chosen among easily invertible ones in order to make decryption easy. For all proposed easily invertible applications at that time, the one-round schemes were broken, i.e., the analogous cryptosystems with secret key $s \circ \phi \circ r$. Therefore, the security of 2R-schemes was based on the difficulty of decomposing a list of m polynomials in $\mathbb{K}[\mathbf{x}] = \mathbb{K}[x_1, \dots, x_m]$, where $\mathbb{K}$ is an arbitrary field. The paper [10] proposed efficient attacks that make the system insecure if m or $m-1$ polynomials in the list are given. Inspired by these ideas, in [1], the authors presented an algorithm that given a list $\mathbf{f} = (f_1, \dots, f_u)$ of u homogeneous polynomials of degree 4 in m variables, finds lists $\mathbf{g} = (g_1, \dots, g_u)$ and $\mathbf{h} = (h_1, \dots, h_m)$ of homogeneous polynomials of degree 2 in m variables such that $f_i = g_i(h_1, \dots, h_m)$ for all $i \in \{1, \dots, u\}$, under some favourable circumstances. The algorithm was extended in [2] to a list of polynomials $\mathbf{f}$ of arbitrary degree $n = r \cdot s$. There is an improvement of the algorithm in [3], together with an algorithm for a list $\mathbf{f}$ of polynomials of degrees $r_1, \dots, r_u$ respectively such that $s > 1$ divides all degrees.

In [4], the dimension of the decomposable univariate polynomials over an algebraically closed field is counted for each fixed degree n, and similarly in [5], for the so called uni-multivariate decomposable polynomials, see [6], over an algebraically closed field.

We aim here at finding the relation among the concept of (r, s)-decomposition of homogeneous polynomials proposed in [1] and the computation of intermediate

$\mathbb{K}$-algebras and intermediate fields, and at counting the dimension of (r, s)-decomposable polynomials in m variables over an algebraically closed field.

2 Computation of Intermediate $\mathbb{K}$-Algebras and (r, s)-Decompositions

We shall start by the definition of (r, s)-decomposable polynomials:

Definition 1. *Let* $\mathbf{f} = (f_1, \ldots, f_u) \in \mathbb{K}[\mathbf{x}]^u$ *be a list of homogeneous polynomials of degree* $n = rs$. *We say that* $\mathbf{f}$ *is* (r, s)*-decomposable if there exist a list* $\mathbf{g} = (g_1, \ldots, g_u) \in \mathbb{K}[\mathbf{x}]^u$ *of homogeneous polynomials of degree* r *and a list* $\mathbf{h} = (h_1, \ldots, h_m) \in \mathbb{K}[\mathbf{x}]^m$ *of homogeneous polynomials of degree* s *such that* $f_i = g_i(h_1, \ldots, h_m)$*, written* $\mathbf{f} = \mathbf{g} \circ \mathbf{h}$*. The tuple* $(\mathbf{g}, \mathbf{h})$ *is called an* (r, s)*-decomposition of* $\mathbf{f}$*.*

If A is a regular matrix, then $\mathbf{g} \circ \mathbf{h} = \mathbf{g} \circ A^{-1} \circ A \circ \mathbf{h}$. To avoid this ambiguity, two decompositions $(\mathbf{g}, \mathbf{h})$ and $(\mathbf{g}', \mathbf{h}')$ of a polynomial are defined to be **equivalent** if there exists a regular matrix A such that $\mathbf{h}'^T = A\mathbf{h}^T$. By this equivalence relation, we guarantee that two non-equivalent decompositions provide two different intermediate $\mathbb{K}$-algebras.

Lemma 1. *Let* $h_1, \ldots, h_m \in \mathbb{K}[\mathbf{x}]$ *be homogeneous polynomials of degree* s *and let* $f_1, \ldots, f_u \in \mathbb{K}[h_1, \ldots, h_m]$ *be homogeneous polynomials of degree* $n = r \cdot s$*. There exist polynomials* $g_1, \ldots, g_u \in \mathbb{K}[\mathbf{x}]$ *such that* $f_i = g_i(h_1, \ldots, h_m)$ *is an* (r, s)*-decomposition of* f*.*

Proof. We can write $f_i = p_i(h_1, \ldots, h_m)$, for each $i \in \{1, \ldots, u\}$. Let $p_i = p_i^{(k_i)} + \cdots + p_i^{(1)} + p_i^{(0)}$ be written as the sum of homogeneous polynomials $p_i^{(j)}$ of degree j, $k_i = \deg p_i$. It is clear that either $\deg p_i^{(j)}(h_1, \ldots, h_m) = j \cdot s$ or $p_i^{(j)}(h_1, \ldots, h_m) = 0$. Consequently, since $f_i = p_i^{(k_i)}(h_1, \ldots, h_m) + \cdots + p_i^{(1)}(h_1, \ldots, h_m) + p_i^{(0)}$, then $k_i \geq r$ and $f_i = p_i^{(r)}(h_1, \ldots, h_m)$. Take $g_i = p_i^{(r)}$.

Theorem 1. *Non-equivalent* (r, s)*-decompositions of a list of polynomials* $\mathbf{f} = (f_1, \ldots, f_u)$ *correspond bijectively to* $\mathbb{K}$*-algebras in* $\mathbb{K}[\mathbf{f}] \subset \mathbb{K}[\mathbf{x}]$ *generated by* m *homogeneous polynomials of degree* s*.*

Proof. Let $h_1, \ldots, h_m, h'_1, \ldots, h'_m \in \mathbb{K}[x_1, \ldots, x_m]$ be homogeneous polynomials of degree s such that $\mathbb{K}[f_1, \ldots, f_u] \subset \mathbb{K}[h_1, \ldots, h_m] = \mathbb{K}[h'_1, \ldots, h'_m]$. By Lemma 1, we have the equality of linear spans: Span $(h_1, \ldots, h_m) =$ Span $(h'_1, \ldots, h'_m)$; and therefore, there exists a regular matrix A such $(h'_1, \ldots, h'_m)^T = A(h_1, \ldots, h_m)^T$.

The algorithm of Faugère and Perret only finds an (r, s)-decomposition of $\mathbf{f}$ if $\mathbf{f}$ has only one equivalence class of decompositions, i.e., it only finds a decomposition when there is exactly one intermediate $\mathbb{K}$-algebra (field) in $\mathbb{K}[\mathbf{f}] \subset \mathbb{K}[\mathbf{x}]$ (in $\mathbb{K}(\mathbf{f}) \subset \mathbb{K}(\mathbf{x})$) generated by m homogeneous polynomials of degree s.

This bijective relation does not extend to a bijective relation among the (r, s)-decompositions of $\mathbf{f}$ and the proper fields in $\mathbb{K}(\mathbf{f}) \subset \mathbb{K}(\mathbf{x})$ generated by a list $\mathbf{h}$ of homogeneous polynomials of degree s in general.

Example 1. Let

$$(h_1, h_2) = ({x_1}^{2\,S}, {x_1}^{S}{x_2}^{S}),$$
$$(H_1, H_2) = ({x_1}^{2\,S} - {x_2}^{2\,S}, {x_2}^{2\,S} + {x_1}^{S}{x_2}^{S}),$$

for $s = 2S$ even, and let

$$(h_1, h_2) = ({x_1}^{2\,S+1}, {x_1}^{S+1}{x_2}^{S}),$$
$$(H_1, H_2) = ({x_1}^{2\,S+1} - x_1{x_2}^{2\,S}, {x_1}^{S+1}{x_2}^{S} + x_1{x_2}^{2\,S}$$

for $s = 2S + 1$ odd.

For both s even and odd, it holds that $\mathbb{K}[h_1, h_2] \neq \mathbb{K}[H_1, H_2]$ because the linear spans $\mathrm{Span}(h_1, h_2)$ and $\mathrm{Span}(H_1, H_2)$ are different, and $\mathbb{K}(h_1, h_2) = \mathbb{K}(H_1, H_2)$, since

$H_1 = \frac{{h_1}^2 - {h_2}^2}{h_1}$, $H_2 = \frac{(h_1+h_2)h_2}{h_1}$, and

$h_1 = \frac{{H_1}^2 + 2\,H_1H_2 + {H_2}^2}{H_1 + 2\,H_2}$, $h_2 = \frac{(H_1+H_2)H_2}{H_1+2\,H_2}$. Moreover, $h_1^2 + h_2^2 + 2h_1h_2 = H_1^2 + H_2^2 + 2H_1H_2$.

Therefore, if

$$f = (h_1^2 + h_2^2 + 2h_1h_2)^k,$$

then

$((x_1^2 + x_2^2 + 2x_1x_2)^k, (h_1, h_2))$ and $((x_1^2 + x_2^2 + 2x_1x_2)^k, (H_1, H_2))$

are non-equivalent $(2k, s)$-decompositions of f, but their associated fields are the same.

This counterexample can be easily extended to an arbitrary number m of variables.

3 The Dimension of (r, s)-Decomposable Polynomials

From now on, $\mathbb{K}$ will denote an algebraically closed field.

Let

$$P_{m,n} = \{f \in \mathbb{K}[\mathbf{x}] : f \text{ is homogeneous of degree } n\}$$

be the vector space of homogeneous polynomials of degree n, whose dimension is $a_{m,n} = \binom{m+n-1}{n}$.

By arranging the monomials of degree n in m variables with respect to the lexicographical order $>_{lex}$, $m(1) = x_1^n, m(2) = x_1^{n-1}x_2, \ldots, m(a_{m,n}) = x_m^n$, we can identify a polynomial in $P_{m,n}$ sorted with respect to the lexicographical order with a tuple in $\mathbb{K}^{a_{m,n}}$, thus identifiying $P_{m,n}$ with the affine space $\mathbb{K}^{a_{m,n}}$.

For $n = rs$, we have the composition map

$$\begin{array}{rccc} \gamma_{m,n,r} : & P_{m,r} \times {P_{m,s}}^m & \longrightarrow & P_{m,n} \\ & (g, h_1, \ldots, h_m) & \mapsto & g(h_1, \ldots, h_m) \end{array}$$

Clearly, the set $D_{m,n,r}$ of (r, s)-decomposable polynomials of degree n is $\mathrm{Im}\,\gamma_{m,n,r}$. The map $\gamma_{m,n,r}$ can be identified with a polynomial map

$$\Gamma_{m,n,r} : \mathbb{K}^{a_{m,r}} \times (\mathbb{K}^{a_{m,s}})^m \longrightarrow \mathbb{K}^{a_{m,n}}$$

that sends the coefficients of $g, h_1, \ldots, h_m$ to the coefficients of $g(h_1, \ldots, h_m)$. This map identifies $D_{m,n,r}$ with $\mathrm{Dec}_{m,n,r} = \mathrm{Im}\,\Gamma_{m,n,r}$. We aim at finding the dimension of the Zariski closure of $\mathrm{Dec}_{m,n,r}$, $\overline{\mathrm{Dec}_{m,n,r}}$.

A straightforward way to compute the dimension is to combine a suitable normalization in (r,s)-decompositions with the following theorem:

Theorem 2. ([8]) *Let X, Y be algebraic sets over $\mathbb{K}$. If $f : X \longrightarrow Y$ is a dominating polynomial map, i.e., such that $Y = \overline{f(X)}$, then there exists an open subset U in Y such that $f^{-1}(y)$ has dimension* $\dim X - \dim Y$ *for all $y \in U$.*

As a consequence, if the map $\Gamma_{m,n,r}|_X : X \longrightarrow \overline{\mathrm{Dec}_{m,n,r}}$ is dominating and such that all polynomials in $\mathrm{Dec}_{m,n,r} \setminus C$ have a finite number of (r,s)-decompositions, for a closed set $C \subsetneq \mathrm{Dec}_{m,n,r}$, then $\dim \mathrm{Dec}_{m,n,r} = \dim X$.

It is clear that for $X = \mathbb{K}^{a_{m,r}+m \cdot a_{m,s}}$ the hypotheses are not satisfied: whenever a polynomial f has the (r,s)-decomposition $f = g \circ \mathbf{h}$, we can decompose f as $f = (g \circ A^{-1}) \circ (A \circ \mathbf{h})$ for every $A \in \mathrm{GL}_m(\mathbb{K})$.

Clearly, finding the set X for which the hypotheses are satisfied is the key point to find the dimension of $\overline{\mathrm{Dec}_{m,n,r}}$. In the following, we will discuss the choice of X.

Assume that $f = g(h_1,\ldots,h_m)$ is an (r,s)-decomposition of f where $h_1,\ldots,h_m$ are linearly independent. Then, the vector space generated by $h_1,\ldots,h_m$ is also generated by m homogeneous polynomials $h'_1,\ldots,h'_m$ of degree s such that each polynomial is monic with respect to the lexicographical order, $\mathrm{lm}(h'_1) >_{lex} \ldots >_{lex} \mathrm{lm}(h'_m)$, and $\mathrm{coeff}_{\mathrm{lm}(h'_i)}(h'_j) = 0$ for $i \neq j$, where $\mathrm{lm}(t)$ denotes the leading monomial of the polynomial t and $\mathrm{coeff}_m(t)$ is the coefficient of the monomial m in the polynomial t. Then, for $\mathbf{h}' = (h'_1,\ldots,h'_m)$, there exists an homogenous polynomial g' in m variables of degree r such that $f = g' \circ \mathbf{h}'$.

Let $V(i_1,\ldots,i_m)$ be the set of vector spaces generated by m polynomials $h_1,\ldots,h_m$, where $i_1 < i_2 < \cdots < i_m$, each h_j is monic with leading coefficient $m(i_j)$, and $\mathrm{coeff}_{\mathrm{lm}(h_j)}(h_i) = 0$ if $i \neq j$:

$$\begin{array}{cc}
 & \begin{array}{cccccccc} & i_1 & & i_2 & & & i_m & \end{array} \\
\begin{array}{c} h_1 \rightarrow \\ h_2 \rightarrow \\ \\ h_m \rightarrow \end{array} & \begin{pmatrix} 0 & 1 & \cdots & 0 & \cdots & \cdots & 0 & \cdots \\ 0 & 0 & 0 & 1 & \cdots & \cdots & 0 & \cdots \\ 0 & 0 & 0 & 0 & \cdots & \cdots & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \cdots \end{pmatrix}
\end{array}$$

Each vector space in $V(i_1,\ldots,i_m)$ can be determined by $m\cdot(a_{m,s}-m)$ coefficients in $\mathbb{K}$ at most, thus identifying $V(i_1,\ldots,i_m)$ with $\mathbb{K}^{m\cdot(a_{m,s}-m)}$.

Let $\hat{V} = \cup_{1 \le i_1 < i_2 < \ldots < i_m \le a_{m,s}} V(i_1,\ldots,i_m)$ and V be the algebraic set corresponding to $\hat{V}$ by the identification between $P_{m,s}$ and $\mathbb{K}^{a_{m,s}}$. Then, $Dec_{m,n,r} = \mathrm{Im}\ \Gamma_{m,n,r}(\mathbb{K}^{a_{m,r}} \times \hat{V})$. Clearly, $\dim \overline{\mathrm{Dec}_{m,n,r}} = \dim \overline{\mathrm{Im}\ \Gamma(\mathbb{K}^{a_{m,r}} \times V)} \le a_{m,r} + m \cdot (a_{m,s} - m)$. Therefore, if $\dim \overline{\mathrm{Im}\ \Gamma(\mathbb{K}^{a_{m,r}} \times V(1,2,\ldots,m))} = a_{m,r} + m \cdot (a_{m,s} - m)$ could be proven, then $\dim \overline{\mathrm{Dec}_{m,n,r}} = a_{m,r} + m \cdot (a_{m,s} - m)$.

This normalization is proven to be the good one for $(2,2)$-decompositions in two variables by using Gröbner basis computations.

Example 2. Let $r = s = m = 2$ and $n = 4$.
Let $V_1 = \{(x^2 + ay^2, xy + by^2) : a, b \in \mathbb{K}\}$.

Let

$$f = c_1x^4 + c_2x^3y + c_3x^2y^2 + c_4xy^3 + c_5y^4,$$
$$g = d_1x^2 + d_2xy + d_3y^2,$$
$$h_1 = x^2 + ay^2, \text{ and}$$
$$h_2 = xy + by^2.$$

If $f = g(h_1, h_2)$, then $d_1 = c_1$ and $d_2 = c_2$. Moreover, the equations

$$c_3 - 2c_1a - c_2b - d_3 = 0,$$
$$c_4 - c_2a - 2d_3b = 0 \text{ and}$$
$$c_5 - c_1a^2 - c_2ab - d_3b^2 = 0$$

must be satisfied.

The Gröbner basis G of $I = (c_3 - 2c_1a - c_2b - d_3, c_4 - c_2a - 2d_3b, c_5 - c_1a^2 - c_2ab - d_3b^2)$ with respect to the lexicographical order such that $a > b > d_3$ is the set of the polynomials

$$p_1 = {d_3}^3 - 2\,c_3{d_3}^2 + \left(c_4c_2 - 4\,c_1c_5 + {c_3}^2\right) d_3 + \left(c_1{c_4}^2 + c_5{c_2}^2 - c_3c_2c_4\right)$$

$$p_2 = \left({c_2}^3 - 4\,c_3c_2c_1 + 8\,{c_1}^2c_4\right) b + (2\,c_2c_1c_4 + 4\,{c_3}^2c_1 - 16\,{c_1}^2c_5 + \left(-8\,c_3c_1 + {c_2}^2\right) d_3 - {c_2}^2c_3 + 4\,{d_3}^2c_1) =: q_1b + q_2$$

$$p_3 = \left({c_2}^3 - 4\,c_3c_2c_1 + 8\,{c_1}^2c_4\right) a + (-c_4{c_2}^2 - 4\,c_3c_1c_4 + (4\,c_1c_4 + 2\,c_3c_2)\,d_3 + 8\,c_2c_1c_5 - 2\,{d_3}^2c_2) =: q_3a + q_4$$

The basis G specifies well for all parameters in $\mathbb{K}^5 \setminus C_1$, where C_1 is subset of tuples of $\mathbb{K}^5$ vanishing in the polynomials q_1 and q_3, i.e., $V(q_1, q_3)$.

For each polynomial f there are only 3 possibles values of the parameter d_3. If there are infitine possible values for a or b for the polynomial f, then $(c_1, c_2, c_3, c_4, c_5) \in C_2 = V(q_1, q_2) \cup V(q_2, q_3)$.

Therefore, only for tuples in the closed and proper subset $C_1 \cup C_2$ of $\mathbb{K}^5$ can polynomials have infinite decompositions. That is, the fibers of $\Gamma_{2,4,2}$ are finite except on a closed and proper subset of $\mathbb{K}^5$. Therefore, $\dim \mathrm{Dec}_{2,4,2} = a_{2,2} + 2(a_{2,2} - 2) = 5$ and all polynomials, except those in a closed and proper subset of $\mathbb{K}^5$, are decomposable.

Counting the dimension of decomposable lists of homogeneous polynomials of the same degree is completely analogous. Let $\mathrm{Dec}_{m,n,r,u}$ be the set of lists $\mathbf{f}$ of u homogeneous polynomials in $\mathbb{K}[\mathbf{x}]$ of degree n that are (r, s)-decomposable, and let

$$\Gamma_{m,n,r,u} : (\mathbb{K}^{a_{m,r}})^u \times V \longrightarrow \overline{\mathrm{Dec}_{m,n,r,u}}$$

be the function that maps the coefficients of the normalized tuple $(\mathbf{g}, \mathbf{h})$ to the coefficients of $\mathbf{g} \circ \mathbf{h}$. If the above normalization were the good one, then the dimension of $\overline{\mathrm{Dec}_{m,n,r,u}}$ = would be $\dim((\mathbb{K}^{a_{m,r}})^u \times V) = u \cdot a_{m,r} + m \cdot (a_{m,s} - m)$.

References

1. Faugère, J.C., Perret, L.: Cryptanalysis of $2R^-$ schemes. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 357–372. Springer, Heidelberg (2006)
2. Faugère, J.C., Perret, L.: An efficient algorithm for decomposing multivariate polynomials and its applications to cryptography. Journal of Symbolic Computation 44, 1676–1689 (2009)
3. Faugère, J.C., Perret, L.: High order derivatives and decomposition of multivariate polynomials. In: Kaltofen, E. (ed.) ISSAC 2009: Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation, Seoul, Korea, pp. 207–214. ACM, New York (2009)
4. Gathen, J.v.z.: The number of decomposable univariate polynomials. In: ISSAC 2009: Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation, pp. 359–366. ACM, New York (2009) (extended abstract)
5. von zur Gathen, J.: Counting decomposable multivariate polynomials. Technical Report arXiv:0811.4726 (2008)
6. von zur Gathen, J., Gutierrez, J., Rubio, R.: Multivariate polynomial decomposition. Appl. Algebra Engrg. Comm. Comput. 14, 11–31 (2003)
7. Matsumoto, T., Imai, H.: Public quadratic polynomial-tuples for efficient signature-verification and message-encryption. In: Günther, C.G. (ed.) EUROCRYPT 1988. LNCS, vol. 330, pp. 419–453. Springer, Heidelberg (1988)
8. Mumford, D.: The red book of varieties and schemes. In: Thalheim, B. (ed.) Semantics in Databases 1995. LNCS, vol. 1358. Springer, Heidelberg (1998)
9. Patarin, J., Goubin, L.: Asymmetric cryptography wiht s-boxes. In: Han, Y., Quing, S. (eds.) ICICS 1997. LNCS, vol. 1334, pp. 369–380. Springer, Heidelberg (1997)
10. Ye, D., Dai, Z., Lam, K.Y.: Decomposing attacks on asymmetric cryptography based on mapping compositions. J. Cryptology 14, 137–150 (2001)

Computing Matrix Representations of Filiform Lie Algebras

Manuel Ceballos[1], Juan Núñez[1], and Ángel F. Tenorio[2]

[1] Departamento de Geometría y Topología, Facultad de Matemáticas, Universidad de Sevilla, Spain
mceballos@us.es, jnvaldes@us.es

[2] Dpto. de Economía, Métodos Cuantitativos e Historia Económica, Escuela Politécnica Superior, Universidad Pablo de Olavide, Ctra. Utrera km. 1. 41013-Seville, Spain
aftenorio@upo.es

Abstract. In this paper, we compute minimal faithful unitriangular matrix representations of filiform Lie algebras. To do it, we use the nilpotent Lie algebra, $\mathfrak{g}_n$, formed of $n \times n$ strictly upper-triangular matrices. More concretely, we search the lowest natural number n such that the Lie algebra $\mathfrak{g}_n$ contains a given filiform Lie algebra, also computing a representative of this algebra. All the computations in this paper have been done using MAPLE 9.5.

Keywords: Filiform Lie Algebra, Minimal Faithful Unitriangular Matrix Representation, Algorithm.

2000 Mathematics Subject Classification: 17B30, 68W40, 68Q25.

1 Introduction

Firstly, we would like to explain the motivation for dealing with filiform Lie algebras. At present, several aspects of Lie algebras remain unknown. In fact, the classification of nilpotent Lie algebras is still an open problem, although the classification of other types of Lie algebras (such as semisimple and simple ones) were already obtained in 1890. In this way, computing faithful representations of nilpotent Lie algebras is one of the main open problems in the theory of finite-dimensional Lie algebras over an algebraically closed field of characteristic zero. Therefore, it seems to be convenient to reduce this problem by dealing with filiform Lie algebras, which constitute the most structured Lie algebras in the nilpotent class. These algebras were introduced by Vergne [13] in the late 1960s.

On the other hand, the research on Lie Theory has a large number of applications to other sciences such as Applied Mathematics, Engineering, Physics, Mathematical Finance and Economics. In all of them the representation of Lie groups and algebras constitute an important subject. For example, in Economics, Polidoro in [10] studied a financial problem by using representation of nilpotent Lie groups. Additionally, Solvable and Nilpotent Lie algebras can be used to deal with problem relative to finance derivatives and other financial and economic problem. In this way, the pricing problem of Asian and European options can be solved using a Lie-algebra and Lie-symmetry approach as can be seen in [9] and [11]. Consequently, we think that providing explicit representations of filiform Lie algebra can be useful to tackle these problems.

Regarding the representation of filiform Lie algebras, the following result is well-known: Given a finite-dimensional nilpotent Lie algebra $\mathfrak{g}$, there exists $n \in \mathbb{N}$ such that $\mathfrak{g}$ is isomorphic to a subalgebra of the algebra $\mathfrak{g}_n$, of $n \times n$ strictly upper-triangular matrices [12, Theorem 3.6.6]. Therefore, an important question is to compute the minimal $n \in \mathbb{N}$ such that a finite-dimensional filiform Lie algebra is contained in $\mathfrak{g}_n$ as a subalgebra.

At this respect, Benjumea et al. [1] already introduced an algorithmic method to compute minimal faithful unitriangular matrix representations of nilpotent Lie groups and algebras (including filiform ones), giving some examples of application. Later, Benjumea et al. [2] gave the list of minimal faithful unitriangular matrix representations for nilpotent Lie algebras of dimension less than 6 using the previous method. Nevertheless, the following question is still unsolved: What Lie algebras have an n-dimensional representation for arbitrary dimensions? In this paper, we will determine which filiform Lie algebras satisfy that property, giving minimal faithful unitriangular representations in the case of both model and non-model algebras.

Other authors, like Burde [4] or Ghanam et al. [8], studied the minimal dimension $\mu(\mathfrak{g})$ for the representations of a given Lie algebra $\mathfrak{g}$. However, these authors considered any faithful $\mathfrak{g}$-module instead of the family of Lie algebras $\mathfrak{g}_n$. Consequently, the value of $\mu(\mathfrak{g})$ is less than or equal to the dimension which we will compute and determine in this paper. In particular, Ghanam et al. [8] computed matrix representations for low dimensional nilpotent Lie algebras, but their minimality was not studied. In fact, some representations in [8] were not minimal.

Independently, some authors like Echarte et al. [5] introduced some invariants of filiform Lie algebras, improving them in [6]. In this paper, we will recall the use of these invariants to express the law of filiform Lie algebras and to classify them.

The structure of this paper is as follows: after reviewing some well-known results about Lie Theory in Section 2, Section 3 is devoted to show the method used to compute a minimal faithful unitriangular matrix representation for filiform Lie algebras. Due to reasons of length, we only compute explicitly minimal faithful unitriangular matrix representations for filiform Lie algebras of dimension less than or equal to six, although the method can be applied to any arbitrary finite-dimensional filiform Lie algebra provided its law is known, which is not easy for higher dimensions. Remember that the classifications of filiform Lie algebras are only known up to dimension 11 (see [3]).

2 Preliminaries

Some preliminary concepts on Lie algebras (including Invariant and Representation Theories) are recalled in this section. For a general overview, the reader can consult [12]. Let us note that only finite-dimensional Lie algebras over the complex number field $\mathbb{C}$ are considered from here on.

2.1 Lie Algebras

The *lower central series* of a given Lie algebra $\mathfrak{g}$ is defined by

$$\mathcal{C}^1(\mathfrak{g}) = \mathfrak{g},\ \mathcal{C}^2(\mathfrak{g}) = [\mathfrak{g},\mathfrak{g}],\ \mathcal{C}^3(\mathfrak{g}) = [\mathcal{C}^2(\mathfrak{g}),\mathfrak{g}],\ \ldots,\ \mathcal{C}^k(\mathfrak{g}) = [\mathcal{C}^{k-1}(\mathfrak{g}),\mathfrak{g}],\ \ldots$$

The Lie algebra $\mathfrak{g}$ is *nilpotent* when there exists a natural integer m such that $\mathcal{C}^m(\mathfrak{g}) \equiv 0$.

Let $\mathfrak{h}$ be a subalgebra of a Lie algebra $\mathfrak{g}$. The centralizer of $\mathfrak{h}$ in $\mathfrak{g}$ is the set of elements of $\mathfrak{g}$ which commute with all of the elements of $\mathfrak{h}$.

Related to the lower central series associated with a subalgebra of $\mathfrak{g}$, the following result holds

Proposition 1. *Let $\mathfrak{h}$ be a subalgebra of a Lie algebra $\mathfrak{g}$. Then $\mathcal{C}^k(\mathfrak{h}) \subseteq \mathcal{C}^k(\mathfrak{g})$, $\forall\, k \in \mathbb{N}$.*

Let us denote by $\mathfrak{g}_n$ the nilpotent matrix algebra formed of all the $n \times n$ strictly upper-triangular matrices, with $n > 1$. The expression of the vectors in $\mathfrak{g}_n$ is the following

$$g_n(x_{r,s}) = \begin{pmatrix} 0 & x_{1,2} & \cdots & x_{1,n-1} & x_{1,n} \\ 0 & 0 & \cdots & x_{2,n-1} & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & x_{n-1,n} \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix} \qquad (x_{i,j} \in C).$$

The dimension of $\mathfrak{g}_n$ is $\frac{n(n-1)}{2}$. Fixed i and j such that $1 \leq i < j \leq n$, a basis of $\mathfrak{g}$ is $\mathcal{B}_n = \{X_{i,j} = g_n(x_{r,s}) \,|\, [x_{r,s} = 1 \Leftrightarrow (r,s) = (i,j)] \wedge [x_{r,s} = 0 \Leftrightarrow (r,s) \neq (i,j)]\}_{1 \leq i<j \leq n}$ with the law: $[X_{i,j}, X_{j,k}] = X_{i,k}$, for $1 \leq i < j < k \leq n$. Consequently, the dimension of each term in the lower central series of $\mathfrak{g}_n$ is

$$(\dim(\mathfrak{g}_n), \dim(\mathfrak{g}_{n-1}), \dim(\mathfrak{g}_{n-2}), \ldots, \dim(\mathfrak{g}_2), 0) \tag{1}$$

A particular family of nilpotent Lie algebras is formed of abelian Lie algebras. A Lie algebra $\mathfrak{g}$ is said to be *abelian* if $[\mathbf{v}, \mathbf{w}] = 0$, for all $\mathbf{v}$, $\mathbf{w} \in \mathfrak{g}$. An equivalent condition is the following: $Z(\mathfrak{g}) = \mathfrak{g}$, where

$$Z(\mathfrak{g}) = \{X \in \mathfrak{g} \quad | \quad [X, Y] = 0\,,\ \forall\, Y \in \mathfrak{g}\}$$

is the center of the algebra $\mathfrak{g}$. We also consider a second subclass of nilpotent Lie algebras in this paper: filiform Lie algebras. An n-dimensional Lie algebra $\mathfrak{g}$ is *filiform* if its lower central series satisfies the following

$$\dim(\mathcal{C}^1(\mathfrak{g})) = n,\ \dim(\mathcal{C}^2(\mathfrak{g})) = n-2,\ \dim(\mathcal{C}^3(\mathfrak{g})) = n-3,\ \ldots, \dim(\mathcal{C}^n(\mathfrak{g})) = 0. \tag{2}$$

A basis $\{e_i\}_{i=1}^n$ of the filiform Lie algebra $\mathfrak{g}$ is called an *adapted basis* if the following relations are verified

$$\begin{array}{lll} [e_1, e_h] = e_{h-1}, & \text{for } 3 \leq h \leq n; \\ [e_2, e_h] = 0, & \text{for } 1 \leq h \leq n; \\ [e_3, e_h] = 0, & \text{for } 2 \leq h \leq n. \end{array} \qquad (3)$$

If $\{e_i\}_{i=1}^n$ is an adapted basis of an n-dimensional filiform Lie algebra $\mathfrak{g}$, then the vector e_2 is the unique element in the center $Z(\mathfrak{g})$ of the algebra.

A filiform Lie algebra is called *model* if the only nonzero brackets in its law are $[e_1, e_h] = e_{h-1}$, for $3 \leq h \leq n$.

2.2 Invariants of Filiform Lie Algebras

This subsection is devoted to recall the definitions of two invariants for filiform Lie algebras given in [6]. First, the invariant z_1 is defined as follows

$$z_1 = \max\{k \in \mathbb{N} \mid C_{\mathfrak{g}}(\mathcal{C}^{n-k+2}(\mathfrak{g})) \supset \mathcal{C}^2(\mathfrak{g})\},$$

where $C_{\mathfrak{g}}(\mathfrak{h})$ is the centralizer of a given subalgebra $\mathfrak{h}$ of $\mathfrak{g}$. Let us note that the set in the previous definition can be empty. In this case, it is easy to prove that $\mathfrak{g}$ is a model filiform Lie algebra. Besides, the definition of z_1 means that the ideal $\mathcal{C}^{n-i+2}(\mathfrak{g})$ is the greatest one whose centralizer contains $\mathcal{C}^2(\mathfrak{g})$. Let us note that the previous definition is equivalent to the following: $z_1 = \min\{k \geq 2 \mid [e_k, e_n] \neq 0\}$, which is more convenient for practical use, and where $\{e_i\}_{i=1}^n$ is an adapted basis of $\mathfrak{g}$.

The invariant z_2 is defined as

$$z_2 = \max\{k \in \mathbb{N} \mid \mathcal{C}^{n-k+1}(\mathfrak{g}) \text{ is abelian}\}.$$

An immediate consequence of this definition is that the ideal $\mathcal{C}^{n-j+1}(\mathfrak{g}) \equiv \langle e_2, \ldots, e_j \rangle$ is the largest abelian subalgebra in the lower central series of $\mathfrak{g}$.

3 Computing Minimal Matrix Representations

In this section, we firstly obtain a minimal faithful unitriangular matrix representation for each model filiform Lie algebra. Next, we give a method to obtain such representations for non-model filiform Lie algebras. Finally, we compute minimal faithful unitriangular matrix representations for filiform Lie algebras of dimension less than 7.

Given a Lie algebra $\mathfrak{g}$, a *representation* of $\mathfrak{g}$ in $\mathbb{C}^n$ is a homomorphism of Lie algebras $\phi : \mathfrak{g} \to \mathfrak{gl}(\mathbb{C}^n) = \mathfrak{gl}(\mathbb{C}, n)$. The natural integer n is called the *dimension* of this representation. Ado's theorem states that every finite-dimensional Lie algebra over a field of characteristic zero has a linear injective representation on a finite-dimensional vector space, that is, a *faithful representation.*

Usually, representations are defined by using an arbitrary n-dimensional vector space V (see [7]) and homomorphisms of Lie algebras from $\mathfrak{g}$ to $\mathfrak{gl}(V)$ of endomorphisms of V; that is, by using $\mathfrak{g}$-*modules.*

With respect to minimal representations of Lie algebras, Burde [4] introduced the following invariant for an arbitrary Lie algebra $\mathfrak{g}$

$$\mu(\mathfrak{g}) = \min\{\dim(M) \mid M \text{ is a faithful } \mathfrak{g}\text{-module}\}.$$

In this section, matrix representations of filiform Lie algebras are studied. Moreover, we are interested in minimal matrix representations of these algebras with a particular restriction: the representations have to be contained in $\mathfrak{g}_n$. In this way, given a filiform Lie algebra $\mathfrak{g}$, we want to compute the minimal value n such that $\mathfrak{g}_n$ contains a subalgebra isomorphic to $\mathfrak{g}$. This value is also an invariant of $\mathfrak{g}$ and its expression is given by

$$\bar{\mu}(\mathfrak{g}) = \min\{n \in \mathbb{N} \mid \exists \text{ subalgebra of } \mathfrak{g}_n \text{ isomorphic to } \mathfrak{g}\}.$$

Let us note that the invariants $\mu(\mathfrak{g})$ and $\bar{\mu}(\mathfrak{g})$ can be different from each other.

Proposition 2. *Let $\mathfrak{g}$ be an n-dimensional filiform Lie algebra. Then $\bar{\mu}(\mathfrak{g}) \geq n$.*

Proof. We have to prove that for a given n-dimensional filiform Lie algebra $\mathfrak{g}$, it is not possible to find a subalgebra of $\mathfrak{g}_{n-1}$ isomorphic to $\mathfrak{g}$.
First, we express the vectors of an adapted basis $\{e_i\}_{i=1}^n$ of $\mathfrak{g}$ as linear combinations of the vectors in the basis $\mathcal{B}_{n-1}$ of $\mathfrak{g}_{n-1}$

$$e_k = \sum_{1 \leq i < j \leq n-1} \lambda_{i,j}^k X_{i,j}, \text{ for } 1 \leq k \leq n.$$

We will prove that each coefficient $\lambda_{i,j}^2$ of $e_2 \in Z(g)$ has to be zero. Effectively: from $[e_1, e_h] = e_{h-1}$ for $3 \leq h \leq n$, the following relations are obtained

$$\lambda_{\beta,\beta+1}^{h-1} = 0;\ \lambda_{\beta,\alpha_\beta}^{h-1} = \sum_{\beta < p < \alpha_\beta} (\lambda_{\beta,p}^1 \lambda_{p,\alpha_\beta}^h - \lambda_{p,\alpha_\beta}^1 \lambda_{\beta,p}^h), \text{ for } 1 \leq \beta \leq n-2 \text{ and } \alpha_\beta \geq \beta+2. \quad (4)$$

From $[e_1, e_3] = e_2$, we can conclude that $\lambda_{\beta,\beta+1}^2 = 0$, for $1 \leq \beta \leq n-2$. Now, we prove that $\lambda_{l,\alpha_l}^2 = 0$, for $1 \leq l \leq n-3$. To do so, we only need to prove that $\lambda_{p,\alpha_\beta}^3 = \lambda_{\beta,p}^3 = 0$.

From $[e_1, e_k] = e_{k-1}$ for $3 \leq k \leq n-1$, we can affirm that $\lambda_{\beta,\beta+1}^{k-1} = 0$ for $1 \leq \beta \leq n-2$. This implies that $\lambda_{p,q}^3 = 0$ when $q - p < n-4$.

If we consider the bracket $[e_1, e_n] = e_{n-1}$, we conclude that $\lambda_{\beta,\beta+1}^{n-1} = 0$, and, therefore, $\lambda_{p,q}^3 = 0$, where $q - p = n-3$.

Consequently, all the coefficients of e_2 are null and this comes into contradiction. □

3.1 Model Filiform Lie Algebras

The law of a fixed n-dimensional model filiform Lie algebra $\mathfrak{g}$ with an adapted basis $\{e_i\}_{i=1}^n$ is the following

$$[e_1, e_h] = e_{h-1}, \quad \text{for } 3 \leq h \leq n. \tag{5}$$

Now, we define the vectors of the adapted basis as linear combinations of the vectors in the basis $\mathcal{B}_n$ of the Lie algebra $\mathfrak{g}_n$

$$e_1 = \sum_{i=1}^{n-2} X_{i,i+1},\ e_2 = X_{1,n},\ e_3 = X_{2,n},\ \dots,\ e_n = X_{n-1,n} \tag{6}$$

In this way, we can define a subalgebra $\mathfrak{f}'_n$ of $\mathfrak{g}_n$ whose elements have the following form

$$f'_n(x_k) = \begin{pmatrix} 0 & x_1 & 0 & \cdots & 0 & x_2 \\ 0 & 0 & x_1 & \cdots & 0 & x_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & x_1 & x_{n-1} \\ 0 & 0 & 0 & \cdots & 0 & x_n \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{pmatrix} \qquad (x_k \in \mathbb{C},\ \text{for } k = 1, \dots, n).$$

The dimension of $\mathfrak{f}'_n$ is n and a basis of $\mathfrak{f}'_n$ is given by the vectors

$$e_h = f'_n(x_k), \quad \text{with } x_k = \begin{cases} 1, \text{if } k = h; \\ 0, \text{if } k \neq h. \end{cases}$$

According to Proposition 2, we can affirm that every n-dimensional model filiform Lie algebra has an n-dimensional minimal faithful unitriangular matrix representation with the representative given in (6).

3.2 Non-Model Filiform Lie Algebras

If the filiform Lie algebra is non-model, then the invariants z_1 and z_2 exist. Hence, there exist some additional nonzero brackets to $[e_1, e_h] = e_{h-1}$, for $3 \leq h \leq n$. Consequently, non-model filiform Lie algebras cannot be represented by the algebras $\mathfrak{f}'_n$.

Now, we show an algorithmic method to compute minimal faithful unitriangular matrix representations for non-model filiform Lie algebras. These representations are minimal in the following sense: Finding a faithful matrix representation of a given Lie algebra $\mathfrak{g}$ in $\mathfrak{g}_n$ means that no representations of $\mathfrak{g}$ can be obtained in $\mathfrak{g}_{n-1}$.

To do so, we give a step-by-step explanation of the method used to determine these minimal representations for a given filiform Lie algebra $\mathfrak{g}$ of dimension $n > 4$. Let us note that this method is better than the one given in [1] for filiform Lie algebras, since it uses properties of filiformity and of the invariants z_1 and z_2 to simplify and avoid computations.

1. According to Proposition 1, we compute the first natural integer l such that the lower central series of $\mathfrak{g}_l$ is compatible with the one associated with $\mathfrak{g}$. By bearing in mind expressions (1) and (2) and Proposition 2 (which is equivalent to the fact that $Z(\mathfrak{g}) = \langle e_2 \rangle = \mathcal{C}^{n-1}(\mathfrak{g}) \nsubseteq \mathcal{C}^{n-1}(\mathfrak{g}_{n-1})$), we will consider $l = n$ for n-dimensional filiform Lie algebras.
2. In virtue of Proposition 2, we have ruled out the Lie algebras $\mathfrak{g}_p$ with $p < l$ (in our case, $p < n$). Hence, we search a subalgebra of $\mathfrak{g}_l$ isomorphic to $\mathfrak{g}$. To do so, an adapted basis $\{e_i\}_{i=1}^n$ of $\mathfrak{g}$ is considered and its vectors are expressed as linear combinations of the basis $\mathcal{B}_n$
$$e_h = \sum_{1 \le i < j \le n} \lambda_{i,j}^h X_{i,j}, \quad \text{for } 1 \le h \le n.$$
3. Then, we impose the brackets given in (5). In this way, the relations shown in (4) are obtained again.
4. After solving the resulting system of equations, we solve the new system obtained when imposing the rest of nonzero brackets in the law of $\mathfrak{g}$.

The solutions of this system depend on the particular Lie algebra studied in each moment. Hence, we have categorized the solutions in a finite number of cases using the invariants z_1 and z_2. The solutions of this system will be computed by using the symbolic computation package Maple 9.5, with the command `solve`, which receives a list of equations and a list of variables as inputs. This command works efficiently with polynomial equations and returns the algebraic expression of the solutions as output. To obtain these solutions, a substitution procedure is used between the algebraic variables. In this way, we will show the family of representatives under certain conditions for the parameters.

Another point to consider is the number of solutions for the system. To study it, we will define the set F of polynomial expressions and we will use the command `is_finite`, which receives as input the set F and determines if the number of solutions is or not finite. Noether normalization lemma is also very useful to describe the elements in an algebraic variety.

Furthermore, for computing a particular solution of the previous system, we will search one whose number of null coefficients is as greater as possible. In this way, the coefficients can be assumed equal to zero when they do not appear in the relations obtained. This will be a natural representative of the Lie algebra $\mathfrak{g}$.

We will also show some examples. The first of them is about a model filiform Lie algebra. Next, we apply the previous algorithmic method to compute the minimal matrix representation of two non-model filiform Lie algebras.

Example 1. Let us consider the model filiform Lie algebra $\mathfrak{f}_4^1$ generated by the vectors $\{e_1, e_2, e_3, e_4\}$ with the law $[e_1, e_3] = e_2$ and $[e_1, e_4] = e_3$.
representation of $\mathfrak{f}_4^1$ by using the Lie algebra $\mathfrak{f}_4'$ as it was shown in the previous subsection. Now, we apply the previous algorithm and determine if the number of solution is a finite or not for the system so obtained. First, we define the vectors

$$e_1 = \lambda^1_{1,2}X_{1,2} + \lambda^1_{1,3}X_{1,3} + \ldots + \lambda^1_{3,4}X_{3,4};$$

$$e_2 = \lambda^2_{1,2}X_{1,2} + \lambda^2_{1,3}X_{1,3} + \ldots + \lambda^2_{3,4}X_{3,4};$$

$$e_3 = \lambda^3_{1,2}X_{1,2} + \lambda^3_{1,3}X_{1,3} + \ldots + \lambda^3_{3,4}X_{3,4};$$

$$e_4 = \lambda^4_{1,2}X_{1,2} + \lambda^4_{1,3}X_{1,3} + \ldots + \lambda^4_{3,4}X_{3,4}.$$

Now, we introduce the following commands in Maple

```
> ec1:=c121*c233-c231*c123-c132:
> ec2:=c121*c243+c131*c343-c241*c123-c341*c133-c142:
> ec3:=c231*c343-c341*c233-c242: ec4:=c122: ec5:=c232:
> ec6:=c342:ec7:=c121*c234-c231*c124-c133:
> ec8:=c121*c244+c131*c344-c241*c124-c341*c134-c143:
> ec9:=c231*c344-c341*c234-c243: ec10:=c123:ec11:=c233:
> ec12:=c343:ec13:=c121*c232-c231*c122:
> ec14:=c121*c242+c131*c342-c241*c122-c341*c132:
> ec15:=c231*c342-c341*c232: ec16:=c122*c233-c232*c123:
> ec17:=c122*c243+c132*c343-c242*c123-c342*c133:
> ec18:=c232*c343-c342*c233: ec19:=c122*c234-c232*c124:
> ec20:=c122*c244+c132*c344-c242*c124-c342*c134:
> ec21:=c232*c344-c342*c234: ec22:=c123*c234-c233*c124:
> ec23:=c123*c244+c133*c344-c243*c124-c343*c134:
> ec24:=c233*c344-c343*c234:

> F:=[ec1,ec2,ec3,ec4,ec5,ec6,ec7,ec8,ec9,ec10,ec11,ec12,
ec13,ec14,ec15,ec16,ec17,ec18,ec19,ec20,ec21,ec22,ec23,ec24];

>is_finite(F);
                                false
```

Consequently, there is an infinite number of solutions. However, if we apply Noether normalization lemma and we intersect the previous set with the algebraic variety defined by the system $\{\lambda^i_{1,2} = 0\}^4_{i=1}$, we obtain a finite number of solutions:

```
eq1:=c121: eq2:=c122: eq3:=c123: eq4:=c124:

H:=[ec1,ec2,ec3,ec4,ec5,ec6,ec7,ec8,ec9,ec10,ec11,ec12,ec13,ec14,
ec15,ec16,ec17,ec18,ec19,ec20,ec21,ec22,ec23,ec24,eq1,eq2,eq3,
eq4]:

> is_finite(H);
                                true
```

Let us note that this reasoning can be done for every model filiform Lie algebra.

Example 2. We consider the Lie algebra $\mathfrak{f}_5^2$ generated by the vectors $\{e_i\}_{i=1}^5$ with the following nonzero brackets

$$[e_1, e_h] = e_{h-1}, \text{ for } 3 \le h \le n; \; [e_4, e_5] = e_2.$$

The lower central series of $\mathfrak{g}_5$ and $\mathfrak{f}_5^2$ are compatible

$$\mathcal{C}^2(\mathfrak{f}_5^2) = \langle e_2, e_3, e_4 \rangle \subseteq \mathcal{C}^2(\mathfrak{g}_5) = \langle X_{1,3}, X_{1,4}, X_{1,5}, X_{2,4}, X_{2,5}, X_{3,5} \rangle$$

$$\mathcal{C}^3(\mathfrak{f}_5^2) = \langle e_2, e_3 \rangle \subseteq \mathcal{C}^3(\mathfrak{g}_5) = \langle X_{1,4}, X_{1,5}, X_{2,5} \rangle$$

$$\mathcal{C}^4(\mathfrak{f}_5^2) = \langle e_2 \rangle \subseteq \mathcal{C}^4(\mathfrak{g}_5) = \langle X_{1,5} \rangle$$

When applying Steps 2 and 3, the following vectors are obtained

$$e_1 = \lambda_{1,2}^1 X_{1,2} + \lambda_{1,3}^1 X_{1,3} + \ldots + \lambda_{3,5}^1 X_{3,5} + \lambda_{4,5}^1 X_{4,5};$$

$$e_2 = \lambda_{1,5}^2 X_{1,5}; \quad e_3 = \lambda_{1,4}^3 X_{1,4} + \lambda_{1,5}^3 X_{1,5} + \lambda_{2,5}^3 X_{2,5};$$

$$e_4 = \lambda_{1,3}^4 X_{1,3} + \lambda_{1,4}^4 X_{1,4} + \lambda_{1,5}^4 X_{1,5} + \lambda_{2,4}^4 X_{2,4} + \lambda_{2,5}^4 X_{2,5} + \lambda_{3,5}^4 X_{3,5};$$

$$e_5 = \lambda_{1,2}^5 X_{1,2} + \lambda_{1,3}^5 X_{1,3} + \ldots + \lambda_{3,5}^5 X_{3,5} + \lambda_{4,5}^5 X_{4,5};$$

restricted under the following relations

$$\begin{cases}
\lambda_{3,5}^4 = \lambda_{3,4}^1 \lambda_{4,5}^5 - \lambda_{4,5}^1 \lambda_{3,4}^5, \; \lambda_{2,5}^3 = \lambda_{2,3}^1 \lambda_{3,4}^1 \lambda_{4,5}^5 - 2\lambda_{4,5}^1 \lambda_{2,3}^1 \lambda_{3,4}^5 + \lambda_{4,5}^1 \lambda_{3,4}^1 \lambda_{2,3}^5, \\
\lambda_{1,5}^4 = \lambda_{1,2}^1 \lambda_{2,5}^5 + \lambda_{1,3}^1 \lambda_{3,5}^5 + \lambda_{1,4}^1 \lambda_{4,5}^5 - \lambda_{4,5}^1 \lambda_{1,4}^5 - \lambda_{3,5}^1 \lambda_{1,3}^5 - \lambda_{2,5}^1 \lambda_{1,2}^5, \\
\lambda_{1,4}^3 = \lambda_{1,2}^1 \lambda_{2,3}^1 \lambda_{3,4}^5 - 2\lambda_{1,2}^1 \lambda_{3,4}^1 \lambda_{2,3}^5 + \lambda_{3,4}^1 \lambda_{2,3}^1 \lambda_{1,2}^5, \\
\lambda_{1,5}^2 = \lambda_{1,2}^1 \lambda_{2,3}^1 \lambda_{3,4}^1 \lambda_{4,5}^5 - 3\lambda_{4,5}^1 \lambda_{1,2}^1 \lambda_{2,3}^1 \lambda_{3,4}^5 + 3\lambda_{4,5}^1 \lambda_{1,2}^1 \lambda_{3,4}^1 \lambda_{2,3}^5 - \lambda_{4,5}^1 \lambda_{3,4}^1 \lambda_{2,3}^1 \lambda_{1,2}^5, \\
\lambda_{1,5}^3 = \lambda_{1,2}^1 \lambda_{2,3}^1 \lambda_{3,5}^5 + \lambda_{1,2}^1 \lambda_{2,4}^1 \lambda_{4,5}^5 - 2\lambda_{4,5}^1 \lambda_{1,2}^1 \lambda_{2,4}^5 - \lambda_{1,2}^1 \lambda_{3,4}^1 \lambda_{2,3}^5 + \lambda_{1,3}^1 \lambda_{3,4}^1 \lambda_{4,5}^5 - \\
2\lambda_{1,3}^1 \lambda_{4,5}^1 \lambda_{3,4}^5 + \lambda_{4,5}^1 \lambda_{2,4}^1 \lambda_{1,2}^5 + \lambda_{4,5}^1 \lambda_{3,4}^1 \lambda_{1,3}^5 - \lambda_{3,5}^1 \lambda_{1,2}^1 \lambda_{2,3}^5 + \lambda_{3,5}^1 \lambda_{2,3}^1 \lambda_{1,2}^5, \\
\lambda_{1,3}^4 = \lambda_{1,2}^1 \lambda_{2,3}^5 - \lambda_{2,3}^1 \lambda_{1,2}^5, \; \lambda_{1,4}^4 = \lambda_{1,2}^1 \lambda_{2,4}^5 + \lambda_{1,3}^1 \lambda_{3,4}^5 - \lambda_{2,4}^1 \lambda_{1,2}^5 - \lambda_{3,4}^1 \lambda_{1,3}^5, \\
\lambda_{2,5}^4 = \lambda_{2,3}^1 \lambda_{3,5}^5 + \lambda_{2,4}^1 \lambda_{4,5}^5 - \lambda_{4,5}^1 \lambda_{2,4}^5 - \lambda_{3,4}^1 \lambda_{2,3}^5, \; \lambda_{2,4}^4 = \lambda_{2,3}^1 \lambda_{3,4}^5 - \lambda_{3,4}^1 \lambda_{2,3}^5.
\end{cases}$$

This system has degree four. It is well-known the existence of efficient methods in Algebraic Geometry to solve this type of systems. For filiform Lie algebras of higher dimension, the degree of the system obtained is always one unit less than the dimension of the algebra considered.

Now, by imposing the bracket $[e_4, e_5] = e_2$, the following vectors are obtained

$$e_1 = \lambda_{1,2}^1 X_{1,2} + \lambda_{1,3}^1 X_{1,3} + \ldots + \lambda_{3,5}^1 X_{3,5} + \lambda_{4,5}^1 X_{4,5};$$

$$e_2 = \lambda_{1,5}^2 X_{1,5}; \quad e_3 = \lambda_{1,4}^3 X_{1,4} + \lambda_{1,5}^3 X_{1,5} + \lambda_{2,5}^3 X_{2,5};$$

$$e_4 = \lambda_{1,3}^4 X_{1,3} + \lambda_{1,4}^4 X_{1,4} + \lambda_{1,5}^4 X_{1,5} + \lambda_{2,5}^4 X_{2,5} + \lambda_{3,5}^4 X_{3,5};$$

$$e_5 = \lambda_{1,2}^5 X_{1,2} + \lambda_{1,3}^5 X_{1,3} + \ldots + \lambda_{1,5}^5 X_{1,5} + \lambda_{2,4}^5 X_{2,4} + \ldots + \lambda_{4,5}^5 X_{4,5};$$

under these relations:

$$\begin{cases} \lambda^4_{1,3} = -\lambda^1_{2,3}\lambda^5_{1,2},\ \lambda^4_{3,5} = \lambda^1_{3,4}\lambda^5_{4,5},\ \lambda^2_{1,5} = \lambda^1_{1,2}\lambda^1_{2,3}\lambda^1_{3,4}\lambda^5_{4,5} - \lambda^1_{4,5}\lambda^1_{3,4}\lambda^1_{2,3}\lambda^5_{1,2}, \\ -2\lambda^1_{3,4}\lambda^5_{4,5}\lambda^5_{1,3} = 2\lambda^5_{3,5}\lambda^1_{2,3}\lambda^5_{1,2} - \lambda^5_{4,5}\lambda^1_{1,2}\lambda^5_{2,4} + 2\lambda^5_{4,5}\lambda^1_{2,4}\lambda^5_{1,2} - \lambda^5_{1,2}\lambda^1_{4,5}\lambda^5_{2,4} \\ +\lambda^1_{1,2}\lambda^1_{2,3}\lambda^1_{3,4}\lambda^5_{4,5} - \lambda^1_{4,5}\lambda^1_{3,4}\lambda^1_{2,3}\lambda^5_{1,2},\ \lambda^3_{1,4} = \lambda^1_{3,4}\lambda^1_{2,3}\lambda^5_{1,2}, \lambda^3_{2,5} = \lambda^1_{2,3}\lambda^1_{3,4}\lambda^5_{4,5}, \\ 2\lambda^1_{3,4}\lambda^5_{4,5}\lambda^4_{1,5} = 2\lambda^1_{1,2}\lambda^5_{2,5}\lambda^1_{3,4}\lambda^5_{4,5} + 2\lambda^1_{1,3}\lambda^5_{3,5}\lambda^1_{3,4}\lambda^5_{4,5} + 2\lambda^1_{1,4}(\lambda^5_{4,5})^2\lambda^1_{3,4} \\ -2\lambda^1_{4,5}\lambda^5_{1,4}\lambda^1_{3,4}\lambda^5_{4,5} + 2\lambda^1_{3,5}\lambda^1_{2,3}\lambda^5_{1,2}\lambda^5_{3,5} - \lambda^1_{3,5}\lambda^5_{4,5}\lambda^1_{1,2}\lambda^5_{2,4} + 2\lambda^5_{4,5}\lambda^1_{2,4}\lambda^5_{1,2}\lambda^1_{3,5} \\ -\lambda^1_{4,5}\lambda^1_{3,5}\lambda^5_{1,2}\lambda^5_{2,4} + \lambda^1_{3,5}\lambda^1_{1,2}\lambda^1_{2,3}\lambda^1_{3,4}\lambda^5_{4,5} - \lambda^1_{3,5}\lambda^1_{4,5}\lambda^1_{3,4}\lambda^1_{2,3}\lambda^5_{1,2} - 2\lambda^1_{2,5}\lambda^5_{1,2}\lambda^1_{3,4}\lambda^5_{4,5}, \\ \lambda^4_{2,5} = \lambda^1_{2,3}\lambda^5_{3,5} + \lambda^1_{2,4}\lambda^5_{4,5} - \lambda^1_{4,5}\lambda^5_{2,4},\ 2\lambda^5_{4,5}\lambda^4_{1,4} = \lambda^5_{4,5}\lambda^1_{1,2}\lambda^5_{2,4} + 2\lambda^5_{3,5}\lambda^1_{2,3}\lambda^5_{1,2} \\ -\lambda^5_{1,2}\lambda^1_{4,5}\lambda^5_{2,4} + \lambda^1_{1,2}\lambda^1_{2,3}\lambda^1_{3,4}\lambda^5_{4,5} - \lambda^1_{4,5}\lambda^1_{3,4}\lambda^1_{2,3}\lambda^5_{1,2},\ -2\lambda^5_{4,5}\lambda^3_{1,5} = \\ -2\lambda^1_{1,2}\lambda^5_{4,5}\lambda^1_{2,3}\lambda^5_{3,5} - 2\lambda^1_{1,2}(\lambda^5_{4,5})^2\lambda^1_{2,4} + 3\lambda^1_{1,2}\lambda^5_{4,5}\lambda^1_{4,5}\lambda^5_{2,4} - 2\lambda^1_{1,3}\lambda^1_{3,4}(\lambda^5_{4,5})^2 \\ +2\lambda^1_{4,5}\lambda^5_{3,5}\lambda^1_{2,3}\lambda^5_{1,2} - (\lambda^1_{4,5})^2\lambda^5_{2,4}\lambda^5_{1,2} + \lambda^1_{4,5}\lambda^1_{1,2}\lambda^1_{2,3}\lambda^1_{3,4}\lambda^5_{4,5} - (\lambda^1_{4,5})^2\lambda^1_{3,4}\lambda^1_{2,3}\lambda^5_{1,2} \\ -2\lambda^1_{3,5}\lambda^1_{2,3}\lambda^5_{1,2}\lambda^5_{4,5}. \end{cases}$$

Table 1. Representatives of minimal faithful unitriangular representations of filiform Lie algebras of dimension ≤ 7

Algebra	**Nonzero brackets**	**Representation**
$\mathfrak{f}^1_3$	$[e_1, e_3] = e_2$.	$e_1 = X_{1,2}, e_2 = X_{1,3}, e_3 = X_{2,3}$.
$\mathfrak{f}^1_4$	$[e_1, e_3] = e_2, [e_1, e_4] = e_3$.	$e_1 = X_{1,2} + X_{2,3}, e_2 = X_{1,4}, e_3 = X_{2,4}, e_4 = X_{3,4}$.
$\mathfrak{f}^1_5$	$[e_1, e_h] = e_{h-1}\,(h \geq 3)$.	$e_1 = X_{1,2} + X_{2,3} + X_{3,4}, e_2 = X_{1,5}, e_3 = X_{2,5}, e_4 = X_{3,5}, e_5 = X_{4,5}$.
$\mathfrak{f}^2_5$	$[e_1, e_h] = e_{h-1}\,(h \geq 3)$, $[e_4, e_5] = e_2$.	$e_1 = X_{1,2} + X_{2,3} + X_{3,4}, e_2 = X_{1,5}, e_3 = X_{2,5}, e_4 = X_{1,4} + X_{3,5}, e_5 = X_{2,4} + X_{4,5}$.
$\mathfrak{f}^1_6$	$[e_1, e_h] = e_{h-1}\,(h \geq 3)$.	$e_1 = X_{1,2} + X_{2,3} + X_{3,4}, e_2 = X_{1,5}, e_3 = X_{2,5}, e_4 = X_{3,5}, e_5 = X_{4,5}, e_6 = X_{5,6}$.
$\mathfrak{f}^2_6$	$[e_1, e_h] = e_{h-1}\,(h \geq 3)$, $[e_5, e_6] = e_2$.	$e_1 = \sum_{i=1}^4 X_{i,i+1}, e_2 = X_{1,6}, e_3 = X_{2,6}, e_4 = X_{3,6}, e_5 = X_{1,5} + X_{4,6}, e_6 = X_{2,5} + X_{4,6}$.
$\mathfrak{f}^3_6$	$[e_1, e_h] = e_{h-1}\,(h \geq 3)$, $[e_4, e_6] = e_2$, $[e_5, e_6] = e_3$.	$e_1 = \sum_{i=1}^4 X_{i,i+1}, e_2 = X_{1,6}, e_3 = X_{2,6}, e_4 = X_{3,6} + X_{1,5}, e_5 = X_{2,5} + X_{4,6}, e_6 = X_{3,5} + X_{5,6}$.
$\mathfrak{f}^4_6$	$[e_1, e_h] = e_{h-1}\,(h \geq 3)$, $[e_4, e_5] = e_2$ $[e_4, e_6] = e_3$, $[e_5, e_6] = e_4$.	$e_1 = X_{1,2} + X_{1,3} - \frac{2}{3}X_{1,4} + \frac{7}{3}X_{1,5} + X_{1,6} + X_{2,3} + X_{2,5} + X_{2,6} - X_{3,4} + 2X_{3,5} + X_{3,6} - X_{4,5} + X_{4,6} + X_{5,6}, e_2 = \frac{8}{3}X_{1,6}, e_3 = \frac{-2}{3}X_{1,5} + \frac{4}{3}X_{1,6} + 2X_{2,6}, e_4 = \frac{-2}{3}X_{1,4} + \frac{10}{3}X_{1,5} + \frac{7}{3}X_{1,6} + 2X_{2,6} + 2X_{3,6}, e_5 = \frac{-2}{3}X_{1,3} + 3X_{1,4} - \frac{4}{3}X_{1,5} + X_{1,6} - X_{2,5} + X_{2,6} - 2X_{4,6}, e_6 = \frac{-1}{3}X_{1,2} + X_{1,3} + X_{1,4} + X_{1,5} + X_{1,6} - X_{2,3} + X_{2,4} + X_{2,6} + X_{3,4} - X_{3,5} + X_{3,6} + X_{4,5} + X_{4,6} + X_{5,6}$.
$\mathfrak{f}^5_6$	$[e_1, e_h] = e_{h-1}\,(h \geq 3)$, $[e_4, e_5] = e_2$ $[e_4, e_6] = e_3 + e_2$, $[e_5, e_6] = e_4 + e_3$.	$e_1 = -X_{1,2} + X_{1,3} + \frac{11}{4}X_{1,4} + X_{1,5} + X_{1,6} + X_{2,3} + X_{2,4} + X_{2,5} + X_{2,6} + X_{3,4} + X_{3,5} + X_{3,6} - X_{4,5} + X_{4,6} + X_{5,6}, e_2 = 6X_{1,6}, e_3 = -3X_{1,5} - 11X_{1,6} - 3X_{2,6}, e_4 = -2X_{1,4} + 3X_{1,5} - \frac{13}{4}X_{1,6} + X_{2,5} + 8X_{2,6} - 2X_{3,6}, e_5 = X_{1,3} - \frac{13}{4}X_{1,5} - \frac{1}{4}X_{1,6} + X_{2,4} - 4X_{2,5} + 7X_{2,6} + 7X_{3,6} - 2X_{4,6}, e_6 = X_{1,2} + X_{1,3} + X_{1,4} + X_{1,5} + X_{1,6} - 2X_{2,3} - 3X_{2,4} + X_{2,5} + X_{2,6} - X_{3,4} - 4X_{3,5} + X_{3,6} + X_{4,5} + X_{4,6} + X_{5,6}$.

Next, we show a particular solution to obtain a representative for the Lie algebra $\mathfrak{f}_5^2$

$$e_1 = X_{1,2} + X_{2,3} + X_{3,4}, \quad e_2 = X_{1,5}, \quad e_3 = X_{2,5},$$
$$e_4 = X_{1,4} + X_{3,5}, \quad e_5 = X_{2,4} + X_{4,5}.$$

3.3 Filiform Lie Algebras of Dimension Less than 7

Due to reasons of length, Table 1 only shows minimal faithful unitriangular matrix representations for filiform Lie algebras of dimension less than 7, computing a natural representative and considering the classification given in [3]. Let us note that this table is very useful in order to find explicit representations of these algebras.

4 Conclusions

At present, researchers on Lie and Representation Theories need to deal with examples of Lie algebras of higher dimensions. The representation of these algebras is a difficult task and, besides, only the classifications of solvable, nilpotent and filiform complex Lie algebras up to dimension 6, 7 and 12, respectively, are actually known.

In this paper we have shown a method to obtain minimal faithful unitriangular matrix representations of filiform Lie algebras of any arbitrary dimension. Therefore, we think that it constitutes a little step forward to tackle the problem of the classification of these algebras and the obtainment of their representations.

Acknowledgments

The authors want to thank Professor Francisco J. Castro, from the University of Seville, for some useful suggestions which helped to improve this paper.

References

1. Benjumea, J.C., Echarte, F.J., Núñez, J., Tenorio, A.F.: A method to obtain the Lie group associated with a nilpotent Lie algebra. Computers & Mathematics with Applications 51, 1493–1506 (2006)
2. Benjumea, J.C., Núñez, J., Tenorio, A.F.: Minimal linear representations of the low-dimensional nilpotent Lie algebras. Mathematica Scandinavica 102, 17–26 (2008)
3. Boza, L., Fedriani, E.M., Núñez, J.: A new method for classifying complex filiform Lie algebras. Applied Mathematics and Computation 121, 169–175 (2001)
4. Burde, D.: On a refinement of Ado's Theorem. Arch. Math. (Basel) 70, 118–127 (1998)
5. Echarte, F.J., Núñez, J., Ramírez, F.: Relations among invariants of complex filiform Lie algebras. Applied Mathematics and Computation 147, 365–376 (2004)

6. Echarte, F.J., Núñez, J., Ramírez, F.: Description of some families of filiform Lie algebras. Houston Journal of Mathematics 34, 19–32 (2008)
7. Fulton, W., Harris, J.: Representation theory: a first course. Springer, New York (1991)
8. Ghanam, R., Strugar, I., Thompson, G.: Matrix representations for low dimensional Lie algebras. Extracta. Math. 20, 151–184 (2005)
9. Lo, C.F., Hui, C.H.: Valuation of financial derivatives with time-dependent parameters: Lie-algebraic approach. Quantitative Finance 1, 73–78 (2001)
10. Polidoro, S.: A Nonlinear PDE in Mathematical Finance. In: Brezzi, F., Buffa, A., Corsaro, S., Murli, A. (eds.) Numerical Mathematics and Advanced Application. Springer, Heidelberg (2003)
11. Taylor, S., Glasgow, S.: A novel reduction of the simple Asian option and Lie-Group invariant solutions. Int. J. Theoret. Appl. Finance 12(8), 1197–1212 (2009)
12. Varadarajan, V.S.: Lie Groups, Lie Algebras and Their Representations. Springer, New York (1984)
13. Vergne, M.: Cohomologie des algèbres de Lie nilpotentes, Application à l'étude de la variété des algebres de Lie nilpotentes. Bull. Soc. Math. France 98, 81–116 (1970)

Type Specialization in Aldor

Laurentiu Dragan and Stephen M. Watt

Computer Science Department
The University of Western Ontario
London, Canada
{ldragan,watt}@csd.uwo.ca

Abstract. Computer algebra in scientific computation squarely faces the dilemma of natural mathematical expression versus efficiency. While higher-order programming constructs and parametric polymorphism provide a natural and expressive language for mathematical abstractions, they can come at a considerable cost. We investigate how deeply nested type constructions may be optimized to achieve performance similar to that of hand-tuned code written in lower-level languages.

1 Introduction

One of the challenges for computer algebra in scientific computation is to achieve high performance when both symbolic and numeric computation are required. Aldor has sought to do this without sacrificing the natural high-level expression of mathematical algorithms.

Aldor has been designed to provide a rich set of composable abstraction mechanisms so that mathematical programs can be written in a natural style. These mechanisms use dependent types and type-producing functions to allow programs to be written in their most general form, and then specialized. The Aldor compiler manages to achieve performance comparable to optimized C or Lisp for programs that perform numeric or symbolic computations, respectively.

Aldor's type-producing functions can be used in a similar manner to templates in C++ or "generics" in Java, but with more natural mathematical structure. Even though writing generic code is highly desirable from the code re-usability perspective, programmers often avoid this approach for performance reasons. Deeply nested generic type constructions (which occur naturally in computer algebra) can hinder the performance of programs to an extent that entices some programmers to specialize code by hand. To avoid this, we propose an optimization that automatically specializes generic code based on the particular type instantiations used in a program. Our optimization specializes both the code and the data representation of the type, producing programs up to an order of magnitude faster.

Beyond its applicability to computer algebra, this work has implications for main-stream programming languages. Because programs in languages such as C++, Java and C# are only now starting to use templates extensively, the performance problems associated with deeply nested generic types have not been widely recognized in those settings.

The contributions of this paper are:

– a compiler optimization that specializes dynamic domain constructors when instantiations are known at compile time,
– an extension to the type specialization optimization that specializes also the data representation of the specialized type, and
– some numerical indication of the improvements that can be expected by an automatic process, and some results for a hand-crafted example.

The remainder of this paper is organized as follows: Section 2 gives a brief introduction to parametric polymorphism implementations and introduces the Aldor programming language and its compiler. Section 3 presents the problem presented by deeply nested type constructions. Section 4 describes the method of code specialization. Section 5 presents the approach used for data specialization. Section 6 presents some performance results and Section 7 concludes the paper.

2 Background

Approaches to parametric polymorphism: There are currently two approaches to implement parametric polymorphism, the *homogeneous* and *heterogeneous* approaches.

The *heterogeneous* approach constructs a special class for each different use of the type parameters. For example, with `vector` from the C++ standard template library, one can construct `vector<int>` or `vector<long>`. This generates two distinct classes that we may think of as: `vector_int` and `vector_long`. This is done by duplicating the code of the vector generic class and producing specialized forms. This is an effective approach from the time efficiency point of view. Two drawbacks of this method are the size of the code and that all parameters must be known statically. Another drawback is that constructors may not themselves be passed as run-time parameters.

The *homogeneous* approach uses the same generic class for every instance of the type parameters. This method is used in Java by erasing the type information, using the Object class in place of the specialized form, and by type casting whenever necessary. This method introduces little run-time overhead, but misses out on the optimizations possible in special cases. More importantly, the code size is not increased at all. For example, Java's `Vector<Integer>` is transformed to `Vector` containing `Object` class objects, and the compiler ensures that `Integer` objects are used as elements of the vector.

The Aldor Programming Language: Aldor was originally designed as an extension programming language for Axiom computer algebra system. It can be used more generally, but its affinity for computer algebra can be seen in its extensive algebra related libraries. One of the strengths of Aldor is that functions and types are first-class values. This way, functions and types can be computed and manipulated just as any other values. The type system in Aldor is organized on two levels, "*domains*" and "*categories*". Domains provide datatypes, and categories provide a sub-type lattice on the domains. More details about the Aldor programming language can be found elsewhere [1,2,3].

The Aldor Compiler: One of the main design goals of the Aldor programming language was to allow efficient implementation of symbolic and numeric mathematical programs. Aldor's compiler therefore implements many optimizations, including procedure inlining, data-structure elimination and control flow optimization. The Aldor compiler uses a platform-independent intermediate code, FOAM (First Order Abstract Machine), with efficient C and Lisp mappings.

After syntactic and semantic analysis, the compiler performs straightforward FOAM code generation, followed by a series of aggressive FOAM-to-FOAM optimizations, producing code that is close in performance to optimized C code.

3 Deeply Nested Type Constructions

We now turn our attention to the main problem, that of deeply nested type constructions. We define the *depth* of a type construction expression to be the number of generic constructors along the longest path from the root of the expression tree to any leaf. An example of a nested type construction is given by the following expression:

```
Set (Matrix (Poly (Complex (Fraction(Integer)))))
```

This expression forms the type of sets whose elements are matrices, the elements of which are polynomials with complex rational coefficients. This is a fully explicit nested type expression. Even though `Set`, `Matrix`, `Poly`, `Complex` and `Fraction` are parametric types, the resulting type is no longer parametric. This kind of fully explicit construction is not very common in Aldor algebra code. It is more usual to see constructions with a few levels, given explicitly, applied to a type parameter, and for this type parameter to be given a value at run time that is a construction of a few levels on another parameter, etc. The result is a deeply nested tower of constructions, built a few layers at a time. Similarly, with the expansion of macros and `typedef`s in C++, deeply nested type constructions frequently arise.

Continuing with our example above, let us imagine that one would call an operation from the `Set` domain to multiply each element by a constant. The set elements are matrices, which requires calling a function (to multiply a matrix by a constant) from the `Matrix` domain. Each element from the matrix is a polynomial which requires invoking the constant multiplication operation from the `Poly` domain, and so on. This operation requires many call frame activations and de-activations. This introduces an overhead that can be avoided by specializing the domain. After specializing the operations of the domain, it is usually possible to optimize the resulting operation further, starting with procedure inlining.

It is often the case that the leaves of type expressions are given by parameters, for example as with the parameter `R` in

```
MyConstruction(R: IntegralDomain): Algebra(R) == {
      s: Set (Matrix (Poly (Complex (Fraction(R))))) := ...
}
```

This leads us to consider the idea of constructing a specialized domain constructor:

$$\texttt{Set} \circ \texttt{Matrix} \circ \texttt{Poly} \circ \texttt{Complex} \circ \texttt{Fraction}$$

The functions from this domain constructor could be specialized using functions from the intermediate domain constructors, even if `R` is not known.

In Aldor, type constructors may themselves be dynamically bound so it is possible that we may not know one of the constructors in a deeply nested expression at compile-time. For example, we may have constructors such as:

$$\texttt{Set} \circ \texttt{Matrix} \circ \texttt{Poly} \circ \texttt{Complex} \circ X$$
$$X \circ \texttt{Matrix} \circ \texttt{Poly} \circ \texttt{Complex} \circ \texttt{Fraction}$$
$$\texttt{Set} \circ \texttt{Matrix} \circ X \circ \texttt{Complex} \circ \texttt{Fraction}$$

where X is an domain constructor unknown at compile-time. These cases must also be handled. For example, in the last line above, this means handling the specialized constructors $F = \texttt{Set} \circ \texttt{Matrix}$ and $G = \texttt{Complex} \circ \texttt{Fraction}$ to build $F \circ X \circ G$ at run-time.

4 Code Specialization

As mentioned previously, generic functions and type-constructing functions ("functors") in Aldor are implemented using the homogeneous approach. While this is very flexible, performance can suffer. The goal of present work was to use a mixture of homogeneous and heterogeneous implementations for deeply nested type expressions to improve efficiency. What makes this different from the analogous question in other programming languages is that in Aldor types are constructed dynamically and, as seen above, both the leaf types and the constructors themselves may be parameters unknown at compile time.

Some authors, e.g. [4], view partial evaluation as program specialization. Partial evaluators typically specialize whole programs rather than individual functions. For example, a partial evaluator may take program P and some of the inputs of that program, and produce a residual program R, that takes only the rest of the inputs of P and produces the same result as P, only faster. We take a similar approach for type specialization. We take a FOAM program in the representation used by the Aldor compiler and we specialize it according to some type instantiations.

In our case, we use a partial evaluator that not only specializes the domain constructor, but also specializes all the exports of that domain, effectively specializing the code of the type constructed by the functor. This creates operations of that specialized domain as monolithic operations that are more efficient. The overhead of the domain creation is more significant, but it does not happen very frequently. The main part of the speedup does not come from eliminating domain creation overhead. Rather it comes from from the optimization of the specialized functions.

Domain Specialization

1. Initialize the data structures.
2. Identify the domain declarations.
3. For each program,
   (a) If there is a domain constructor, generate a new specialized domain based on the domain constructor.
   (b) Replace the call to the generic domain constructor to a call to the specialized one.
   (c) In the specialized domain try to find the imports from other domains, and if found, modify the FOAM representation to help Aldor in-liner identify the called function.
4. Construct the FOAM code back from the data structures used by the tower optimizer.

Fig. 1. Algorithm to specialize domain code by FOAM-to-FOAM transformation

Our method tries to create specialized forms of the functions of instantiated domains. These are then used to construct the specialized run-time domains. The specialized functions should in-line code from domains that are type parameters. This way the resulting domains will have functions that do not require expensive function dispatch when executing the code from an instance domain.

Aldor domains are run-time entities, and are therefore not always completely defined at compile-time. In such cases, a full optimization is not possible. However, even in these cases parts of the type can be defined statically as in the examples given in Section 3. In these cases, a partial specialization is still possible. The algorithm used to transform the FOAM code is presented in Figure 1.

The FOAM code corresponding to Aldor source comprises two parts: the declaration of the data and the programs that manipulate that data. The optimization performs a single scan of all the programs found in FOAM and looks for functors. For each functor found, the type information for the original domain is retrieved as saved by the type analysis phase or it is reconstructed. Then the code for the type expression (i.e. the domain construction and all its operands) is cloned. Once all the operations of the original domain have been duplicated, the resulting cloned domain is updated by scanning all the operations for imports from the domains used as parameters. Once a target program has been found, the caller marks its call site with the address of the callee. This way the usual inliner can find the target function to be expanded locally. The final decision, whether the function should actually be expanded locally, remains with the general purpose inliner. This approach avoids expanding all the functions and it relies on the rules of the usual inliner to expand only the worthwhile cases.

In many cases not all parts of a type expression are known at compile time. In this situation, the above procedure is applied to the parts that are known. The specialized types will preserve the signature of all the exported symbols, so they can be used instead of the original call without affecting the rest of the caller's code. Our specialization is done in pairs of two starting from the innermost to the outermost domain constructor.

Preliminary results, presented in [6], showed that most of the speedup is obtained by specializing the innermost domains. For example, in case of a deeply nested type `Dom4(Dom3(Dom2(Dom1)))`, most of the speedup is obtained by specializing `Dom2(Dom1)`. On the other hand, specializing `Dom4(Dom3(X))` will not produce as significant a speedup.

This optimization is restricted to those cases where the type is fully or partly known at compile time. For those types that are completely dynamically constructed at run-time, as is the case with some functions able to generate new types at run-time, this transformation is not applicable and a dynamic optimizer must be used.

5 Data Specialization

Another important optimization that can be performed on opaque domains is data representation specialization. We have found this can have a very significant performance impact. We see this already in other environments: Even though parametric polymorphism has been introduced to Java and C#, their performance is still not as good as specialized code. We now describe our data specialization and how we measured the performance improvement.

While trying to measure the performance, we searched for benchmarks to measure generic code performance. We found only Stepanov's abstraction "penalty" benchmark [9], which is rather simple. To obtain meaningful measurements we transformed a well-known benchmark for scientific computing (SciMark) to use generic code (SciGMark) [7]. While constructing SciGMark we experimented with different types of specializations and discovered that most of the speedup between the hand specialized code and generic code was achieved when the data representation was changed from heap allocated to stack allocated structures.

In the process of implementing SciGMark, we noticed that a considerable speedup was obtained from data representation specialization. This transformation is possible if the data representation is private, because access to data is always through accessor functions. This way there is no risk of access to data through aliases. This is indeed the case with the representation of Aldor domains.

The Aldor compiler already offers an optimization for data structure elimination, which tries to flatten records, eventually moving heap allocation to the stack or registers if the objects do not escape local scope. A similar escape analysis for Java was presented by Choi [8]. Our proposed specialization goes a step further by eliminating data structures not only in each function, but also across operations that belong to a specialized domain.

The idea behind this optimization is to incorporate the data structure associated with the inner domain into the data structure of the outer domain. Since the data representation is private to the domain, this change will be invisible to its clients. The passing or conversion of data will be handled automatically by the operations of the transformed domain. The rest of the program can remain unchanged.

To illustrate how this works, we use a simple polynomial multiplication as example. One usual representation of a polynomial, using a dense representation,

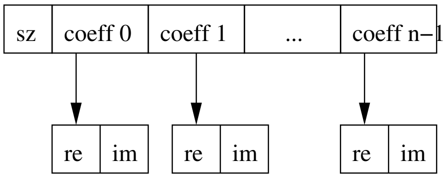


Fig. 2. Data rep. for polynomial with complex coefficients (before specialization.)

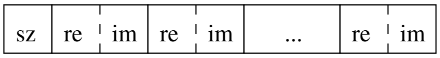


Fig. 3. Data rep. for polynomial with complex coefficients (after specialization.)

is as an array of coefficients values. A generic polynomial type constructor can accept different algebraic types for the coefficients. Suppose we use a complex number type as coefficients. In the same way, the complex type constructor can be generic, accepting a type for the real and imaginary parts. Suppose we take these to be integers. This is created in Aldor using `Poly(Complex(Integer))` and the data representation is an array similar to Figure 2. If the polynomial is specialized for such complex numbers, the resulting representation can be seen in Figure 3. This new representation has a much better memory locality, eliminates some indirections and more importantly, it eliminates the need to allocate and manage heap objects.

To illustrate how data representation specialization works, we created a small example that multiplies two polynomials that use a dense representation for the coefficients.

The implementation of `Ring` category and the `Complex` domain can be seen in Figure 4. The `Ring` category declares a type that has operations such as addition, multiplication, the ability to construct a neutral element and to print on screen. The `Complex` domain is an example of a domain that implements a `Ring` type, by providing implementations for all the operations declared in `Ring`.

In Figure 4, the Complex domain is not a parametric domain. In this case, the data representation used for the real and imaginary part is the domain `MachineInteger`. We take this simple approach for this example, because we shall perform the code and data specialization by hand, without any help from an automatic system. In this case the type of `Complex` is a `Ring` extended with some extra operations like `complex`, `real` and `imag`. The identifier `%` (percent) is the local name of the domain itself, The two macros `per` and `rep` are used to convert between the `%` type (the abstract type) and the `Rep` type (the internal representation). The rest of the code is self explanatory, and the implementation of the operations is a straight forward implementation of the definition of addition and multiplication for complex numbers.

The Aldor code for the corresponding `Poly` domain can be seen in Figure 5. Again, the `Poly` domain is a sub-type of `Ring`, by augmenting the `Ring` type with an operation to construct the object. This time the domain is parametric,

```
Ring: Category == with {
    +: (%, %) -> %;
    *: (%, %) -> %;
    0: %;
    <<: (TextWriter, %) -> TextWriter;
}
MI ==> MachineInteger;

Complex: Ring with
         { complex: (MI, MI) -> %; real: % -> MI; imag: % -> MI }
== add {
    Rep == Record(re: MI, im: MI);
    import from Rep;
    complex(r: MI, i: MI): % == per [r, i];
    real(t: %): MI == rep(t).re;
    imag(t: %): MI == rep(t).im;
    0: % == complex(0, 0);
    (a: %) + (b: %): % == complex(rep(a).re+rep(b).re,
                                  rep(a).im+rep(b).im);
    (a: %) * (b: %): % == {
        ra := rep.a; rb := rep.b;
        r := ra.re*rb.re-ra.im*rb.im; i := ra.re*rb.im+ra.im*rb.re;
        complex(r, i);
    }
    (w: TextWriter) << (t: %): TextWriter == w << rep.t.re << "+i*"
                     << rep.t.im
}
```

Fig. 4. Implementation of a simplified complex domain

requiring the type of the coefficients of the polynomial. The internal representation, given by `Rep`, is an `Array(C)`. Here, `Array` is another parametric domain that implements the generic collection similar to arrays in other programming languages. For brevity, we show here only the implementation of the addition between two polynomials, the multiplication and the display functions are similar. The Aldor compiler uses type inference to fill the type information when it is not provided. However, in some cases a disambiguating operator must be used, for example when we specified that `i` gets the value `0` (`0@MI`) from `MachineInteger` rather than the one from `C` or the zero provided by `%`.

An example of usage of the polynomial operations can be seen in Figure 6. The domain constructor creates two polynomials with all coefficients 0, of degree 9999 and then in multiplies them.

For the specialized version, the generic type `C` will be replaced with `Complex`, according to the instantiation given on the first line of Figure 6. The specialized implementation can be seen in Figure 7. One can note that parametric `Poly` domain has become `Poly__Complex`, the internal representation has changed to `TrailingArray`, and `Cross` is used instead of the `Record` for complex numbers. In Aldor, `Record` is a reference type and `Cross` is not, so we perform all data

```
Poly(C: Ring): Ring with { poly: Array(C) -> % } == add {
    Rep == Array(C);
    import from Rep;
    import from C;

    poly(size: MachineInteger): % == {
        res: Array(C) := new(size);
        i:=0@MI;
        while i < size repeat { res.i := 0@C; i := i + 1; }
        per res;
    }
    (a: %) + (b: %): % == {
        c: Array(C) := new(#rep(a));
        i := 0@MI;
        while i < #rep(a) repeat { c.i := rep(a).i + rep(b).i;
                                     i := i + 1; }
        per c;
    }
    0: % == poly(1);

    ...
}
```

Fig. 5. Implementation of a simplified generic polynomial type constructor

```
import from Poly(Complex);
sz == 10000; a := poly(sz); b := poly(sz); d := a * b;
```

Fig. 6. Polynomial multiplication

movement locally, without heap allocation. `TrailingArray` is another data type supported at the FOAM level. It models a record possibly with a fixed set of leading fields, followed by a set of fields that repeats any desired number of times. This can be used to convert an array of pointers to records to a single structure removing the indirections. The specialized case is presented in Figure 7. It uses both data specialization and code specialization specializations. The code specialization is done by creating a new domain `Poly__Complex`, copying the operations from `Complex` into `Poly__Complex`, and finally inlining the code of complex addition and multiplication into polynomial addition and multiplication.

Data specialization can not be performed in all the cases. If the size of one of the fields is not known it is not possible to inline the data. In some cases, it might be possible to rearrange the field order to keep the variable size structure at the end, but this would still not help once expanded in the outer domain.

The data specialization optimization is still a work in progress, but we have seen that it is possible to obtain some significant improvements as a result of application of this optimization on top of code specialization optimization that is already implemented.

```
Poly__Complex: Ring with { poly: MI  -> % } == add {
    T == Cross(re: MI, im: MI);
    Rep == TrailingArray(MI, (MI, MI));
    import from Rep;
    import from T;

    poly(size: MI): % == {
        i:MI := 1;
        res: TrailingArray(s:MI,(re:MI,im:MI)) := [size, size, (0,0)];
        while i <= size repeat { res(i, re) := 0; res(i, im) := 0;
                                          i := i + 1; }
        per res;
    }
    (a: %) + (b: %): % == {
        local ra: TrailingArray(s:MI,(re:MI,im:MI)) := rep(a);
        local rb: TrailingArray(s:MI,(re:MI,im:MI)) := rep(b);
        res: TrailingArray((s:MI),(re:MI,im:MI)) := [ra.s, ra.s, (0,0)];
        i:MI := 1;

        while i <= ra.s repeat {
            a1 := (ra(i,re), rb(i,re)); b1 := (ra(i,im), rb(i,im));
            aa: Cross(MI, MI) := a1;    bb: Cross(MI, MI) := b1;
            (ar, ai) := aa;             (br, bi) := bb;
            res(i, re) := ar+br;        res(i, im) := ai+bi;
            i := i + 1;
        }
        return per res;
    }
    0: % == poly(1);

    -- Brought from Complex
    local complex(r: MI, i: MI): T == (r, i);
    local a__C(a: T, b: T): T == {
        aa: Cross(MI, MI) := a; bb: Cross(MI, MI) := b;
        (ar, ai) := aa;         (br, bi) := bb;
        complex(ar+br, ai+bi);
    }
    ...
}
```

Fig. 7. Specialized polynomial representation

6 Results

We modified the Aldor compiler to perform the specialization of domain constructors and exported operations, as described. Table 1 presents the results of testing this optimization. The results vary from one to several times better. Tests one to three are only of depth one, and one can see there is no speedup between the regular Aldor optimizer and our proposed optimization. Tests four to seven

Table 1. Speedup obtained by automatically specializing the domains

Test	Original Time (s)	Optimized Time (s)	Ratio
Test1	87.13	86.24	1.01
Test2	35.66	35.55	1.00
Test3	35.27	35.27	1.00
Test4	37.71	0.17	∞
Test5	157.78	151.69	1.04
Test6	6.32	0.02	∞
Test7	12.92	1.54	8.39

Table 2. Time and run-time memory improvement after hand specialization of polynomial multiplication

Test	Original	Optimized	Ratio
Time (s)	119.19	7.98	14.94
Space (MB)	79.6	3.6	22.11

use deeply nested types made out of domains that contain simple functions. In test number four, the difference is big because the regular optimizer does not optimize at all and only our optimization is used. In test five, a simple tower type is used and thus is also optimized by the regular optimizer of the Aldor compiler, but there is still a 4% increase in speedup. Tests six and seven construct the same deeply nested types as in four and five, but they are not fully defined in one place, rather they are constructed in segments. The improvements for tests four and six are too large to measure. These are places were the regular optimizer was unable to optimize. The code specialization optimization does not modify the data representation therefore Table 1 does not mention memory usage.

The tests presented in Table 1 are simple functions that take full advantage of the inline expansion optimization. The next step is to see how this optimization performs on larger functions. An example of the application of this optimization together with the data specialization optimization can be seen in Figure 7.

The results of the specialization applied to the polynomial multiplication problem can be seen in Table 2. For the data representation optimization the creation of objects (mostly temporary objects resulted from arithmetic operations) on the heap is replaced by stack allocated objects and this should produce a decrease in memory usage.

All these tests were performed using Aldor compiler version 1.0.3. The backend C compiler used by the Aldor compiler was gcc 4.1.1. The CPU was a Pentium 4 3.2 GHz with 1MB cache and 2GB RAM. The actual hardware specification is not very important since we are only interested in relative values presented in the ratio columns.

7 Conclusions and Future Work

There are two principal strategies to optimize code: one is from the bottom, as with peep-hole optimization, and the other is from the top, as whole program optimization. When the program is taken as a whole and some properties can be inferred about the code that lead to some very effective optimizations. Program specialization techniques use the second approach to optimization. This second approach can provide significant improvements when it can be applied. The optimization proposed here for Aldor types are of this sort. They could also be applied to Java or C#, which also use a homogeneous approach to implement parametric polymorphism.

Our code and data specialization optimizations could be very well implemented by transforming Aldor source code directly. We chose to use the intermediate representation to take advantage of the existing infrastructure. We note that with nested types the code specialization optimization alone might not bring much improvement. However, with the help of data representation specialization and a nice data structure that allows specialization, as was the case with the polynomial domain, the code can become an order of magnitude faster, even on shallow types.

We have found these results to be sufficiently encouraging that we believe it would be of value to integrate the data representation optimization into the Aldor compiler and to test the compiler using a wider range of real algorithms used in scientific computation.

References

1. Watt, S.M., Broadbery, P.A., Dooley, S.S., Iglio, P., Steinbach, J.M., Sutor, R.S.: A First Report on the $A^{\#}$ Compiler. In: Proc. ISSAC 1994, pp. 25–31. ACM Press, New York (1994)
2. Aldor User Guide (2000), `http://www.aldor.org/`
3. Watt, S.M.: Aldor. In: Grabmeier, J., Kaltofen, E., Weispfenning, V. (eds.) Handbook of Computer Algebra, pp. 265–270. Springer, Heidelberg (2003)
4. Jones, N., Gomard, C., Sestoft, P.: Partial Evaluation And Automatic Program Generation. Prentice Hall, Englewood Cliffs (1993), ISBN 0-13-020249-5
5. Watt, S.M., Broadbery, P.A., Iglio, P., Morrison, S.C., Steinbach, J.M.: FOAM: First Oder Abstract Machine, `http://www.aldor.org`
6. Dragan, L., Watt, S.M.: Parametric Polymorphism Optimization for Deeply Nested Types in Computer Algebra. In: Maple Summer Workshop, Waterloo, Canada, pp. 243–259 (2005), ISBN 1-89451-185-9
7. Dragan, L., Watt, S.M.: Performance Analysis of Generics in Scientific Computing. In: Proceedings of Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania, pp. 90–100 (2005), ISBN 0-7695-2453-2
8. Choi, J.-D., Gupta, M., Serrano, M., Shreedhar, V.C., Midkiff, S.: Escape Analysis for Java. In: Proc. OOPSLA 1999, pp. 1–19. ACM Press, New York (1999)
9. Stepanov, A.A.: Appendix D.3 of Technical Report on C++ Performance, ISO/IEC PDTR 18015 (2003)

An Algebraic Implicitization and Specialization of Minimum KL-Divergence Models

Ambedkar Dukkipati[1] and Joel George Manathara[2]

[1] Department of Computer Science and Automation
Indian Institute of Science, Bangalore 560012, India
ambedkar@csa.iisc.ernet.in
http://www.csa.iisc.ernet.in/~ambedkar/
[2] Department of Aerospace Engineering
Indian Institute of Science, Bangalore 560012, India
joel@aero.iisc.ernet.in

Abstract. In this paper we study representation of KL-divergence minimization, in the cases where integer sufficient statistics exists, using tools from polynomial algebra. We show that the estimation of parametric statistical models in this case can be transformed to solving a system of polynomial equations. In particular, we also study the case of Kullback-Csisźar iteration scheme. We present implicit descriptions of these models and show that implicitization preserves specialization of prior distribution. This result leads us to a Gröbner bases method to compute an implicit representation of minimum KL-divergence models.

Keywords: Gröbner Bases, statistical models, elimination.

1 Introduction

The following Kullback minimum discrimination theorem [1] establishes important connections between statistics and information theory.

Theorem 1. *Given a probability space,* $(X, \mathfrak{M}, R)$*, define a probability measure* P *as*

$$P(A) = Z^{-1} \int_A \exp(T)\,\mathrm{d}R \ , \quad \forall A \in \mathfrak{M} \tag{1}$$

where T *a real valued function on* X *such that* $Z = \mathrm{E}_{[R]} \exp(T) < \infty$*. Suppose* T *is* P *integrable then*

$$I(P' \| R) \geq I(P \| R) = \mathrm{E}_{[P]} T - \ln Z \ ,$$

where $I(P\|R)$ *is known as Kullback-Leibler divergence (KL divergence) defined as*

$$I(P\|R) = \begin{cases} \displaystyle\int_X \ln \frac{\mathrm{d}P}{\mathrm{d}R}\,\mathrm{d}P & \text{if} \quad P \ll R \ , \\ +\infty & \text{otherwise.} \end{cases} \tag{2}$$

This result also leads to Kullback minimum discrimination principle. Suppose that the only information we know about a random variable are its prior estimate and some constraints in terms of expected values. The principle tells us to choose the distribution which minimizes the KL-divergence with prior estimate satisfying given constraints. The resulting models are known as minimum KL-divergence models.

Considering the well established role of information theory in statistics *cf.* [1, 2], this paper attempts to treat minimum KL-divergence models with the algebraic formalisms that has been recently studied in algebraic statistics.

The use of computational algebra and algebraic geometry in statistics was initiated in the work of Diaconis and Sturmfels [3] on exact hypothesis tests of conditional independence in contingency tables, and in the work of Pistone *et al.* [4] in experimental design. The term 'Algebraic Statistics' was first coined in the monograph by Pistone *et al.* [4] and appeared recently in the title of a book by Pachter and Sturmfels [5].

To extract the underlying algebraic structures in discrete statistical models, algebraic statistics treats statistical models as algebraic varieties. Here, parametric statistical models are described in terms of a polynomial (or rational) mapping from a set of parameters to distributions. One can show that many statistical models, for example independence models, Bernoulli random variables etc., can be given this algebraic formulation, and such models are referred to as algebraic statistical models [5].

We organize our paper as follows. In § 2 we discuss KL-divergence models and show that the estimation of these models can be transformed to solving a set of polynomial equations. We present our main result, implicit descriptions of minimum KL-divergence models and specializations, in § 3. We demonstrate these results with some examples in § 4 and make concluding remarks in § 5.

2 Minimum Divergence Distributions and Multivariate Polynomial Equations

2.1 Minimum KL-Divergence Models

Let Y be a discrete random variable taking finitely many values from the set $[m] = \{1, 2, \ldots m\}$. A probability distribution p of Y is naturally represented as a vector $p = (p_1, \ldots, p_m) \in \mathbb{R}^m$ if we fix the order on $[m]$. The set of all probability density functions (pdfs) of Y w.r.t counting measure on $[m]$ (we refer to such a pdf as probability mass functions or pmf) is called the probability simplex

$$\Delta_{m-1} = \{p = (p_1, \ldots, p_m) \in \mathbb{R}^m_{\geq 0} : \sum_{i=1}^{m} p_i = 1\} \ . \tag{3}$$

Suppose that the only information available about the pmf $p = (p_1, \ldots, p_m)$, of Y are the prior estimate $r = (r_1, \ldots, r_m) \in \Delta_{m-1}$ and observations in the form of expected values of the functions $t_i : [m] \to \mathbb{R}$, $i = 1, \ldots, d$ (sufficient statistic). We therefore have

$$\sum_{j=1}^{m} t_i(j) p_j = T_i \ \ , i = 1, \dots d, \tag{4}$$

where T_i, $i = 1, \dots, d$, are assumed to be known.

In information theoretic approach to statistics, known as Kullback's Minimum I-divergence principle (generalization of the maximum entropy principle that considers the cases where a prior estimate of the distribution p is available), one would choose the pdf $p \in \Delta_{m-1}$ that minimizes the Kullback-Leibler divergence (discrete version of (2))

$$I(p \| r) = \sum_{j=1}^{m} p_j \ln \frac{p_j}{r_j} \tag{5}$$

with respect to the constraints (4).

The set

$$\left\{ p \in \Delta_{m-1} : \sum_{j=1}^{m} t_i(j) p_j = T_i \ \ , i = 1, \dots, d \right\} \ ,$$

if non-empty, is called a *linear family* of probability distributions. The corresponding Lagrangian $\Xi(p, \xi)$ can be written as

$$-I(p \| r) - \xi_0 \left(\sum_{j=1}^{m} p_j - 1 \right) - \sum_{i=1}^{d} \xi_i \left(\sum_{j=1}^{m} t_i(j) p_j - T_i \right) \ .$$

Holding $\xi = (\xi_1, \dots, \xi_d)$ fixed, the unconstrained maximum of Lagrangian $\Xi(p, \xi)$ over all $p \in \Delta_{m-1}$ is given by an exponential family

$$p_j(\xi) = Z^{-1}(\xi) r_j \exp \left(-\sum_{i=1}^{d} \xi_i t_i(j) \right) \ , j = 1, \dots, m, \tag{6}$$

where $Z(\xi)$ is a normalizing constant (or partition function) given by

$$Z(\xi) = \sum_{j=1}^{m} r_j \exp \left(-\sum_{i=1}^{d} \xi_i t_i(j) \right) \ . \tag{7}$$

This model is an exponential family and is known as minimum KL-divergence model.

One can show that the Lagrange parameters in (6) can be estimated, when the values of T_i, $i = 1, \dots, d$ are available, by solving the set of partial differential equations [6]

$$\frac{\partial}{\partial \xi_i} \ln Z(\xi) = T_i \ \ , i = 1, \dots, d, \tag{8}$$

which has no explicit analytical solution. One could employ Newton-Raphson procedures to solve (8) or Darroch and Ratcliff's generalized iterative scaling algorithm [7].

As shown in [8], one can see that estimating minimum divergence distributions can be translated to solving polynomial equations, when the feature functions are nonnegative and integer valued. The polynomial system one would solve in this case is

$$\sum_{j=1}^{m} r_j (t_\ell(j) - T_\ell) \prod_{i=1}^{d} \theta_i^{t_i(j)} = 0 \ , \quad \ell = 1, \ldots, d. \tag{9}$$

Hence we have the following proposition.

Proposition 1. *The estimation of minimum KL-divergence model (6) given the information in the form of (4) amounts to solving a set of polynomial equations provided that the sufficient statistic t_i, $i = 1, \ldots, d$ are nonnegative and integer valued.*

2.2 Kullback-Csisźar Scheme

Estimating minimum divergence distributions involves simultaneously solving a system of nonlinear equations which is computationally inefficient. Therefore, one can employ the Kullback-Csiszár scheme where one would estimate the distribution considering only one constraint at a time. We describe this procedure as follows.

At the N^{th} iteration, the algorithm computes the distribution $p^{(N)}$ that minimizes $I(p^{(N)} \| p^{(N-1)})$ with respect to the i^{th} constraint, if $N = nd + i$ $(1 \leq i \leq d)$ for any positive integer n. This iteration will ultimately converge to the minimum divergence distribution, which is demonstrated for discrete distributions by [9], and for continuous distributions by [10]. A general and rigorous treatment of convergence, existence, and uniqueness analysis is given in [11]. (We refer to this procedure as Kullback-Csiszár scheme.)

In this iterative procedure, we have $p^{(0)} = r$ and $p^{(1)}$ is given by

$$p_j^{(1)} = r_j \left(Z^{(1)} \right)^{-1} \zeta_1^{t_1(j)},$$

where $Z^{(1)} = \sum_{j=1}^{m} r_j \zeta_1^{t_1(j)}$. The first constraint in (4) can be estimated by solving polynomial equation

$$\sum_{j=1}^{m} r_j (t_1(j) - T_1) \zeta_1^{t_1(j)} = 0 \ , \tag{10}$$

with indeterminate ζ_1. Similarly, we have

$$p_j^{(2)} = r_j \left(Z^{(1)} \right)^{-1} \left(Z^{(2)} \right)^{-1} \zeta_1^{t_1(j)} \zeta_2^{t_2(j)} \ ,$$

where $Z^{(2)} = \sum_{j=1}^{m} \zeta_2^{t_2(j)}$. Considering the first two constrains in (4), the minimum divergence distribution can be estimated by solving

$$\sum_{j=1}^{m} r_j (t_2(j) - T_2) \zeta_1^{t_1(j)} \zeta_2^{t_2(j)} = 0 \ , \tag{11}$$

along with (10).

In general, when $N = nd + i$ for some positive integer n, $p_j^{(N)}$ is given by

$$p_j^{(N)} = r_j \left(Z^{(1)}\right)^{-1} \dots \left(Z^{(N)}\right)^{-1} \zeta_1^{t_1(j)} \dots \zeta_N^{t_i(j)} \ , \quad N = 1, 2 \dots$$

and is determined by the following system of polynomial equations

$$\left.\begin{aligned} \sum_{j=1}^{m} r_j (t_1(j) - T_1) \zeta_1^{t_1(j)} &= 0 \ , \\ \sum_{j=1}^{m} r_j (t_2(j) - T_2) \zeta_1^{t_1(j)} \zeta_2^{t_2(j)} &= 0 \ , \\ &\vdots \\ \sum_{j=1}^{m} r_j (t_i(j) - T_i) \zeta_1^{t_1(j)} \zeta_2^{t_2(j)} \dots \zeta_N^{t_i(j)} &= 0 \ . \end{aligned}\right\} \tag{12}$$

Note that this system has a triangular structure which makes it easy to solve. Also, one can observe that the technique of Kullback-Csiszár iteration reflects in the system of polynomial equations which determines the minimum divergence distribution.

In this paper, we show that one could alternatively give an 'implicit' representation of minimum KL-divergence models (polynomial equations involving only $p_1, \dots, p_m$).

3 Embedding Minimum KL Divergence Models in Algebraic Varieties

3.1 Notation and Definitions

Throughout this paper, k represents a field (*e.g.*, $\mathbb{R}$, $\mathbb{C}$). The algebraic closure of the field k is represented by $\bar{k}$ (the algebraic closure of $\mathbb{R}$ is $\mathbb{C}$). From now on, k represents the field $\mathbb{R}$ and $\bar{k}$ represents $\mathbb{C}$.

Set of all monomials in indeterminates $x_1, \dots, x_n$ is denoted by $\mathbb{Z}^n_{\geq 0}$ (since set of all monomials is in one-to-one correspondence with $\mathbb{Z}^n_{\geq 0}$) and set of all polynomials in indeterminates $x_1, \dots, x_n$ with coefficients in k is denoted by $k[x_1, \dots, x_n]$. Let $f_1, \dots, f_s \in k[x_1, \dots, x_n]$. We use the notation $\mathcal{V}(f_1, \dots, f_s) = V$ to represent the varieties, where

$$V = \{(c_1, \dots c_n) \in k^n : f_i(c_1, \dots c_n) = 0, 1 \leq i \leq s\} \ .$$

V is uniquely determined by the ideal generated by $f_1, \dots, f_s$. This ideal is denoted by $\langle f_1, \dots, f_s \rangle$ and hence we have

$$\mathcal{V}(f_1, \dots, f_s) = \mathcal{V}(\langle f_1, \dots, f_s \rangle) \ .$$

In this paper, we use Gröbner bases in the methods we propose. The basic idea behind Gröbner bases is generalization of the division algorithm in single variable case ($k[x]$) to the multivariate case ($k[x_1, \ldots, x_n]$). To define Gröbner bases, we need the notion of monomial order.

Definition 1. *A* monomial order *or* term order *on $k[x_1, \ldots, x_n]$ is a relation $\prec$ (we use $\succ$ for the corresponding 'greater than') on $\mathbb{Z}^n_{\geq 0}$ that satisfies following conditions (i) $\prec$ is a total (or linear) ordering on $\mathbb{Z}^n_{\geq 0}$, (ii) if $\alpha \prec \beta$, for $\alpha, \beta \in \mathbb{Z}^n_{\geq 0}$ then for any $\gamma \in \mathbb{Z}^n_{\geq 0}$ it holds $\alpha + \gamma \prec \beta + \gamma$, and (iii) $\prec$ is a well-ordering on $\mathbb{Z}^n_{\geq 0}$.*

Given such ordering $\prec$, one can define the leading term of non-zero polynomial $f \in k[x_1, \ldots, x_n]$ as a term of f (the coefficient times its monomial) whose monomial is maximal for $\prec$. We denote this leading term by $\mathrm{LT}_\prec(f)$ and the corresponding monomial is denoted by $\mathrm{LM}_\prec(f)$.

Definition 2. *An ideal $\mathfrak{a} \subset k[x_1, \ldots, x_n]$ is said to be a monomial ideal if there is a set $A \subset \mathbb{Z}^n_{\geq 0}$, possibly infinite, such that $\mathfrak{a} = \langle x^\alpha : \alpha \in A \rangle$.*

Given any ideal $\mathfrak{a} \subset k[x_1, \ldots, x_n]$, the ideal defined as $\langle \mathrm{LM}_\prec(f) : f \in \mathfrak{a} \rangle$ is a monomial ideal and is denoted by $\mathrm{LM}_\prec(\mathfrak{a})$, which is known as leading monomial ideal of $\mathfrak{a}$. By Dickson's lemma [12, p. 69], the ideal $\mathrm{LT}_\prec(\mathfrak{a})$ is generated by a finite set of monomials. Dickson's lemma and the multivariate division algorithm leads to a proof of Hilbert bases theorem which states that every polynomial ideal can be finitely generated, which further lead to a definition of Gröbner basis [12, § 2.5].

Definition 3. *Fix a monomial order $\prec$ on $k[x_1, \ldots, x_n]$. A finite subset $G = \{g_1, \ldots, g_t\}$ of an ideal $\mathfrak{a} \subset k[x_1, \ldots, x_n]$ is a Gröbner basis if and only if*

$$\mathrm{LT}_\prec(\mathfrak{a}) = \langle \mathrm{LT}_\prec(g_1), \ldots, \mathrm{LT}_\prec(g_t) \rangle.$$

One of the important results in Gröbner bases theory is the 'elimination theorem', which we use in the results presented in this paper. To state this theorem, we need the notion of elimination ideals.

Definition 4. *Given $\mathfrak{a} \subset k[x_1, \ldots, x_n]$, the l^{th} elimination ideal $\mathfrak{a}_l$ in the polynomial ring $k[x_{l+1}, \ldots, x_n]$ is defined as $\mathfrak{a}_l = \mathfrak{a} \cap k[x_{l+1}, \ldots, x_n]$.*

Now follows the statement of elimination theorem.

Theorem 2. *[13, p. 69] Let $\mathfrak{a} \subset k[x_1, \ldots, x_n]$ be an ideal and let $G \subset k[x_1, \ldots, x_n]$ be a Gröbner basis of $\mathfrak{a}$ with respect a term order $\{x_1, \ldots, x_l\} \succ \{x_{l+1}, \ldots, x_n\}$ (for example consider the term order $x_1 \succ \ldots \succ x_n$) for $0 \leq l \leq n$. Then the set $G_l = G \cap k[x_{l+1}, \ldots, x_n]$ is a Gröbner basis of l^{th} elimination ideal $\mathfrak{a}_l$.*

3.2 Embedding Minimum Divergence Model in an Algebraic Variety

Following equations (6) and (7), given a positive integer valued sufficient statistic $t_i, i = 1, \ldots, d$, we have minimum KL-divergence model as image of rational map

$$\begin{aligned} f : k^d &\to k^m - W \\ (\theta_1, \ldots, \theta_d) &\mapsto \left(\frac{r_1 \prod_{i=1}^d \theta_i^{t_i(1)}}{\sum_{j=1}^m r_j \prod_{i=1}^d \theta_i^{t_i(j)}}, \ldots, \frac{r_m \prod_{i=1}^d \theta_i^{t_i(m)}}{\sum_{j=1}^m r_j \prod_{i=1}^d \theta_i^{t_i(j)}} \right) \quad , \end{aligned} \tag{13}$$

where $W = \mathcal{V}(\sum_{j=1}^m r_j \prod_{i=1}^d \theta_i^{t_i(j)})$. When we specify $r \in \Delta_{m-1}$ the above mapping gives a parametrization in terms of θ_i, $i = 1, \ldots d$. Note that this is bigger parametrization as we allow negative probabilities. Also note that the image of a polynomial map (or rational map) need not be a variety. In this case, the usual technique employed is to take the Zariski closure (Zariski closure of a set $A \subset k^n$ is the smallest variety that contains A [13, §2.5]).

The following lemma will allow us to treat minimum KL-divergence model (6) as image of a polynomial function rather than image of rational function (13) (see [8] [14] for more discussion in the context of maximum entropy models).

Lemma 1. *Consider following polynomial map*

$$\begin{aligned} \tilde{f} : k^{d+1} &\to k^m \\ (\theta_0, \theta_1, \ldots, \theta_d) &\mapsto \left(r_1 \theta_0 \prod_{i=1}^d \theta_i^{t_i(1)}, \ldots, r_m \theta_0 \prod_{i=1}^d \theta_i^{t_i(m)} \right) \quad . \end{aligned} \tag{14}$$

Then

$$\operatorname{im}(f) = \operatorname{im}(\tilde{f}) \cap \Delta_{m-1} \quad . \tag{15}$$

Proof. Clearly we have $\operatorname{im}(f) \subseteq \operatorname{im}(\tilde{f}) \cap \Delta_{m-1}$. Let $(a_1, \ldots, a_m) \in \operatorname{im}(\tilde{f}) \cap \Delta_{m-1}$. Then $\exists (b_0, b_1, \ldots, b_d) \in k^{d+1}$ such that

$$a_j = r_j b_0 \prod_{i=1}^d b_i^{t_i(j)} \quad , j = 1, \ldots, m.$$

Since $(a_1, \ldots, a_m) \in \Delta_{m-1}$, $\sum_{j=1}^m a_j = 1$. This implies

$$b_0{}^{-1} = \sum_{j=1}^m r_j \prod_{i=1}^d b_i^{t_i(j)} \quad .$$

Hence $(a_1, \ldots, a_m) \in \operatorname{im}(f)$ and $\operatorname{im}(f) \supseteq \operatorname{im}(\tilde{f}) \cap \Delta_{m-1}$. □

Note that (14) represents unnormalized KL-divergence model. Now, by specializing prior distribution $r = (r_1, \ldots, r_m)$ with real values $c = (c_1, \ldots, c_m) \in \mathbb{R}^m$, we get the model (13) as the image of the function

$$\begin{aligned} f_{r \to c} : k^d &\to k^m - W \\ (\theta_1, \ldots, \theta_d) &\mapsto \left(\frac{c_1 \prod_{i=1}^d \theta_i^{t_i(1)}}{\sum_{j=1}^m c_j \prod_{i=1}^d \theta_i^{t_i(j)}}, \ldots, \frac{c_m \prod_{i=1}^d \theta_i^{t_i(m)}}{\sum_{j=1}^m c_j \prod_{i=1}^d \theta_i^{t_i(j)}} \right) \quad , \end{aligned} \tag{16}$$

where $W = \mathcal{V}(\sum_{j=1}^{m} c_j \prod_{i=1}^{d} \theta_i^{t_i(j)})$. Hence one can use the result from [14], to deduce the implicitization as follows.

Theorem 3. *Let $f_{r\to c}$ be the rational function defined as above that parametrizes a minimum KL-divergence model with respect to sufficient statistics $t_i : \mathbb{R} \to \mathbb{Z}_{\geq 0}$ according to (16). Then*

$$\mathrm{im}(f_{r\to c}) \subseteq \mathcal{V}(\ker(\tilde{f}^*_{r\to c})) \cap \mathcal{V}(\sum_{j=1}^{m} p_j - 1) \ , \tag{17}$$

where $\tilde{f}^*_{r\to c}$ is a k-algebra homomorphism

$$\begin{aligned} \tilde{f}^*_{r\to c} : k[p_1, \ldots, p_m] &\to k[\theta_0, \theta_1, \ldots, \theta_d] \\ p_j &\mapsto c_j \theta_0 \prod_{i=1}^{d} \theta_i^{t_i(j)} \ . \end{aligned} \tag{18}$$

Further, given $r \in \Delta_{m-1}$, $\mathcal{V}(\ker(\tilde{f}^*_{r\to c})) \cap \mathcal{V}(\sum_{j=1}^{m} p_j - 1)$ is the smallest variety that contains the minimum divergence model.

We now show that specialization preserves the implicitization. This gives rise to a Gröbner basis method to compute the implicit model for arbitrary specializations. Before we proceed to this result we define following substitution homomorphism.

$$\begin{aligned} \eta : k[p_1, \ldots, p_m, r_1, \ldots, r_m, \theta_1, \ldots, \theta_d] &\to k[p_1, \ldots, p_m, \theta_1, \ldots, \theta_d] \\ p_j &\mapsto p_j, \quad j = 1, \ldots, m \\ \theta_i &\mapsto \theta_i, \quad i = 1, \ldots, d \\ r_j &\mapsto c_j, \quad j = 1, \ldots, m \end{aligned} \tag{19}$$

where $c_j \in \mathbb{R}, j = 1, \ldots, m$. It is easy to verify that η is a surjective homomorphism.

Proposition 2. *Consider the rational function (16) that parametrized unnormalized minimum KL-divergence model with respect to sufficient statistics $t_i : \mathbb{R} \to \mathbb{Z}_{\geq 0}$. Then*

$$\mathrm{im}(f_{r\to c}) \subseteq \mathcal{V}\left(\eta(\mathfrak{a} \cap k[p_1, \ldots, p_m, r_1, \ldots, r_m])\right) \cap \mathcal{V}(\sum_{j=1}^{m} p_j - 1) \tag{20}$$

where

$$\mathfrak{a} = \left\langle p_j - r_j \theta_0 \prod_{i=1}^{d} \theta_i^{t_i(j)} : j = 1, \ldots, m \right\rangle \ .$$

Proof. By incorporating substitution homomorphism η (19) we can state Theorem 3 as

$$\mathrm{im}(f_{r\to c}) \subseteq \mathcal{V}(\ker(\eta \circ \tilde{f}^*)) \cap \mathcal{V}(\sum_{j=1}^{m} p_j - 1) \tag{21}$$

where $\tilde{f}^*$ is a k-algebra homomorphism defined as

$$\begin{aligned} \tilde{f}^* : k[p_1, \dots, p_m] &\to k[r_1, \dots, r_m, \theta_0, \theta_1 \dots, \theta_d] \\ p_j &\mapsto r_j \theta_0 \prod_{i=1}^{d} \theta_i^{t_i(j)} \ . \end{aligned} \tag{22}$$

This implies that we only have to show that

$$\ker(\eta \circ \tilde{f}^*) = \eta(\mathfrak{a} \cap k[p_1, \dots, p_m, r_1, \dots, r_m]) \ . \tag{23}$$

Suppose

$$f_j = r_j \theta_0 \prod_{i=1}^{d} \theta_i^{t_i(j)} \ , j = 1, \dots, m$$

We have

$$\eta(f_j) = \eta(r_j) \theta_0 \prod_{i=1}^{d} \theta_i^{t_i(j)} \ , j = 1, \dots, m$$

$$\begin{aligned} \ker\left(\eta \circ \tilde{f}^*\right) &= \langle \eta(p_j - f_j) : \ j = 1, \dots, m \rangle \cap k[p_1, \dots, p_m] \\ &= \eta\left(\langle (p_j - f_j) : \ \ j = 1, \dots, m \rangle\right) \cap k[p_1, \dots, p_m] \\ &= \eta\left(\mathfrak{a}\right) \cap k[p_1, \dots, p_m] \\ &= \eta\left(\mathfrak{a} \cap k[p_1, \dots, p_m, r_1, \dots, r_m]\right) \end{aligned} \tag{24}$$

Note that the steps in (24) are consequence of the fact that η is a surjective homomorphism. □

By the elimination theorem (Theorem 2) we can find the generators for ideal $\ker\left(\eta \circ \tilde{f}^*\right)$ easily by first computing the Gröbner bases for $\mathfrak{a}$ and removing all the polynomials involving indeterminates $\theta_0, \theta_1, \dots, \theta_d$ and then specializing r by the substitution homomorphism η.

Corollary 1

$$\mathrm{im}(f_{r \to c}) \subseteq \mathcal{V}(\eta\left(G \cap k[p_1, \dots, p_m, r_1, \dots, r_m]\right)) \cap \mathcal{V}(\sum_{j=1}^{m} p_j - 1) \ , \tag{25}$$

where G is the Gröbner basis of

$$\mathfrak{a} = \left\langle p_j - r_j \theta_0 \prod_{i=1}^{d} \theta_i^{t_i(j)} : j = 1, \dots, m \right\rangle$$

with respect to a term order satisfying

$$\{p_1, \dots, p_m\} \prec \{r_1, \dots, r_m, \theta_0, \theta_1, \dots, \theta_d\} \ .$$

4 Example

Consider the following example with sufficient statistic

$$[t_i(j)]_{((i=1,2)\times(j=1,\ldots,7))} = \begin{pmatrix} 2\,1\,3\,1\,5\,2\,1 \\ 1\,2\,1\,4\,3\,3\,1 \end{pmatrix} \ . \tag{26}$$

The corresponding minimum KL-divergence model with prior distribution $r = (r_1, \ldots, r_m)$ can be written as

$$\begin{aligned}&(p_1, p_2, p_3, p_4, p_5, p_6, p_7) \\ &= \left(\frac{r_1\theta_1^2\theta_2}{Z}, \frac{r_2\theta_1\theta_2^2}{Z}, \frac{r_3\theta_1^3\theta_2}{Z}, \frac{r_4\theta_1\theta_2^4}{Z}, \frac{r_5\theta_1^5\theta_2^3}{Z}, \frac{r_6\theta_1^2\theta_2^3}{Z}, \frac{r_7\theta_1\theta_2}{Z} \right) \ ,\end{aligned} \tag{27}$$

where $Z = r_1\theta_1^2\theta_2 + r_2\theta_1\theta_2^2 + r_3\theta_1^3\theta_2 + r_4\theta_1\theta_2^4 + r_5\theta_1^5\theta_2^3 + r_6\theta_1^2\theta_2^3 + r_7\theta_1\theta_2$. The corresponding unnormalized model is given by the parametrization

$$\begin{aligned}&p_1 = r_1\theta_0\theta_1^2\theta_2 \;\; p_2 = r_2\theta_0\theta_1\theta_2^2 \;\; p_3 = r_3\theta_0\theta_1^3\theta_2 \;\; p_4 = r_4\theta_0\theta_1\theta_2^4 \\ &p_5 = r_5\theta_0\theta_1^5\theta_2^3 \;\; p_6 = r_6\theta_0\theta_1^2\theta_2^3 \;\; p_7 = r_7\theta_0\theta_1\theta_2\end{aligned}$$

Now let $\mathfrak{a}$ be the ideal generated by the polynomials

$$\begin{aligned}&p_1 - r_1\theta_0\theta_1^2\theta_2,\; p_2 - r_2\theta_0\theta_1\theta_2^2,\; p_3 - r_3\theta_0\theta_1^3\theta_2,\; p_4 - r_4\theta_0\theta_1\theta_2^4, \\ &p_5 - r_5\theta_0\theta_1^5\theta_2^3,\; p_6 - r_6\theta_0\theta_1^2\theta_2^3,\; p_7 - r_7\theta_0\theta_1\theta_2\end{aligned}$$

in $k[p_1, p_2, p_3, p_4, p_5, p_6, p_7, r_1, r_2, r_3, r_4, r_5, r_6, r_7, \theta_0, \theta_1, \theta_2]$. By using any software that supports algebraic geometric computations (we used *Mathematica* for the computations in this paper), the Gröbner basis, G, for the above ideal $\mathfrak{a}$ with the term order $p_1 \prec p_2 \prec p_3 \prec p_4 \prec p_5 \prec p_6 \prec p_7 \prec r_1 \prec r_2 \prec r_3 \prec r_4 \prec r_5 \prec r_6 \prec r_7 \prec \theta_0 \prec \theta_1 \prec \theta_2$ can be found.

From Corollary 1 the minimum KL-divergence model is contained in the variety defined by $\sum_{i=1}^{7} p_i - 1$ and $G \cap k[p_1, \ldots, p_m, r_1, \ldots, r_m]$ specialized under any substitution homomorphism η.

Often the prior distribution r is specified as an exponential distribution (which has only one parameter). That is, $r_j = (1/\alpha)\exp(-j/\alpha), j = 1, \ldots, m$. Using the transformation $1/\alpha = \ln\beta$, we have $r_j = (-\ln\beta)\,\beta^j, j = 1, \ldots, m$. In this model, the substitution homomorphism, η, can be specified by fixing the parameter β. However, to facilitate the computation of Gröbner basis, we introduce two parameters $a = -\ln\beta$ and $b = \beta$ which are fixed by specifying β. With the lex ordering $p_1 \prec p_2 \prec p_3 \prec p_4 \prec p_5 \prec p_6 \prec p_7 \prec a \prec b \prec \theta_0 \prec \theta_1 \prec \theta_2$, we calculate the Gröbner basis, G, of the ideal generated by polynomials

$$\begin{aligned}&p_1 - ab\theta_0\theta_1^2\theta_2,\; p_2 - ab^2\theta_0\theta_1\theta_2^2,\; p_3 - ab^3\theta_0\theta_1^3\theta_2,\; p_4 - ab^4\theta_0\theta_1\theta_2^4, \\ &p_5 - ab^5\theta_0\theta_1^5\theta_2^3,\; p_6 - ab^6\theta_0\theta_1^2\theta_2^3,\; p_7 - ab^7\theta_0\theta_1\theta_2\end{aligned}$$

For this example, $G \cap k[p_1, \ldots, p_m, a, b]$ is found as

$$\begin{array}{rrr}
p_1^2p_3p_4^2 - p_2^4p_5, & p_2^5p_3^6p_4 - p_1^8p_6^4, & p_2p_3^7p_4^3 - p_1^6p_5p_6^4, \\
-p_3^8p_4^5 + p_1^4p_2^3p_5^2p_6^4, & -p_2^6p_3^3 + p_1^6p_4^2p_7, & p_1^2p_2p_6^4 - p_3^3p_4^3p_7, \\
-p_2^2p_3^4 + p_1^4p_5p_7, & p_1^4p_6^4 - p_2^3p_3^2p_4p_5p_7, & -p_3^5p_4^2 + p_1^2p_2^2p_5^2p_7, \\
p_2^7p_6^4 - p_1^4p_4^5p_7^2, & p_2^3p_3p_6^4 - p_1^2p_4^3p_5p_7^2, & p_3^2p_6^4 - p_2p_4p_5^2p_7^2, \\
p_2^4p_6^8 - p_3^2p_4^6p_5p_7^3, & p_1^2p_6^8 - p_3p_4^4p_5^2p_7^3, & -p_2^3p_6^{12} + p_4^7p_5^3p_7^5, \\
bp_2^9p_3^7 - p_1^{11}p_4p_6^3p_7, & -bp_2^3p_3^4p_4 + p_1^5p_6^3, & -bp_3^5p_4^3 + p_1^3p_2p_5p_6^3, \\
p_2^2p_3^2 - bp_1^3p_6, & -bp_2^4p_3p_6 + p_1^3p_4^2p_7, & -bp_3^2p_6 + p_1p_5p_7, \\
p_3^3p_4^2 - bp_1p_2^2p_5p_6, & bp_1p_2p_6^5 - p_3p_4^3p_5p_7^2, & -bp_2^5p_6^5 + p_1p_3^2p_4^5p_7^2, \\
bp_2^4p_6^9 - p_1p_4^6p_5^2p_7^4, & -bp_1p_3p_6^9 + p_4^4p_5^3p_7^4, & p_2^3p_6^3 - bp_1p_3p_4^3p_7, \\
-p_1p_6^3 + bp_2p_4p_5p_7, & -p_2p_3^4p_6^3 + bp_1^3p_4p_5^2p_7^2, & -p_3^5p_4p_6^3 + bp_1p_2^3p_5^3p_7^2, \\
p_2^2p_3p_6^7 - bp_1p_4^4p_5^2p_7^3, & b^2p_2^7p_3^5 - p_1^8p_4p_6^2p_7, & -b^2p_2p_3^2p_4 + p_1^2p_6^2, \\
-b^2p_3^3p_4^3 + p_2^3p_5p_6^2, & -b^2p_1^2p_6^2 + p_2^2p_5p_7, & b^2p_2^2p_6^2 - p_3p_4^2p_7, \\
b^2p_2p_3p_6^6 - p_4^3p_5^2p_7^3, & -p_2^5p_3p_6^2 + b^2p_1^4p_4^3p_7, & -p_2p_3^2p_6^2 + b^2p_1^2p_4p_5p_7, \\
-p_3^3p_4p_6^2 + b^2p_2^3p_5^2p_7, & p_2^2p_6^6 - b^2p_3p_4^4p_5p_7^2, & -p_2p_3p_6^{10} + b^2p_4^5p_5^3p_7^4, \\
b^3p_2^5p_3^3 - p_1^5p_4p_6p_7, & -b^3p_1p_4 + p_2p_6, & -b^3p_2^3p_5 + p_1p_3p_4p_6, \\
p_3^5p_4p_6 - b^3p_1^3p_2p_5^2p_7, & -b^4p_2^3p_3 + p_1^2p_4p_7, & b^4p_3^2p_4 - p_2p_5p_7, \\
p_3^3p_4 - b^4p_1^2p_2p_5, & -p_6^4 + b^4p_4^2p_5p_7, & b^5p_1p_2p_6 - p_3p_4p_7, \\
b^5p_1p_3p_6^5 - p_4^2p_5^2p_7^3, & p_3^3p_6^3 - b^5p_1p_2^2p_5^2p_7, & -b^6p_2^2p_5 + p_3p_6^2, \\
b^6p_2p_3p_6^2 - p_4p_5p_7^2, & b^7p_2^2p_3 - p_1p_6p_7, & b^7p_1^3p_5 - p_3^3p_6, \\
-b^8p_1^2 + p_3p_7, & b^9p_2^4p_6 - p_1p_4^2p_7^2, & b^9p_1p_3p_6 - p_5p_7^2, \\
b^{10}p_2p_3p_4 - p_6^2p_7, & -b^{12}p_2^3 + p_4p_7^2, & b^{15}p_1p_2^2 - p_6p_7^2
\end{array}$$

and one can replace the symbols a, b with $-\ln\beta$ and β. Note that substitution does not preserve the Gröbner property [15]. This set of polynomial equations together with $\sum_{i=1}^{7} p_i - 1$ gives implicit representation of the model in the context for any specialization of β.

5 Closing Remarks

Given a parametric statistical model, computing the implicit representation of the model allows one to solve the problem of determining whether a probability distribution belongs to the given model or not. In this paper we have given implicit description of minimum KL-divergence models. We also showed that specialization preserves the implicitization and hence one can compute Zariski closure of the model, which gives the implicitization, for arbitrary specialization of prior distribution.

Acknowledgments. Part of this research was conducted when first author was at EURANDOM, Eindhoven.

References

1. Kullback, S.: Information Theory and Statistics. Wiley, New York (1959)
2. Csiszár, I., Shields, P.: Information Theory and Statistics: A Tutorial. Foundations and Trends in Communications and Information Theory, vol. 1. Now Publications (2004)
3. Diaconis, P., Sturmfels, B.: Algebraic algorithms for sampling from conditional distributions. Annals of Statistics 26, 363–397 (1998)
4. Pistone, G., Riccomagno, E., Wynn, H.: Algebraic Statistics. In: Computational Commutative Algebra in Statistics. Chapman and Hall, New York (2001)
5. Pachter, L., Sturmfels, B.: Algebraic Statistics and Computational Biology. Cambridge University Press, Cambridge (2005)
6. Jaynes, E.T.: Prior probabilities. IEEE Transactions on Systems Science and Cybernetics sec-4(3), 227–241 (1968)
7. Darroch, J.N., Ratcliff, D.: Generalized iterative scaling for log-linear models. The Annals of Mathematical Statistics 43(5), 1470–1480 (1972)
8. Dukkipati, A.: On parametric and implicit algebraic descriptions of maximum entropy models. Technical Report 2008-035, EURANDOM, Eindhoven, The Netherlands (2008)
9. Ireland, C., Kullback, S.: Contingency tables with given marginals. Biometrika 55, 179–188 (1968)
10. Kullback, S.: Probability densities with given marginals. The Annals of Mathematical Statistics 39(4), 1236–1243 (1968)
11. Csiszár, I.: I-divergence of probability distributions and minimization problems. Ann. Prob. 3(1), 146–158 (1975)
12. Cox, D., Little, J., O'Shea, D.: Ideals, Varieties, and Algorithms, 2nd edn. Springer, New York (1991)
13. Adams, W.W., Loustaunau, P.: An Introduction to Gröbner Bases. Graduate Studies in Mathematics, vol. 3. American Mathematical Society, Providence (1994)
14. Dukkipati, A.: On embedding maximum entropy models in algebraic varieties by Gröbner bases methods. In: Proceedings of IEEE International Symposium on Information Theory (ISIT), pp. 1904–1908. IEEE Press, Los Alamitos (2009)
15. Gianni, P.: Properties of Gröbner bases under specialization. In: Davenport, J.H. (ed.) ISSAC 1987 and EUROCAL 1987. LNCS, vol. 378, pp. 293–297. Springer, Heidelberg (1989)

On Sufficient Conditions for Integrability of a Planar System of ODEs Near a Degenerate Stationary Point

Victor Edneral[1] and Valery G. Romanovski[2,3]

[1] Skobeltsyn Institute of Nuclear Physics
of Lomonosov Moscow State University
Leninskie Gory 1, Moscow, 119991, Russia
`edneral@theory.sinp.msu.ru`
[2] CAMTP - Center for Applied Mathematics and Theoretical Physics
University of Maribor, Krekova 2, Maribor SI-2000, Slovenia
[3] Faculty of Natural Science and Mathematics, University of Maribor
Koroška cesta 160, SI-2000 Maribor, Slovenia
`valery.romanovsky@uni-mb.si`

Abstract. We consider an autonomous system of ordinary differential equations, which is solved with respect to derivatives. To study the local integrability of the system near a degenerate stationary point we use an approach based on the Power Geometry Method and on the computation of resonant normal forms. For a planar system depending on five parameters we give four series of conditions on parameters of the system for which it is integrable near the degenerate stationary point.

Keywords: planar ordinary differential equations, integrability, resonant normal form, power geometry, computer algebra.

1 Introduction

We consider an autonomous system of ordinary differential equations

$$dx_i/dt \overset{\text{def}}{=} \dot{x}_i = \varphi_i(X), \quad i = 1, \ldots, n, \tag{1}$$

where $X = (x_1, \ldots, x_n) \in \mathbb{C}^n$ and $\varphi_i(X)$ are polynomials.

The system (1) is called integrable in a neighborhood U of the stationary point $X = X^0$ if it has in U the sufficient number m of first integrals of the form

$$a_j(X)/b_j(X), \quad j = 1, \ldots, m, \tag{2}$$

where functions $a_j(X)$ and $b_j(X)$ are analytic in U. Otherwise we call the system (1) *locally non-integrable* in this neighborhood. For integrability of a planar ($n = 2$) system it is sufficient to have a single first integral of the form (2).

In a neighborhood of the stationary point $X = 0$, the system (1) can be written in the form

$$\dot{X} = A\,X + \tilde{\Phi}(X), \tag{3}$$

where $\tilde{\Phi}(X)$ has no linear terms.

In [1,2] a method for the analysis of integrability of the system (3) based on power transformations and computation of normal forms near stationary solutions of transformed systems has been proposed.

Let $\lambda_1, \lambda_2, \ldots, \lambda_n$ be eigenvalues of the matrix A. If at least one of λ_i differs from 0, then the stationary point $X = 0$ is called an *elementary* stationary point. In this case, the system (3) has a normal form which is a system with a simpler structure. If all eigenvalues vanish, then the stationary point $X = 0$ is called a *non-elementary* stationary point. In this case, there is no normal form for the system (3). However using power transformations a non-elementary stationary point $X = 0$ can be blown up to a set of elementary stationary points. After that, it is possible to compute the normal forms and verify that the condition A [4] is satisfied at each elementary stationary point.

In [5,6], an example of a planar autonomous system of ordinary differential equations depending on five parameters was considered, and local integrability in a neighborhood of a degenerate stationary point was studied. For $b^2 \neq 2/3$, the authors found the complete set of necessary conditions on parameters of the system for which the system is locally integrable near the degenerate stationary point. The case $b^2 = 2/3$ should be treated separately.

In this paper we prove that the conditions mentioned above are also the sufficient conditions for integrability. We show that for each such domain there are local first integrals of motion. The study involves laborious computations which would be impossible to complete without the tools of computer algebra. To perform most of symbolic computations we use the computer algebra system MATHEMATICA, which is very efficient for simplification of algebraic and non-algebraic expressions, including those with special functions, and integration. However, MATHEMATICA is rather inefficient tool for solving systems of multivariate algebraic polynomials, so to find solutions of such systems we use the computer algebra system SINGULAR, which is a computer algebra system for polynomial computations, with special emphasis on commutative and non-commutative algebra, algebraic geometry, and singularity theory (it is free and open-source under the GNU General Public License developed at the Department of Mathematics of the University of Kaiserslautern).

2 The Studied Equation

We study the planar polynomial system

$$\begin{aligned} dx/dt &= -y^3 - b\,x^3y + a_0\,x^5 + a_1\,x^2y^2, \\ dy/dt &= c\,x^2y^2 + x^5 + b_0\,x^4y + b_1\,x\,y^3. \end{aligned} \tag{4}$$

Systems with a nilpotent matrix of the linear part were thoroughly studied by Lyapunov and many others (see e.g. [7,8] and references therein). There is no linear part in (4), i.e., it is degenerate at the stationary point $x = y = 0$, and its first approximation is not homogeneous. This is the simplest case of a planar system without the linear part and with Newton's open polygon [3] consisting of a single edge. In general case, such problems have not been studied. However, the system with such support was considered in [9], where the authors studied the Hamiltonian case.

2.1 The First Quasi-homogeneous Approximation

Let us consider the first quasi-homogeneous approximation of system (4), that is the system of the form

$$\dot{x} = -y^3 - b\,x^3\,y, \quad \dot{y} = c\,x^2\,y^2 + x^5. \tag{5}$$

We study the problem: what are the conditions on parameters of (4) under which the system has a local analytic first integral. As it was demonstrated in [5,6], (5) has an integral, but it can be not analytic.

In [5,6], the following statement has been proven.

Theorem 1. *In the case* $D \overset{\text{def}}{=} (3\,b + 2\,c)^2 - 24 \neq 0$*, the system* (5) *is locally integrable if and only if the number*

$$\varGamma = \frac{3\,b - 2\,c}{\sqrt{D}} \tag{6}$$

is rational.

The system with the same support was considered for the first time in [9], where the authors studied the case $3\,b - 2\,c = 0$. In this case, the order $\varGamma$ of (6) is equal to zero, and the integral of the system (5) is

$$I = 2\,x^6 + 4\,c\,x^3 y^2 + 3\,y^4. \tag{7}$$

The system in this case is Hamiltonian with the Hamiltonian function $H = I/12$. Furthermore, the authors of [9] studied the Hamiltonian case of the full system under the additional assumption that the Hamiltonian function can be written as a product of square-free factors.

In this paper, we study another simple particular case, when the order $\varGamma$ of (6) is equal to ± 1. In this case, $c = 1/b$ and $2\,x^3 + 3\,b\,y^2$ is the first integral of (5). The system in this case is written as

$$\begin{aligned} dx/dt &= -y^3 - b\,x^3 y + a_0\,x^5 + a_1\,x^2 y^2, \\ dy/dt &= 1/b\;x^2 y^2 + x^5 + b_0\,x^4 y + b_1\,x\,y^3, \end{aligned} \tag{8}$$

Like in Theorem 1, the first quasi-homogeneous approximation has an analytic integral but it is not a Hamiltonian system. The set of necessary conditions of local integrability of system (8) has been found in [5,6].

3 The Necessary Conditions of Local Integrability

If $n = 2$ then rationality of the ratio λ_1/λ_2 and the condition A of [4] are the necessary and sufficient conditions for local integrability of the system near the elementary stationary point.

After the power transformation

$$x = u\,v^2, \quad y = u\,v^3 \tag{9}$$

and time rescaling

$$dt = u^2 v^7 d\tau,$$

we obtain from (8) the system

$$\begin{array}{l} du/d\tau = -3\,u - (3\,b + 2/b)u^2 - 2\,u^3 + (3\,a_1 - 2\,b_1)u^2 v + (3\,a_0 - 2\,b_0)u^3 v, \\ dv/d\tau = v + (b + 1/b)u\,v + u^2 v + (b_1 - a_1)u\,v^2 + (b_0 - a_0)u^2 v^2. \end{array} \tag{10}$$

Under the power transformation (9) the point $x = y = 0$ blows up into two straight lines $u = 0$ and $v = 0$. Along the line $u = 0$ the system (10) has a single stationary point, $u = v = 0$. Along the second line, $v = 0$, the system has four elementary stationary points

$$u = 0, \quad u = -\frac{1}{b}, \quad u = -\frac{3b}{2}, \quad u = \infty. \tag{11}$$

We look for conditions of local integrability at all stationary points (11), then it will yield the conditions of local integrability of the system (8) near the point $x = y = 0$. We restrict our study to the case $b^2 \neq 2/3$ when the linear part of (10), after the shift $u = \tilde{u} - 1/b$, has non-vanishing eigenvalues. At $b^2 = 2/3$, the shifted system (10) in new variables $\tilde{u}$ and v has the linear part with the Jordan cell with both eigenvalues equal to zero. This case can be studied by making use of one more power transformation. Except of this point, where there can be additional domains of integrability, in [5,6] the complete set of necessary conditions on parameters of the system for which (8) is locally integrable near the degenerate stationary point has been found. It consists of the following four two-dimensional domains in the parameter space:

$$\begin{array}{l} 1)\ a_0 = 0, \quad a_1 = -b_0\,b, \quad b_1 = 0, \\ 2)\ b_1 = -2a_1,\ a_0 = a_1 b,\ b_0 = b_1 b, \\ 3)\ b_1 = 3a_1/2,\ a_0 = a_1 b,\ b_0 = b_1 b, \\ 4)\ b_1 = 8a_1/3,\ a_0 = a_1 b,\ b_0 = b_1 b. \end{array} \tag{12}$$

This result was obtained using the power transformation (9) and calculation of corresponding normal forms by means of the program described in [10].

4 The Sufficient Conditions of Integrability

4.1 The First Domain

If the first of conditions (12), that is, $a_0 = 0$, $a_1 = -b_0\,b$, $b_1 = 0$, is fulfilled it is possible to construct the first integral by the method of indeterminate coefficients. It means that we look for a first integral of (10) near the origin $u = 0$, $v = 0$ in the form of a polynomial (in our case, we took the polynomial of total degree 12) in variables u, v with indeterminate coefficients. We substitute this polynomial into the equation for the first integral I,

$$\frac{d\,I}{d\,t} = \frac{\partial\,I}{\partial\,u}\dot{u} + \frac{\partial\,I}{\partial\,v}\dot{v} = 0, \tag{13}$$

where $\dot{u}$ and $\dot{v}$ satisfy equations (10) with condition 1), that is,

$$\begin{aligned} du/d\tau &= -3\,u - (3\,b + 2/b)u^2 - 2\,u^3 - b_0\,(3\,b + 2\,u)u^2 v,\\ dv/d\tau &= v + (b + 1/b)u\,v + u^2 v + b_0\,(b + u)u\,v^2. \end{aligned}$$

We collect similar terms and equate to zero the coefficients of the monomials $u^k v^s$ to obtain a linear system of equations for the indeterminate coefficients. Solving the system with MATHEMATICA we find

$$I_1(u, v) = u^2(3\,b + 2\,u)v^6. \tag{14}$$

Using the inversion of the power transformation (9)

$$u = x^3/y^2, \quad v = y/x, \tag{15}$$

we obtain the first integral for the system (8)

$$I_1(x, y) = 2\,x^3 + 3\,b\,y^2. \tag{16}$$

There is no restriction $b^2 \neq 2/3$, and both integrals (14) and (16) are analytic.

4.2 The Second Domain

For the second condition from (12), that is, $b_1 = -2a_1$, $a_0 = a_1 b$, $b_0 = b_1 b$ the integrals can be found in the same way. The calculation of the first integrals yields

$$\begin{aligned} I_2(u, v) &= u^2\,v^6\,[3\,b + u\,(2 - 6\,a_1\,b\,v)],\\ I_2(x, y) &= 2\,x^3 - 6\,a_1\,b\,x^2\,y + 3\,b\,y^2. \end{aligned} \tag{17}$$

There is here also no restriction $b^2 \neq 2/3$ in (17), and the integrals are analytic.

4.3 The Third Domain

Under the third condition from (12), that is, $b_1 = 3a_1/2$, $a_0 = a_1 b$, $b_0 = b_1 b$ the system (10) has the form

$$\begin{aligned} du/d\tau &= -3\,u - (3\,b + 2/b + 2\,u)u,\\ dv/d\tau &= v + (b + 1/b)u\,v + u^2 v + \tfrac{a_1}{2}\,(1 + b\,u)u\,v^2. \end{aligned}$$

Dividing the second equation by the first one we obtain a Bernoulli differential equation, which can be integrated. The calculations yield the first integrals

$$\begin{aligned} I_3(u, v) &= \frac{4 - 4a_1\,u\,v + 3^{5/6} a_1\,{}_2F_1(2/3, 1/6; 5/3; -2u/(3b))\,u\,v\,(3 + 2u/b)^{1/6}}{u^{1/3} v\,(3b + 2u)^{1/6}},\\ I_3(x, y) &= \frac{a_1 x^2(-4 + 3^{5/6}\,{}_2F_1\big(2/3, 1/6; 5/3; -2\,x^3/(3\,b\,y^2)\big)\,(3 + 2x^3/(b\,y^2))^{1/6}) + 4y}{y^{4/3}(3\,b + 2\,x^3/y^2)^{1/6}}, \end{aligned} \tag{18}$$

where ${}_2F_1$ is the hypergeometric series [11].

We note that cubes of $I_3(u,v)$ and $I_3(x,y)$ are analytical integrals. Indeed, in accordance with paragraph 2.1.1 of [11] for the values of parameters of the hypergeometric series $I_3(u,v)$ in (18), the power series is absolutely convergent at $|2\,u/(3\,b)|=1$, so it is analytic at $u=0$. The situation with the series for $I_3(x,y)$ is similar. It is absolutely convergent at point $x=0, y\neq 0$. However, we need to study analytic properties at other points, where hypergeometric function (an analytic continuation of the corresponding series) can have singularity. For the parameters in (18) ${}_2F_1$ is proportional to the Legendre function (see paragraph 3.2, formulae (8) of [11])

$$ {}_2F_1\left(2/3,1/6;5/3;-2\,x^3/(3\,b\,y^2)\right) \sim (\tfrac{3\,b\,y^2}{2\,x^3})^{1/6}\ (\tfrac{3\,b\,y^2}{2\,x^3}+1)^{5/12}\ Q^{5/6}_{1/6}((-\tfrac{3\,b\,y^2}{2\,x^3})^{1/2}). $$

The Legendre function has 3 singular points: ± 1 and ∞. It is proportional for the current values of parameters (see paragraph 3.9.2 of [11])

$$ Q^{5/6}_{1/6}(z) \sim (z-1)^{-5/12}, \quad \text{near} \quad z=1, $$

$$ Q^{5/6}_{1/6}(x) \sim (x+1)^{-5/12}, \quad \text{near} \quad x=-1, $$

$$ Q^{5/6}_{1/6}(z) \sim z^{1/6}, \quad \text{near} \quad z\to\infty. $$

So, we see that in each case we can represent $I_3(x,y)$ as a rational power of a rational function of x,y. That is, there is a rational power of this function which satisfies condition (2).

4.4 The Fourth Domain

For the values of parameters corresponding to the last condition of (12), that is, $b_1=8a_1/3$, $a_0=a_1b$, $b_0=b_1b$, the system (10) has the form

$$ \begin{aligned} du/d\tau &= -u\,(9\,b+6\,u+7\,a_1\,b\,u\,v) = U(u,v),\\ dv/d\tau &= v\,(3\,b+3\,u+5\,a_1\,b\,u\,v) = V(u,v). \end{aligned} \tag{19} $$

We remind that by the definition the Darboux factor of the system

$$ dx/dt = P(x,y), \quad dy/dt = Q(x,y) \tag{20} $$

is a polynomial $f(x,y)$ such that

$$ \frac{\partial f}{\partial x}P + \frac{\partial f}{\partial y}Q = Kf, \tag{21} $$

with $K(x,y)$ being a polynomial. If P and Q are polynomials in x and y of degree at most n, then $K(x,y)$ is a polynomial of degree at most $n-1$ ($K_i(x,y)$

is called the cofactor). A simple computation shows that if there are Darboux factors $f_1, f_2, \ldots, f_k$ with the cofactors $K_1, K_2, \ldots, K_k$ satisfying

$$\sum_{i=1}^{k} \alpha_i K_i + P'_x + Q'_y = 0 \tag{22}$$

then the system admits the integrating factor

$$M = f_1^{\alpha_1} \cdots f_k^{\alpha_k} \tag{23}$$

(see, e.g., [12] for more details). It is well known that if a system (20) with a non-degenerate singular point at the origin has an integrating factor of the form $M = 1 + h.o.t$, then it also has an analytic first integral (see e.g. [7]).

It is obvious that $u = 0$ and $v = 0$ are the Darboux factors of system (19) with the cofactors

$$K_1 = -(9\,b + 6\,u + 7\,a_1\,b\,u\,v)$$

and

$$K_2 = (3\,b + 3\,u + 5\,a_1\,b\,u\,v),$$

respectively. We then try to find more Darboux factors using the method of indeterminate coefficients. We look for the factor f_3 in the form of the polynomial of degree 4 with indeterminate coefficients t_i and s_i, and for the cofactor K_3 in the form of the polynomial of degree 2 with indeterminate coefficients s_i. Substituting f_3 and K_3 into (21) and equating the coefficients of the same monomials $u^k v^m$ we obtain a system of polynomials in s_i and t_i. It turns out that the system is too difficult to solve with MATHEMATICA, however we can easily solve it using the package *primdec.lib* [13] of the computer algebra system SINGULAR [14]. Performing computations we find

$$f_3 = \frac{1}{9} {a_1}^2 u^2 v^2 + \frac{2 a_1 u v}{3} + \frac{2u}{3b} + 1$$

with the cofactor

$$K_3 = -2u(3 + 2a_1 b v).$$

Solving the equation

$$\alpha_1 K_1 + \alpha_2 K_2 + \alpha_3 K_3 + U'_u + V'_v = 0$$

we find

$$\alpha_1 = -\frac{4}{3}, \quad \alpha_2 = -2, \quad \alpha_3 = -7/6.$$

Therefore, the system admits the integrating factor

$$M(u, v) = \frac{1}{u^{4/3} v^2 [6\,u + b\,(3 + a_1\,u\,v)^2]^{7/6}}. \tag{24}$$

Using the factor we find the first integral of motion in the form

$$I_4(u,v) = \frac{u\,(3+2\,a_1^2 b u)+6\,a_1\,b\,v}{3\,u\,[u^3(6+a_1^2 b\,u)+6\,a_1^2 b\,u^2 v+9\,b\,v^2]^{1/6}} - \frac{8\,a_1\sqrt{-b}}{3^{5/3}} B_{6+a_1\,\sqrt{-6\,b\,u}+3\,v\,\sqrt{-6\,b/u^3}}(5/6, 5/6), \tag{25}$$

where $B_x(p,q)$ is an incomplete beta function [11],

$$B_x(p,q) = \int_0^x t^{p-1}(1-t)^{q-1}dt = \frac{x^p}{p}\,{}_2F_1(p, 1-q; p+1; x).$$

Above ${}_2F_1$ is a hypergeometric function.

By substitution (15) from $I_4(u,v)$ we can obtain the first integral $I_4(x,y)$ but the integrals are not analytic. However, by Theorem 4.13 of [15] if a system (20) has a Darboux integrating factor of the form

$$\mu = x^{\beta_1} y^{\beta_2} (1 + \text{h.o.t})^{\beta}$$

then it has an analytic first integral except of the case when both β_1 and β_2 are integer numbers greater than 1. Since for the factor (24) $\beta_1 = -4/3$ is not an integer, the theorem is applicable and yields the existence of an analytic first integral (however, we do not know its explicit form). The rational substitution (15) will not destroy the property of analyticity, so for the fourth domain there is the first analytic integral of the system (8).

5 Conclusions

To summarize, for planar non-Hamiltonian system (8) depending on five parameters with $b^2 \neq 2/3$ we have found a set of conditions on parameters at which the system (8) is integrable near the degenerate point $X = 0$. The integrable systems form four two-parametric domains in the space of the parameters of the system.

The investigation of the cases $b^2 = 2/3$, $c = 1/b$, and $c \neq 1/b$ but with a rational value of Γ of (6) is still an open problem.

Acknowledgements

The first author was supported by the the Russian Foundation for Basic Research (project 08-01-00082) and the grants of the President of the Russian Federation for Support of the Leading Scientific Schools NSh.-3159.2010.2 and NSh.-41422010.2. The second author acknowledges the support by the Slovenian Research Agency and by the Transnational Access Programme at RISC-Linz of the European Commission Framework 6 Programme for Integrated Infrastructures Initiatives under the project SCIEnce (contract no. 026133). The authors are also grateful to Professor A.D. Bruno for important discussions and useful advices and to the referees for their valuable remarks which helped to improve the manuscript.

References

1. Bruno, A.D., Edneral, V.F.: Algorithmic analysis of local integrability. Dokl. Akademii Nauk 424(3), 299–303 (2009) (Russian) = Doklady Mathem. 79(1), 48–52 (2009) (English)
2. Bruno, A.D.: Power Geometry in Algebraic and Differential Equations. Fizmatlit, Moscow (1998) (Russian) = Elsevier Science, Amsterdam (2000) (English)
3. Bruno, A.D.: Local Methods in Nonlinear Differential Equations. Nauka, Moscow (1979) (Russian) = Springer-Verlag, Berlin (1989) (English)
4. Bruno, A.D.: Analytical form of differential equations (I,II). Trudy Moskov. Mat. Obsc. 25, 119–262 (1971); 26, 199–239 (1972) (Russian) = Trans. Moscow Math. Soc. 25, 131–288 (1971); 26, 199–239 (1972) (English)
5. Bruno, A.D., Edneral, V.F.: On Integrability of a Planar System of ODE's near Degenerate Stationary Point. Zapiski Nauchnykh Seminarov POMI 373, 34–47 (2009) (Russian)
6. Bruno, A.D., Edneral, V.F.: On Integrability of a Planar ODE System of near Degenerate Stationary Point. In: Gerdt, V.P., Mayr, E.W., Vorozhtsov, E.V. (eds.) CASC 2009. LNCS, vol. 5743, pp. 45–53. Springer, Heidelberg (2009)
7. Amelkin, V.V., Lukashevich, N.A., Sadovskii, A.P.: Nonlinear Oscillations in Second Order Systems. BSU, Minsk (1982) (in Russian)
8. Liu, Y., Li, J.: New study on the center problem and bifurcations of limit cycles for the Lyapunov system (I). Internat. J. Bifur. Chaos Appl. Sci. Engrg. 19(11), 3791–3801 (2009)
9. Algaba, A., Gamero, E., Garcia, C.: The integrability problem for a class of planar systems. Nonlinearity 22, 395–420 (2009)
10. Edneral, V.F.: On algorithm of the normal form building. In: Ganzha, V.G., Mayr, E.W., Vorozhtsov, E.V. (eds.) CASC 2007. LNCS, vol. 4770, pp. 134–142. Springer, Heidelberg (2007)
11. Bateman, H., Erdêlyi, A.: Higer Transcendental Functions, vol. 1. McGraw-Hill Book Company, Inc., New York (1953)
12. Romanovski, V.G., Shafer, D.S.: The Center and Cyclicity Problems: A Computational Algebra Approach. Birkhüser, Boston (2009)
13. Decker, W., Pfister, G., Schönemann, H.A.: Singular 2.0 library for computing the primary decomposition and radical of ideals primdec.lib (2001)
14. Greuel, G.M., Pfister, G., Schönemann, H.: SINGULAR 3.0. A Computer Algebra System for Polynomial Computations. Centre for Computer Algebra, University of Kaiserslautern (2005), http://www.singular.uni-kl.de
15. Christopher, C., Mardešić, P., Rousseau, C.: Normalizable, integrable, and linearizable saddle points for complex quadratic systems in $\mathbf{C}^2$. J. Dyn. Control Sys. 9, 311–363 (2003)

Symbolic-Numeric Algorithms for Computer Analysis of Spheroidal Quantum Dot Models

A.A. Gusev[1,2], O. Chuluunbaatar[1], V.P. Gerdt[1], V.A. Rostovtsev[1,2], S.I. Vinitsky[1], V.L. Derbov[3], and V.V. Serov [3]

[1] Joint Institute for Nuclear Research, Dubna, Russia
[2] Dubna International University of Nature, Society & Man, Dubna, Russia
[3] Saratov State University, Saratov, Russia
gooseff@jinr.ru

Abstract. A computational scheme for solving elliptic boundary value problems with axially symmetric confining potentials using different sets of one-parameter basis functions is presented. The efficiency of the proposed symbolic-numerical algorithms implemented in Maple is shown by examples of spheroidal quantum dot models, for which energy spectra and eigenfunctions versus the spheroid aspect ratio were calculated within the conventional effective mass approximation. Critical values of the aspect ratio, at which the discrete spectrum of models with finite-wall potentials is transformed into a continuous one in strong dimensional quantization regime, were revealed using the exact and adiabatic classifications.

1 Introduction

To analyze the geometrical, spectral and optical characteristics of quantum dots in the effective mass approximation and in the regime of strong dimensional quantization following [1], many methods and models were used, including the exactly solvable model of a spherical impermeable well [2], the adiabatic approximation for a lens-shaped well confined to a narrow wetting layer [3] and a hemispherical impermeable well [4], the model of strongly oblate or prolate ellipsoidal impermeable well [5], as well as numerical solutions of the boundary value problems (BVPs) with separable variables in the spheroidal coordinates for wells with infinite and finite wall heights [6,7,8]. However, thorough comparative analysis of spectral characteristics of models with different potentials, including those with non-separable variables, remains to be a challenging problem. This situation stimulates the study of a wider class of model well potentials with application of symbolic-numerical algorithms (SNA) and problem-oriented software developed by the authors of the present paper during years [9,10,11,12,13,14].

Here we analyse the spectral characteristics of the following models: a spherical quantum dot (SQD), an oblate spheroidal quantum dot (OSQD), and a prolate spheroidal quantum dot (PSQD). We make use of the Kantorovich method that reduces the problem to a set of ordinary differential equations (ODE) [15]. In contrast to the well-known method of adiabatic representation [16], this method implies neither adiabatic separation of fast and slow variables nor the presence of a small parameter. We present a calculation scheme for solving elliptical BVPs with axially-symmetric potentials in cylindrical coordinates (CC), spherical coordinates (SC), oblate spheroidal coordinates (OSC), and prolate spheroidal coordinates (PSC). Basing on the SNA developed for axially-symmetric potentials, different sets of solutions are constructed for the parametric BVPs related to the fast subsystem, namely, the eigenvalue problem solutions (the terms and the basis functions), depending upon the slow variable as a parameter, as well as the matrix elements, i.e., the integrals of the products of basis functions and their derivatives with respect to the parameter, which are calculated analytically by means of elaborated SNA MATRA implemented in MAPLE, or numerically using the program ODPEVP [13] implementing the finite-element method (FEM). These terms and matrix elements form the matrices of variable coefficients in the set of second-order ODE with respect to the slow variable. The BVP for this set of ODEs is solved by means of the program KANTBP [11], also implementing the FEM. The efficiency of the calculation scheme and the SNA used is demonstrated by comparison of the spectra versus the ellipticity of the prolate or oblate spheroid in the models of quantum dots with different confining potentials, such as the isotropic and anisotropic harmonic oscillator, the spherical and spheroidal well with finite or infinite walls approximated by smooth short-range potentials as well as by constructing the adiabatic classification of the states.

The paper is organized as follows. In Section 2, the calculation scheme for solving elliptic BVPs with axially-symmetric confining potentials is presented. In Section 3, SNA MATRA for solving parametric BVP and corresponding integrals implemented in Maple is described. Section 4 is devoted to the analysis of the spectra of quantum dot models with three types of axially-symmetric potentials, including the benchmark exactly solvable models. In Conclusion we summarize the results and discuss the future applications of our calculation scheme and the SNA project presented.

2 Problem Statement

Within the effective mass approximation under the conditions of strong dimensional quantization, the Schrödinger equation for the slow envelope of the wave function $\tilde{\Psi}(\tilde{\boldsymbol{r}})$ of a charge carrier (electron e or hole h) in the models of a spherical, prolate or oblate spheroidal quantum dot (SQD, PSQD or OSQD) has the form

$$\{\tilde{\hat{H}} - \tilde{E}\}\tilde{\Psi}(\tilde{\boldsymbol{r}}) = \{(2\mu_p)^{-1}\tilde{\hat{\boldsymbol{P}}}^2 + \tilde{U}(\tilde{\boldsymbol{r}}) - \tilde{E}\}\tilde{\Psi}(\tilde{\boldsymbol{r}}) = 0, \qquad (1)$$

where $\tilde{\boldsymbol{r}} \in \mathbf{R^3}$ is the position vector of the particle having the effective mass $\mu_p = \mu_e$ (or $\mu_p = \mu_h$), $\tilde{\hat{\boldsymbol{P}}} = -i\hbar\nabla_{\tilde{\boldsymbol{r}}}$ is the momentum operator, $\tilde{E}$ is the energy of the particle, $\tilde{U}(\tilde{\boldsymbol{r}})$ is the axially-symmetric potential confining the particle motion in SQD, PSQD or OSQD. In Model A, $\tilde{U}(\tilde{\boldsymbol{r}})$ is chosen to be the potential of an isotropic or anisotropic axially-symmetric harmonic oscillator with the angular frequency $\tilde{\omega} = \gamma_{r_0}\hbar/(\mu_p \tilde{r}_0^2)$, $\gamma_{r_0} \sim \pi^2/3$ being an adjustable parameter:

$$\tilde{U}^{\mathrm{L}}(\tilde{\boldsymbol{r}}) = \mu_p \tilde{\omega}^2(\zeta_1(\tilde{x}^2 + \tilde{y}^2) + \zeta_3 \tilde{z}^2)/2, \tag{2}$$

$r_0 = \sqrt{\zeta_1(\tilde{x}_0^2 + \tilde{y}_0^2) + \zeta_3 \tilde{z}_0^2}$ is the radius of a spherical QD ($\zeta_1 = 1, \zeta_3 = 1$) or that of a spheroidal QD ($\zeta_1 = (\tilde{r}_0/\tilde{a})^4$, $\zeta_3 = (\tilde{r}_0/\tilde{c})^4$), inscribed into a spherical one, where $\tilde{a}$ and $\tilde{c}$ are the semiaxes of the ellipse which transforms into a sphere at $\tilde{a} = \tilde{c} = \tilde{r}_0$. For Model B, $\tilde{U}(\tilde{\boldsymbol{r}})$ is the potential of a spherical or axially-symmetric well

$$\tilde{U}^{\mathrm{B}}(\tilde{\boldsymbol{r}}) = \{0, 0 \le (\tilde{x}^2 + \tilde{y}^2)/\tilde{a}^2 + \tilde{z}^2/\tilde{c}^2 < 1; \tilde{U}_0, (\tilde{x}^2 + \tilde{y}^2)/\tilde{a}^2 + \tilde{z}^2/\tilde{c}^2 \ge 1\}, \tag{3}$$

with walls of finite or infinite height $1 \ll \tilde{U}_0 < \infty$. For Model C, $\tilde{U}(\tilde{\boldsymbol{r}})$ is taken to be a spherical or axially-symmetric diffuse potential

$$\tilde{U}^{\mathrm{C}}(\tilde{\boldsymbol{r}}) = \tilde{U}_0\left[1 + \exp(((\tilde{x}^2 + \tilde{y}^2)/\tilde{a}^2 + \tilde{z}^2/\tilde{c}^2 - 1)/s)\right]^{-1}, \tag{4}$$

where s is the edge diffusiveness parameter of the function smoothly approximating the vertical walls of finite height $\tilde{U}_0$. Below we restrict ourselves by considering Model B with infinite walls $\tilde{U}_0 \to \infty$ and Model C with walls of finite height $\tilde{U}_0$. We make use of the reduced atomic units: $a_B^* = \kappa\hbar^2/\mu_p e^2$ is the reduced Bohr radius, κ is the DC permittivity, $E_R \equiv Ry^* = \hbar^2/(2\mu_p {a_B^*}^2)$ is the reduced Rydberg unit of energy, and the following dimensionless quantities are introduced: $\tilde{\Psi}(\tilde{\boldsymbol{r}}) = {a_B^*}^{-3/2}\Psi(\boldsymbol{r})$, $2\hat{H} = \tilde{\hat{H}}/Ry^*$, $2E = \tilde{E}/Ry^*$, $2U(\boldsymbol{r}) = \tilde{U}(\tilde{\boldsymbol{r}})/Ry^*$, $\boldsymbol{r} = \tilde{\boldsymbol{r}}/a_B^*$, $a = \tilde{a}/a_B^*$, $\tilde{c} = c/a_B^*$, $r_0 = \tilde{r}_0/a_B^*$, $\omega = \gamma_{r_0}/r_0^2 = \hbar\tilde{\omega}/(2Ry^*)$. For an electron with the reduced mass $\mu_p \equiv \mu_e = 0.067m_0$ at $\kappa = 13.18$ in GaAs: $a_B^* = 102$Å$= 10.2$ nm, $Ry^* = E_R = 5.2$ meV.

Since the Hamiltonian $\hat{H}$ in (1)–(4) commutes with the z-parity operator ($z \to -z$ or $\eta \to -\eta$), the solutions are divided into even ($\sigma = +1$) and odd ($\sigma = -1$) ones. The solution of Eq. (1), periodical with respect to the azimuthal angle φ, is sought in the form of a product $\Psi(x_f, x_s, \varphi) = \Psi^{m\sigma}(x_f, x_s)e^{im\varphi}/\sqrt{2\pi}$, where $m = 0, \pm1, \pm2, ...$ is the magnetic quantum number. Then the function $\Psi^{m\sigma}(x_f, x_s)$ satisfies the following equation in the two-dimensional domain $\Omega = \Omega_{x_f}(x_s) \cup \Omega_{x_s} \subset \mathbf{R}^2\backslash\{0\}$, $\Omega_{x_f}(x_s) = (x_f^{\min}(x_s), x_f^{\max}(x_s))$, $\Omega_{x_s} = (x_s^{\min}, x_s^{\max})$:

$$\left(\hat{H}_1(x_f; x_s) + \hat{H}_2(x_s) + V(x_f, x_s) - 2E\right)\Psi^{m\sigma}(x_f, x_s) = 0. \tag{5}$$

The Hamiltonian of the slow subsystem $\hat{H}_2(x_s)$ is expressed as

$$\hat{H}_2(x_s) = \check{H}_2(x_s) = -\frac{1}{g_{1s}(x_s)}\frac{\partial}{\partial x_s}g_{2s}(x_s)\frac{\partial}{\partial x_s} + \check{V}_s(x_s), \tag{6}$$

and the Hamiltonian of the fast subsystem $\hat{H}_1(x_f; x_s)$ is expressed via the reduced Hamiltonian $\check{H}_f(x_f; x_s)$ and the weighting factor $g_{3s}(x_s)$:

$$\begin{aligned} \hat{H}_1(x_f; x_s) &= g_{3s}^{-1}(x_s)\check{H}_f(x_f; x_s), \\ \check{H}_f(x_f; x_s) &= -\frac{1}{g_{1f}(x_f)}\frac{\partial}{\partial x_f}g_{2f}(x_f)\frac{\partial}{\partial x_f} + \check{V}_f(x_f) + \check{V}_{fs}(x_f, x_s). \end{aligned} \tag{7}$$

Table 1. The values of conditionally fast x_f and slow x_s independent variables, the coefficients $g_{is}(x_s)$, $g_{jf}(x_f)$, and the potentials $\check{V}_f(x_f)$, $\check{V}_s(x_s)$, $\check{V}_{fs}(x_f,x_s)$, in Eqs.(5)–(7) for SQD, OSQD and PSQD in cylindrical (CC), spherical (SC), and oblate & prolate spheroidal (OSC & PSC) coordinates with $(d/2)^2 = \pm(a^2-c^2)$, + for OSC, − for PSC.

	CC		SC	OSC &PSC
	OSQD	PSQD	SQD	OSQD & PSQD
x_f	z	ρ	η	η
x_s	ρ	z	r	ξ
g_{1f}	1	ρ	1	1
g_{2f}	1	ρ	$1-\eta^2$	$1-\eta^2$
g_{1s}	ρ	1	r^2	1
g_{2s}	ρ	1	r^2	$\xi^2 \pm 1$
g_{3s}	1	1	r^2	1
$\check{V}_f(x_f)$	$\omega^2\zeta_3 z^2$	$m^2/\rho^2+\omega^2\zeta_1\rho^2$	m^2/g_{2f}	$m^2/g_{2f} \pm (d/2)^2 g_{2f} 2E$
$\check{V}_s(x_s)$	$m^2/\rho^2+\omega^2\zeta_1\rho^2$	$\omega^2\zeta_3 z^2$	0	$\mp m^2/g_{2s} - ((d/2)^2 g_{2s} - 1)2E$
$\check{V}_{fs}(x_f,x_s)$	0	0	$\check{V}(r,\eta)$	$\check{V}(\xi,\eta)$

Table 1 contains the values of conditionally fast x_f and slow x_s independent variables, the coefficients $g_{1s}(x_s)$, $g_{2s}(x_s)$, $g_{3s}(x_s)$, $g_{1f}(x_f)$, $g_{2f}(x_f)$, and the reduced potentials $\check{V}_f(x_f)$, $\check{V}_s(x_s)$, $\check{V}_{fs}(x_f,x_s)$, entering Eqs. (5)–(7) for SQD, OSQD, and PSQD in cylindrical ($\boldsymbol{x}=(z,\rho,\varphi)$), spherical ($\boldsymbol{x}=(r,\eta=\cos\theta,\varphi)$), and oblate/prolate spheroidal ($\boldsymbol{x}=(\xi,\eta,\varphi)$) coordinates [17]. In spherical coordinates, the potential $\check{V}(r,\eta)$ in Table 1 using the definitions (2), (4) in the reduced atomic units, for Model A is expressed as

$$\check{V}(r,\eta) = 2r^2V(r,\eta) = \omega^2 r^4(\zeta_1(1-\eta^2)+\zeta_3\eta^2),$$

and for Model C as

$$\check{V}(r,\eta) = 2r^2V(r,\eta) = 2r^2U_0\left[1+\exp((r^2((1-\eta^2)/a^2+\zeta_3\eta^2/c^2)-1)/s)\right]^{-1},$$

both having zero first derivatives in the vicinity of the origin $r=0$ (equlibrium point). For Model B. the potentials $\check{V}_{fs}$ are zero, since the potential (3) is reformulated below in the form of boundary conditions with respect to the variables x_f and x_s. The solution $\Psi_i^{m\sigma}(x_f,x_s) \equiv \Psi_i^{Em\sigma}(x_f,x_s)$ of the problem (5)–(7) is sought in the form of Kantorovich expansion [15]

$$\Psi_i^{Em\sigma}(x_f,x_s) = \sum_{j=1}^{j_{\max}} \Phi_j^{m\sigma}(x_f;x_s)\chi_j^{(m\sigma i)}(E,x_s), \qquad (8)$$

using as a set of trial functions the eigenfunctions $\Phi_j^{m\sigma}(x_f;x_s)$ of the Hamiltonian $\check{H}_f(x_f;x_s)$ from (7), i.e., the solutions of the parametric BVP

$$\left\{\check{H}_f(x_f;x_s) - \check{\lambda}_i(x_s)\right\}\Phi_i^{m\sigma}(x_f;x_s) = 0, \qquad (9)$$

in the interval $x_f \in \Omega_{x_f}(x_s)$ depending on the conditionally slow variable $x_s \in \Omega_{x_s}$ as on a parameter. These solutions obey the boundary conditions

$$\lim_{x_f \to x_f^t(x_s)} \left(N_f^{(m\sigma)}(x_s) g_{2f}(x_f) \frac{d\Phi_j^{m\sigma}(x_f; x_s)}{dx_f} + D_f^{(m\sigma)}(x_s) \Phi_j^{m\sigma}(x_f; x_s) \right) = 0 \quad (10)$$

at the boundary points $\{x_f^{\min}(x_s), x_f^{\max}(x_s)\} = \partial\Omega_{x_f}(x_s)$, of the interval $\Omega_{x_f}(x_s)$. In Eq. (10), $N_f^{(m\sigma)}(x_s) \equiv N_f^{(m\sigma)}$, $D_f^{(m\sigma)}(x_s) \equiv D_f^{(m\sigma)}$, unless specially declared, are determined by the relations $N_f^{(m\sigma)} = 1$, $D_f^{(m\sigma)} = 0$ at $m = 0$, $\sigma = +1$ (or at $\sigma = 0$, i.e., without parity separation), $N_f^{(m\sigma)} = 0$, $D_f^{(m\sigma)} = 1$ at $m = 0$, $\sigma = -1$ or at $m \neq 0$. The eigenfunctions satisfy the orthonormality condition with the weighting function $g_{1f}(x_f)$ in the same interval $x_f \in \Omega_{x_f}(x_s)$:

$$\left\langle \Phi_i^{m\sigma} | \Phi_j^{m\sigma} \right\rangle = \int_{x_f^{\min}(x_s)}^{x_f^{\max}(x_s)} \Phi_i^{m\sigma}(x_f; x_s) \Phi_j^{m\sigma}(x_f; x_s) g_{1f}(x_f) dx_f = \delta_{ij}. \quad (11)$$

Here $\check{\lambda}_1(x_s) < ... < \check{\lambda}_{j_{\max}}(x_s) < ...$ is the desired set of real eigenvalues. The corresponding set of potential curves $2E_1(x_s) < ... < 2E_{j_{\max}}(x_s) < ...$ of Eqs. (7) is determined by $2E_j(x_s) = g_{3s}^{-1}(x_s)\check{\lambda}_j(x_s)$. Note that for OSC and PSC, the desired set of real eigenvalues $\check{\lambda}_j(x_s)$ depends on a combined parameter, $x_s \to p^2 = (d/2)^2 2E$, the product of spectral $2E$ and geometrical $(d/2)^2$ parameters of the problem (5). The solutions of the problem (9)–(11) for Models A and B are calculated in the analytical form, while for Model C this is done using the program ODPEVP [13].

Substituting the expansion (8) into Eq. (5) in consideration of (9) and (11), we get a set of ODEs for the slow subsystem with respect to the unknown vector functions $\boldsymbol{\chi}^{(m\sigma i)}(x_s, E) \equiv \boldsymbol{\chi}^{(i)}(x_s) = (\chi_1^{(i)}(x_s), ..., \chi_{j_{\max}}^{(i)}(x_s))^T$:

$$\left(-\frac{1}{g_{1s}(x_s)} \mathbf{I} \frac{d}{dx_s} g_{2s}(x_s) \frac{d}{dx_s} + 2\mathbf{E}(x_s) + \mathbf{I}\check{V}_s(x_s) - 2\mathbf{I}E \right) \boldsymbol{\chi}^{(i)}(x_s) = \quad (12)$$

$$= -\left(\frac{g_{2s}(x_s)}{g_{1s}(x_s)} \mathbf{W}(x_s) + \frac{1}{g_{1s}(x_s)} \frac{dg_{2s}(x_s)\mathbf{Q}(x_s)}{dx_s} + \frac{g_{2s}(x_s)}{g_{1s}(x_s)} \mathbf{Q}(x_s) \frac{d}{dx_s} \right) \boldsymbol{\chi}^{(i)}(x_s).$$

Here $2\mathbf{E}(x_s) = \mathrm{diag}(g_{3s}^{-1}(x_s)\check{\lambda}_j(x_s))$, $\mathbf{W}(x_s)$, and $\mathbf{Q}(x_s)$ are matrices of the dimension $j_{\max} \times j_{\max}$,

$$W_{ij}(x_s) = W_{ji}(x_s) = \int_{x_f^{\min}(x_s)}^{x_f^{\max}(x_s)} g_{1f}(x_f) \frac{\partial \Phi_i(x_f; x_s)}{\partial x_s} \frac{\partial \Phi_j(x_f; x_s)}{\partial x_s} dx_f, \quad (13)$$

$$Q_{ij}(x_s) = -Q_{ji}(x_s) = -\int_{x_f^{\min}(x_s)}^{x_f^{\max}(x_s)} g_{1f}(x_f) \Phi_i(x_f; x_s) \frac{\partial \Phi_j(x_f; x_s)}{\partial x_s} dx_f,$$

calculated analytically for Model B and by means of the program ODPEVP [13] for Model C. Note that for Model A in SC or CC and Model B in OSC or PSC, the variables x_f and x_s are separated so that the matrix elements $W_{ij}(x_s) = Q_{ij}(x_s) \equiv 0$ are put into the r.h.s. of Eq. (12), and $\check{V}_s(x_s)$ are substituted from Table 1. The discrete spectrum solutions $2E : 2E_1 < 2E_2 < ... < 2E_t < ...$ that obey the boundary conditions at points $x_s^t = \{x_s^{\min}, x_s^{\max}\} = \partial\Omega_{x_s}$ bounding the interval Ω_{x_s}:

$$\lim_{x_s \to x_s^t} \left(N_s^{(m\sigma)} g_{2s}(x_s) \frac{d\boldsymbol{\chi}^{(m\sigma p)}(x_s)}{dx_s} + D_s^{(m\sigma)} \boldsymbol{\chi}^{(m\sigma p)}(x_s) \right) = 0, \tag{14}$$

where $N_s^{(m\sigma)} = 1$, $D_s^{(m\sigma)} = 0$ at $m = 0, \sigma = +1$ (or at $\sigma = 0$, i.e without parity separation), $N_s^{(m\sigma)} = 0$, $D_s^{(m\sigma)} = 1$ at $m = 0, \sigma = -1$ or at $m \neq 0$, and the orthonormality conditions

$$\int_{x_s^{\min}}^{x_s^{\max}} (\boldsymbol{\chi}^{(i)}(x_s))^T \boldsymbol{\chi}^{(j)}(x_s) g_{1s}(x_s) dx_s = \delta_{ij}, \tag{15}$$

are calculated by means of the program KANTBP [11]. To ensure the prescribed accuracy of calculation of the lower part of the spectrum discussed below with eight significant digits we used $j_{\max} = 16$ basis functions in the expansion (8) and the discrete approximation of the desired solution by Lagrange finite elements of the fourth order with respect to the grid pitch $\Omega^p_{h^s(x_s)} = [x^s_{\min}, x^s_k = x^s_{k-1} + h^s_k, x^s_{\max}]$.

3 SNA MATRA for Calculation of the BVP and Integrals

To calculate the effective potentials of the problem (12)–(15) for each value $x_s = x_k^s$ of the FEM grid $\Omega^p_{h^s(x_s)} = [x^s_{\min}, x^s_{\max}]$ we consider a discrete representation of solutions $\Phi(x_f; x_s) \equiv \Phi^{m\sigma}(x_f; x_s)$ of the problem (9) by means of the FEM on the grid, $\Omega^p_{h^f(x_f)}(x_s) = [x_0^f = x^f_{\min}(x_s),\ x_k^f = x_{k-1}^f + h_k^f, x_{\bar{n}}^f = x^f_{\max}(x_s)]$, in a finite sum:

$$\Phi(x_f; x_s) = \sum_{\mu=0}^{\bar{n}p} \Phi_\mu^h(x_s) N_\mu^p(x_f) = \sum_{k=1}^{\bar{n}} \sum_{r=0}^{p} \Phi^h_{r+p(k-1)}(x_s) N^p_{r+p(k-1)}(x_f), \tag{16}$$

where $N_\mu^p(x_f)$ are local functions, and $\Phi_\mu^h(x_s)$ are node values of $\Phi(x_\mu^f; x_s)$. The local functions $N_\mu^p(x_f)$ are piece-wise polynomials of the given order, p equals one only in the node x_μ^f and equals zero in all other nodes $x_\nu^f \neq x_\mu^f$ of the grid $\Omega^p_{h^f(x_f)}(x_s)$, i.e., $N_\nu^p(x_\mu^f) = \delta_{\nu\mu}$, $\mu, \nu = 0, 1, \ldots, \bar{n}p$. The coefficients $\Phi_\nu(x_s)$ are formally connected with the solution $\Phi(x_{k,r}^{fp}; x_s)$ in a node $x_\nu^f = x_{k,r}^{fp}$, $k = 1, \ldots, \bar{n}$, $r = 0, \ldots, p$:

$$\Phi_\nu^h(x_s) = \Phi^h_{r+p(k-1)}(x_s) \approx \Phi(x_{k,r}^{fp}; x_s), \quad x_{k,r}^{fp} = x_{k-1}^f + \frac{h_k^f}{p} r.$$

The theoretical estimate for the $\mathbf{H^0}$ norm between the exact and numerical solution has the order of

$$|\check{\lambda}_j(x_s) - \check{\lambda}^h_j(x_s)| \le c_1 h^{2p}, \quad \left\|\Phi_j(x_f;x_s) - \boldsymbol{\Phi}^h_j(x_s)\right\|_0 \le c_2 h^{p+1}, \tag{17}$$

where $h^f = \max_{1<j<\bar{n}} h^f_j$ is the maximal step of the grid, and the constants $c_1 > 0$, $c_2 > 0$ do not depend on the step h^f [19]. It has been shown possible to construct schemes for solving the BVPs and integrals with high order of accuracy comparable with that of the computer in accordance with the following estimations [13]

$$\left|\frac{\partial\check{\lambda}_j(x_s)}{\partial x_s} - \frac{\partial\check{\lambda}^h_j(x_s)}{\partial x_s}\right| \le c_3 h^{2p}, \quad \left\|\frac{\partial\Phi_j(x_f;x_s)}{\partial x_s} - \frac{\partial\boldsymbol{\Phi}^h_j(x_s)}{\partial x_s}\right\|_0 \le c_4 h^{p+1}, \tag{18}$$

$$\left|Q_{ij}(x_s) - Q^h_{ij}(x_s)\right| \le c_5 h^{2p}, \quad \left|W_{ij}(x_s) - W^h_{ij}(x_s)\right| \le c_6 h^{2p}, \tag{19}$$

where h^f is the grid step, p is the order of finite elements, i, j are the numbers of the corresponding solutions, and the constants c_3, c_4, c_5, and c_6 do not depend on the step h^f. The proof is straightforward following the scheme of the proof of estimations (17) in accordance with [19,20]. Verification of the above estimations is provided by numerical analysis on condensed grids and by comparison with examples of exact solvable models A and B.

Let us consider the reduction of BVP (9), (11) in the interval $\Delta: \ x^f_{\min}(x_s) < x_f < x^f_{\max}(x_s)$ with the boundary conditions (10) at points $x^f_{\min}(x_s)$ and $x^f_{\max}(x_s)$ rewritten in the form

$$\mathbf{A}(x_s)\Phi_j(x_f;x_s) = \check{\lambda}_j(x_s)\mathbf{B}(x_s)\Phi_j(x_f;x_s), \tag{20}$$

where $\mathbf{A}(x_s)$ is a differential operator, and $\mathbf{B}(x_s)$ is a multiplication operator, differentiable with respect to the parameter $x_s \in \Omega_{x_s}$. Substituting the expansion (16) into (20) and performing integration with respect to x_f by parts in the interval $\Delta = \cup^{\bar{n}}_{k=1}\Delta_k$, we arrive at a set of linear algebraic equations

$$\mathbf{a}^p_{\mu\nu}(x_s)\Phi^h_{j,\mu}(x_s) = \check{\lambda}^h_j(x_s)\mathbf{b}^p_{\mu\nu}(x_s)\Phi^h_{j,\mu}(x_s), \tag{21}$$

in the framework of the briefly described FEM. Using the p-order Lagrange elements [19], we present below Algorithm 1 for constructing the algebraic problem (21) by the FEM in the form of conventional pseudocode. Its MAPLE realization allows us to show explicitly the recalculation of indices μ, ν and to test the corresponding modules of the parametric matrix problems, derivatives of solutions by parameter, and calculation of integrals.

Algorithm 1. Generation of parametric algebraic problems

Input:
$\varDelta=\cup_{k=1}^{\bar{n}}\varDelta_k=[x^f_{\min}(x_s),x^f_{\max}(x_s)]$ is the interval of changing of the independent variable x_f, whose boundaries depend on the parameter $x_s=x^s_{k'}$;
$h^f_k=x^f_k-x^f_{k-1}$ is the grid step;
$\bar{n}$ is the number of subintervals $\varDelta_k=[x^f_{k-1},x^f_k]$;
p is the order of finite elements;
$\mathbf{A}(x_s),\mathbf{B}(x_s)$ are the differential operators in Eq. (20);
Output:
$N^p_\mu(x_f)$ are the basis functions in (16);
$\mathbf{a}^p_{\mu\nu}(x_s)$, $\mathbf{b}^p_{\mu\nu}(x_s)$ are the matrix elements in the system of algebraic equations (21);
Local:
$x^{fp}_{k,r}$ are the nodes; $\phi^p_{k,r}(x_f)$ are the Lagrange elements; $\mu,\nu=0,1,\ldots,\bar{n}p$;

```
1: for $k$:=1 to $\bar{n}$ do
        for $r$:=0 to $p$ do
            $x^{fp}_{k,r}=x^f_{k-1}+\frac{h^f_k}{p}r$
        end for;
    end for;
2: $\phi^p_{k,r}(x_f)=\prod_{r'\neq r}[(x_f-x^{fp}_{k,r'})(x^{fp}_{k,r}-x^{fp}_{k,r'})^{-1}]$
3: $N^p_0(x_f)$:= if $x_f\in\varDelta_1$ then $\phi^p_{1,0}(x_f)$ else 0;
    for $k$:=1 to $\bar{n}$ do
        for $r$:=1 to $p-1$ do
            $N^p_{r+p(k-1)}(x_f)$: = if $x_f\in\varDelta_k$ then $\phi^p_{k,r}(x_f)$ else 0;
        end for;
        $N^p_{kp}(x_f)$:= if $x_f\in\varDelta_k$ then $\phi^p_{k,p}(x_f)$
                    else if $x_f\in\varDelta_{k+1}$ then $\phi^p_{k+1,0}(x_f)$ else 0;
    end for;
    $N^p_{\bar{n}p}(x_f)$:= if $x_f\in\varDelta_{\bar{n}}$ then $\phi^p_{\bar{n},p}(x_f)$ else 0;
4: for $\mu,\nu$:=0 to $\bar{n}p$ do
        $\mathbf{a}^p_{\mu\nu}(x_s):=\int_\varDelta g_1(x_f)N^p_\mu(x_f)\mathbf{A}(x_s)N^p_\nu(x_f)dx_f$;
        $\mathbf{b}^p_{\mu\nu}(x_s):=\int_\varDelta g_1(x_f)N^p_\mu(x_f)\mathbf{B}(x_s)N^p_\nu(x_f)dx_f$;
    end for;
```

Remarks:

1. For equation (9), the matrix elements of the operator (7), and $V(x_f;x_s)=\check{V}_{fs}(x_f,x_s)+\check{V}_f(x_f)$ between the local functions $N_\mu(x_f)$ and $N_\nu(x_f)$ defined in the same interval $\varDelta_j$ calculated by formula using $x_f=x^f_{k-1}+0.5h^f_k(1+\eta_f)$, $q,r=\overline{0,p}$:

$$(\mathbf{a}(x_s))_{\mu,\nu} = \int\limits_{-1}^{+1} \left\{ \tfrac{4}{(h_k^f)^2} g_{2f}(x_f)(\phi_{k,q}^p)'(\phi_{k,r}^p)' + g_{1f}(x_f)V(x_f;x_s)\phi_{k,q}^p\phi_{k,r}^p \right\} \tfrac{h_k^f}{2} d\eta_f,$$
$$(\mathbf{b}(x_s))_{\mu,\nu} = \int\limits_{-1}^{+1} g_{1f}(x_f)\phi_{k,q}^p\phi_{k,r}^p \tfrac{h_k^f}{2} d\eta_f, \quad \mu = q + p(k-1), \quad \nu = r + p(k-1).$$

2. If the integrals can not be calculated analytically (see, e.g., section 4), then they are calculated by numerical methods [19], namely, by means of the Gauss quadrature formulae of the order $p+1$.

3. For OSQD&PSQD model C, the problem (9)–(11) has been solved using the grid $\Omega^p_{h^f(x_f)}(x_s)[x^f_{\min}, x^f_{\max}] = -1(20)1$ (the number in parentheses denotes the number of finite elements of order $p = 4$ in each interval).

Generally, 10-16 iterations are required for the subspace iterations to converge the subspace to within the prescribed tolerance. If the matrix $\mathbf{a}^p \equiv \mathbf{a}^p(x_s)$ in Eq. (21) is not positively defined, the problem (21) is replaced by the following problem:

$$\tilde{\mathbf{a}}^p\, \boldsymbol{\Phi}^h = \tilde{\lambda}^h\, \mathbf{b}^p\, \boldsymbol{\Phi}^h, \quad \tilde{\mathbf{a}}^p = \mathbf{a}^p - \alpha \mathbf{b}^p. \tag{22}$$

The number α (the shift of the energy spectrum) is chosen in such a way that the matrix $\tilde{\mathbf{a}}^p$ is positive. The eigenvector of the problem (22) is the same, and $\check{\lambda}^h = \tilde{\lambda}^h + \alpha$, where the shift α is evaluated by Algorithm 2.

Algorithm 2. Evaluating the lower bound for the lowest eigenvalue of the generalized eigenvalue problem

Generally it is impossible to define the lower bound for the lowest eigenvalue of Eq. (22) because the eigenvalues $\check{\lambda}_1^h(x_s) < ... < \check{\lambda}_i^h(x_s) < ... < \check{\lambda}_{j_{max}}^h(x_s)$ depend upon the parameter x_s. However, we can use the following algorithm to find the lower bound for the lowest eigenvalue $\check{\lambda}_1^h(x_s)$ at a fixed value of x_s:

```
Step 1. Calculate $\mathbf{L\,D\,L^T}$ factorization of $\mathbf{A}^p - \alpha\mathbf{B}^p$.
Step 2. If some elements of the diagonal matrix $\mathbf{D}$ are less than zero
            then put $\alpha = \alpha - 1$ and go to Step 3, else go to Step 5.
Step 3. Calculate $\mathbf{L\,D\,L^T}$ factorization of $\mathbf{A}^p - \alpha\mathbf{B}^p$.
Step 4. If some elements of the diagonal matrix $\mathbf{D}$ are less than zero
            then put $\alpha = \alpha - 1$ and go to Step 3, else put $\alpha = \alpha - 0.5$
            and go to Step 8.
Step 5. Put $\alpha = \alpha + 1$ and calculate $\mathbf{L\,D\,L^T}$ factorization of $\mathbf{A}^p - \alpha\mathbf{B}^p$.
Step 6. If all elements of the diagonal matrix $\mathbf{D}$ are greater than zero
            then go to Step 5.
Step 7. Put $\alpha = \alpha - 1.5$.
Step 8. End.
```

After using the above algorithm one should find the lower bound for the lowest eigenvalue, and always $\check{\lambda}_1^h(x_s) - \alpha \leq 1.5$.

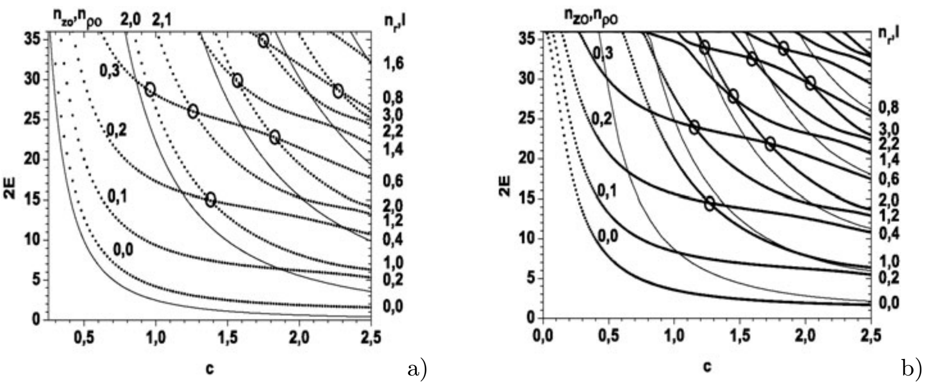


Fig. 1. The energies $2E = \tilde{E}/E_R$ of even $\sigma = +1$ lower states for OSQD versus the minor c, $\zeta_{ca} = c/a \in (1/5, 1)$ being the spheroid aspect ratio: a) well with impermeable walls, b) diffusion potential with $2U_0 = 36$, $s = 0.1$, the major semiaxis $a = 2.5$ and $m = 0$. Tine lines are minimal values $2E_i^{min} \equiv 2E_i(x_s = 0)$ of potential curves.

4 Spectral Characteristics of Spheroidal QDs

Models B and C for Oblate Spheroidal QD. At fixed coordinate x_s of the slow subsystem, the motion of the particle in the fast degree of freedom x_f is localized within the potential well having the effective width

$$\tilde{L}\,(x_s) = 2c\sqrt{1 - x_s^2/a^2}, \tag{23}$$

where $L = \tilde{L}/a_B^*$. The parametric BVP (9)–(11) at fixed values of the coordinate x_s, $x_s \in (0, a)$, is solved in the interval $x_f \in (-L\,(x_s)/2, L\,(x_s)/2)$ for Model C using the program ODPEVP, and for Model B the eigenvalues $\tilde{E}_{n_o}(x_s)/E_R \equiv 2E_i\,(x_s)$, $n_o = i = 1, 2, ...$, and the corresponding parametric eigenfunctions $\Phi_i^\sigma\,(x_f; x_s)$, obeying the boundary conditions (10) and the normalization condition (11), are expressed in the analytical form:

$$2E_i\,(x_s) = \frac{\pi^2 n_o^2}{L^2\,(x_s)}, \quad \Phi_i^\sigma\,(x_f; x_s) = \sqrt{\frac{2}{L\,(x_s)}} \sin\left(\frac{\pi n_o}{2}\left(\frac{x_f}{L\,(x_s)/2} - 1\right)\right), \tag{24}$$

where the even solutions $\sigma = +1$ are labelled with odd $n_o = n_{zo} + 1 = 2i - 1$, and the odd ones $\sigma = -1$ with even $n_o = n_{zo} + 1 = 2i$, $i = 1, 2, 3, ...$. The effective potentials (13) in Eq. (12) for the slow subsystem are expressed analytically via the integrals over the fast variable x_f of the basis functions (24) and their derivatives with respect to the parameter x_s including states with both parities $\sigma = \pm 1$:

$$2E_i(x_s) = \frac{a^2\pi^2 n_o^2}{4c^2(a^2 - x_s^2)}, \quad W_{ii}(x_s) = \frac{3 + \pi^2 n_o^2}{12}\frac{x_s^2}{(a^2 - x_s^2)^2}, \tag{25}$$

$$W_{ij}(x_s) = \frac{2n_o n_o'(n_o^2 + n_o'^2)(1 + (-1)^{n_o + n_o'})}{(n_o^2 - n_o'^2)^2}\frac{x_s^2}{(a^2 - x_s^2)^2},$$

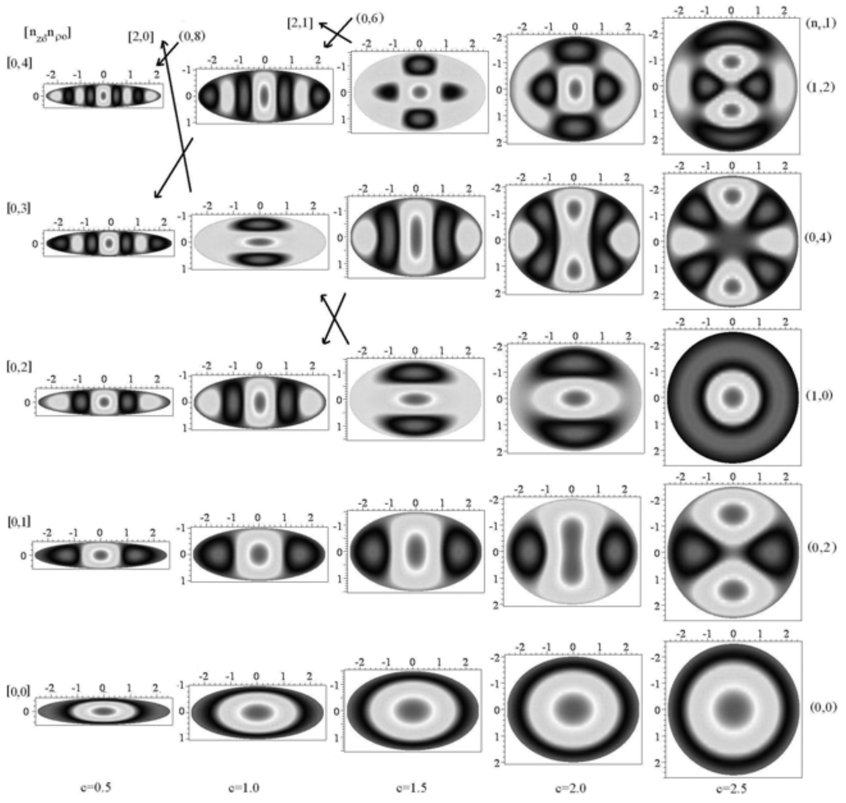


Fig. 2. Contour lines of the first five even-parity wave functions $\sigma = +1$ in the xz plane of Model B of OSQD for the major semiaxis $a = 2.5$ and different values of the minor semiaxis c ($\zeta_{ca} = c/a \in (1/5, 1)$)

$$Q_{ij}(x_s) = \frac{n_o n_o'(1 + (-1)^{n_o + n_o'})}{(n_o^2 - n_o'^2)^2} \frac{x_s}{a^2 - x_s^2}, \quad n_o' \neq n_o.$$

For Model B at $c = a = r_0$ the OSQD turns into SQD with known analytically expressed energy levels $E_t \equiv E^{sp}_{nlm}$ and the corresponding eigenfunctions

$$2E^{sp}_{nlm} = \frac{\alpha^2_{n_r+1,l+1/2}}{r_0^2}, \; \Phi^{sp}_{nlm}(r,\theta,\varphi) = \frac{\sqrt{2} J_{l+1/2}(\sqrt{2E^{sp}_{nlm}} r)}{r_0\sqrt{r}|J_{l+3/2}(\alpha_{n_r+1,l+1/2})|} Y_{lm}(\theta,\varphi), \quad (26)$$

where $\alpha_{n_r+1,l+1/2}$ are zeros of the Bessel function of semi-integer index $l + 1/2$, numbered in ascending order $0 < \alpha_{11} < \alpha_{12} < ... < \alpha_{iv} < ...$ by the integer $i, v = 1, 2, 3, ...$ Otherwise one can use equivalent pairs $iv \leftrightarrow \{n_r, l\}$ with $n_r = 0, 1, 2, ...$ numbering the zeros of Bessel function and $l = 0, 1, 2, ...$ being the orbital quantum number that determines the parity of states $\hat{\sigma} = (-1)^l = (-1)^m \sigma$, $\sigma = (-1)^{l-m} = \pm 1$. At fixed l, the energy levels $\tilde{E}_{nlm}/E_R = 2E_t$ degenerate with respect to the magnetic quantum number m, are labelled with

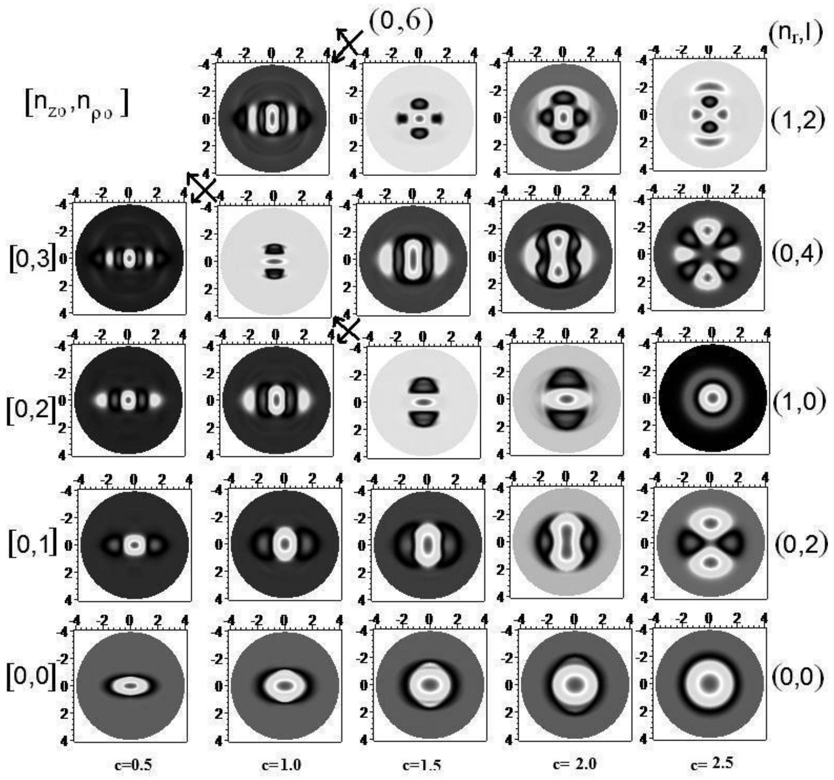


Fig. 3. Contour lines of the first five even-parity wave functions $\sigma = +1$ in the xz plane of Model C of OSQD with $2U_0 = 36$ and $s = 0.1$ for the major semiaxis $a = 2.5$ and different values of the minor semiaxis c ($\zeta_{ca} = c/a \in (1/5, 1)$)

the quantum number $n = n_r + 1 = i = 1, 2, 3, \ldots$, in contrast to the spectrum of a spherical oscillator, degenerate with respect to the quantum number $\lambda = 2n_r + l$. Figures 1, 2, and 3 show the lower part of non-equidistant spectrum $\tilde{E}(\zeta_{ca})/E_R = 2E_t$ and the eigenfunctions $\Psi_t^{m\sigma}$ from Eq. (8) for even states OSQD Models B and C at $m = 0$. There is a one-to-one correspondence rule $n_o = n_{zo} + 1 = 2n - (1+\sigma)/2, n = 1, 2, 3, \ldots$, $n_\rho = (l - |m| - (1-\sigma)/2)/2$, between the sets of spherical quantum numbers $(n, l, m, \hat{\sigma})$ of SQD with radius $r_0 = a = c$ and spheroidal ones $(n_\xi = n_r, n_\eta = l - |m|, m, \sigma)$ of OSQD with the major a and the minor c semiaxes, and the adiabatic set of cylindrical quantum numbers $(n_{zo}, n_\rho, m, \sigma)$ at continuous variation of the parameter $\zeta_{ca} = c/a$. The presence of crossing points of the energy levels of similar parity under the symmetry change from spherical $\zeta_{ca} = 1$ to axial, i.e., under the variation of the parameter $0 < \zeta_{ca} < 1$, in the BVP with two variables at fixed m for Model B is caused by the possibility of variable separation in the OSC [17], i.e., the r.h.s. of Eq. (12) equals zero. The transformation of eigenfunctions occurring in the course of a transition through the crossing points (marked by circles) in Fig. 1, is shown in Fig. 2 for model B and in Fig. 3 for model C (marked by arrows). From

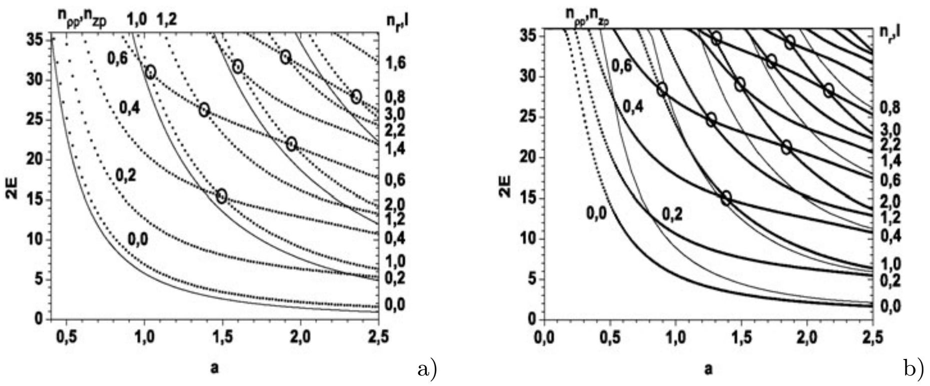


Fig. 4. The energies $2E = \tilde{E}/E_R$ of even $\sigma = +1$ lowest states for PSQD depending on the minor semiaxis a ($\zeta_{ac} = a/c \in (1/5, 1)$ is the spheroid aspect ratio): a) well with impermeable walls, b) diffusion potential, $2U_0 = 36$, $s = 0.1$, for the major semiaxis $c = 2.5$ and $m = 0$. Tine lines are minimal values $2E_i^{min} \equiv 2E_i(x_s = 0)$ of potential curves.

comparison of these Figures one can see that if the eigenfunctions are ordered according to increasing eigenvalues of the BVPs, then for both Models B and C, the number of nodes [18] is invariant under the variation of parameter c from $c = a = 2.5$ to $c = 0.5$ of potentials (3) and (4). For Model B, such a behavior follows from the fact of separation of variables of the BVP with potential (3) in the OSC (see Table 1), while for Model C, further investigation is needed because the coordinate system, where the variables of the BVP with potential (4) are separated, is unknown. So, at small value of deformation parameter (ζ_{ca} for OSQD or ζ_{ac} for PSQD) there are nodes only along corresponding major axis. For Model C, at each value of the parameter a their is a finite number of discrete energy levels limited by the value $2U_0$ of the well walls height. As shown in Fig. 1b, the number of levels of OSQD, equal to that of SQD at $a = c = r_0$, is reduced with the decrease of the parameter c (or ζ_{ca}), in contrast to Models A and B that have countable spectra, and avoided crossings appear just below the threshold.

Models B and C for Prolate Spheroidal QD. In contrast to OSQD, for PSQD at fixed coordinate x_s of the slow subsystem the motion of the particle is confined to a 2D potential well with the effective variable radius

$$\rho_0(x_s) = a\sqrt{1 - x_s^2/c^2}, \tag{27}$$

where $\rho_0(x_s) = \tilde{\rho}_0(x_s)/a_B$. The parametric BVP (9)–(11) at fixed values of the coordinate x_s from the interval $x_s \in (-c, c)$ is solved in the interval $x_f \in (0, \rho_0(x_s))$ for Model C using the program ODPEVP, while for Model B the eigenvalues $\tilde{E}_{n_{\rho p}+1}(x_s)/E_R \equiv 2E_i(x_s)$, $n_{\rho p} + 1 = i = 1, 2, ...$, and the corresponding parametric basis functions $\Phi_i^{m\sigma=0}(x_f; x_s) \equiv \Phi_i^m(x_f; x_s)$ without

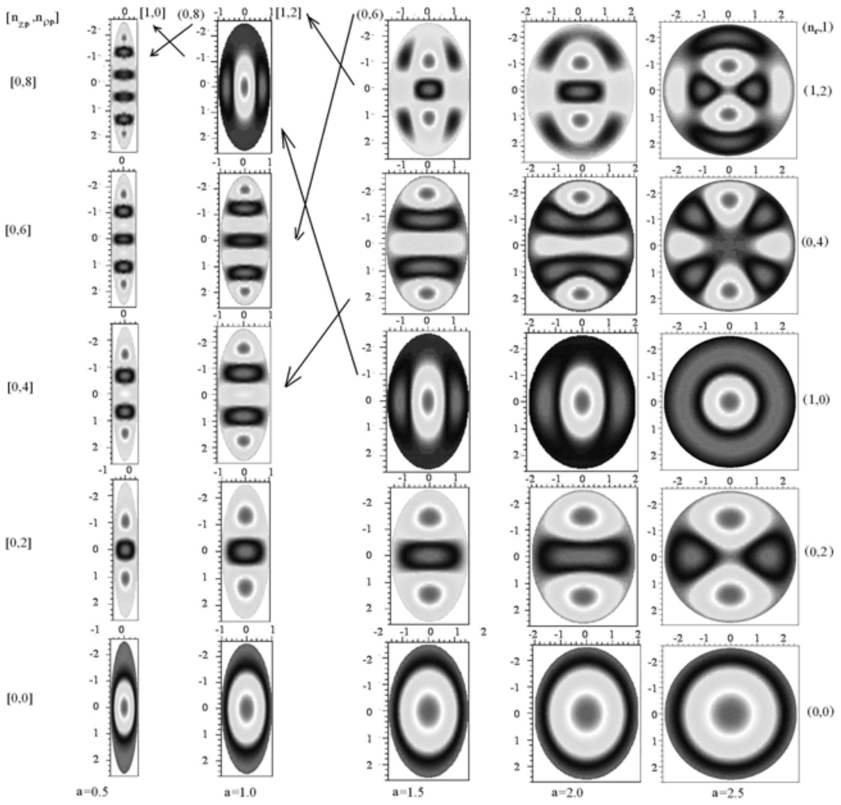


Fig. 5. Contour lines of the first five even-parity wave functions $\sigma = +1$ in the xz plane of Model B of PSQD for the major semiaxis $c = 2.5$ and different values of the minor semiaxis a ($\zeta_{ac} = a/c \in (1/5, 1)$)

parity separation obeying the boundary conditions (10) and the normalization condition (11) are expressed in the analytical form:

$$2E_i(x_s) = \frac{\alpha^2_{n_{\rho p}+1,|m|}}{\rho_0^2(x_s)}, \quad \Phi^m_{n_{\rho p}}(x_s) = \frac{\sqrt{2}}{\rho_0(x_s)} \frac{J_{|m|}(\sqrt{2E_{n_{\rho p}+1,|m|}(x_s)}x_f)}{|J_{|m|+1}(\alpha_{n_{\rho p}+1,|m|})|}, \quad (28)$$

where $\alpha_{n_{\rho p}+1,|m|} = \bar{J}^{n_{\rho p}+1}_{|m|}$ are positive zeros of the Bessel function of the first kind $J_{|m|}(x_f)$ labeled in the ascending order with the quantum number $n_{\rho p}+1 = i = 1, 2, \ldots$. The effective potentials (13) in Eq.(12) for the slow subsystem are calculated numerically in quadratures via the integrals over the fast variable x_f of the basis functions(28) and their derivatives with respect to the parameter x_s using SNA MATRA from Section 2. Figures 4 and 5 illustrate the lower part of the non-equidistant spectrum $E(\zeta_{ac})/E_R = 2\tilde{E}_t$ and the eigenfunctions $\Psi^{m\sigma}_t$ from Eq. (8) of even states of PSQD Models B and C. There is a one-to-one correspondence rule $n_{\rho p} + 1 = n_p = i = n = n_r + 1$, $i = 1, 2, \ldots$ and $n_{zp} = l - |m|$ between the sets of quantum numbers $(n, l, m, \hat{\sigma})$ of SQD with the radius $r_0 = a = c$ and spheroidal ones $(n_\xi = n_r, n_\eta = l - |m|, m, \sigma)$ of PSQD with the major c and the minor a semiaxes, and the adiabatic set of quantum numbers $(n = n_{\rho p}+1, n_{zp}, m, \sigma)$ under the continuous variation of the parameter $\zeta_{ac} = a/c$. The presence of crossing points of similar-parity energy levels in Fig. 4 under the change of symmetry from spherical $\zeta_{ac} = 1$ to axial, i.e., under the variation of the parameter $0 < \zeta_{ac} < 1$, in the BVP with two variables at fixed m for Model B is caused by the possibility of variable separation in the PSC [17], i.e., r.h.s. of Eq. (12) equals zero. For Model C, at each value of the parameter c there is also only a finite number of discrete energy levels limited by the value $2U_0$ of the well walls height. As shown in Fig. 4b, the number of energy levels of PSQD, equal to that of SQD at $a = c = r_0$, which is determined by the product of mass μ_e of the particle, the well depth $\tilde{U}_0$, and the square of the radius $\tilde{r}_0$, is reduced with the decrease of the parameter $\tilde{a}$ (or ζ_{ac}) because of the promotion of the potential curve (lower bound) into the continuous spectrum, in contrast to Models A and B having countable spectra. Note that the spectrum of Model C for PSQD or OSQD should approach that of Model B with the growth of the walls height U_0 of the spheroidal well. However, at critical values of the ellipsoid aspect ratio it is shown that in the effective mass approximation, both the terms (lower bound) and the discrete energy eigenvalues in models of the B type move into the continuum. Therefore, when approaching the critical aspect ratio values, it is necessary to use models such as the lens-shaped self-assembled QDs with a quantum well confined to a narrow wetting layer [3] or if a minor semiaxis becomes comparable with the lattice constant to consider models (see,e.g.[21]), different from the effective mass approximation.

5 Conclusion

By examples of the analysis of energy spectra of SQD, PSQD, and OSQD models with thee types of axially symmetric potentials, the efficiency of the developed computational scheme and SNA is demonstrated. Only Model A (anisotropic harmonic oscillator potential) is shown to have an equidistant spectrum, while Models B and C (wells with infinite and finite walls height) possess non-equidistant spectra. In Model C, there is a finite number of energy levels. This number becomes smaller as the parameter a or c (ζ_{ac} or ζ_{ca}) is reduced because the potential curve (lower bound) moves into the continuum. Models A and B have countable discrete spectra. This difference in spectra allows verification of SQD, PSQD, and OSQD models using experimental data [2], e.g., photoabsorption, from which not only the energy level spacing, but also the mean geometric dimensions of QD may be derived [5,7,8]. It is shown that there are critical values of the ellipsoid aspect ratio, at which in the approximation of effective mass the discrete spectrum of models with finite-wall potentials turns into a continuous one. Hence, using experimental data, it is possible to verify different QD models like the lens-shaped self-assembled QDs with a quantum well confined to a narrow wetting layer [3], or to determine the validity domain of the effective mass approximation, if a minor semiaxis becomes comparable with the lattice constant and to proceed opportunely to more adequate models such as [21].

Note *a posteriori* that the diagonal approximation of the slow-variable ODE (12) without the diagonal matrix element W_{ii} (so-called rude adiabatic approximation) provides the lower estimate of the calculated energy levels. With this matrix element taken into account (adiabatic approximation), the upper estimate of energy is provided, unless in the domain of the energy level crossing points. Therefore, the Born–Oppenheimer (BO) approximation is generally applicable only for estimating the ground state at an appropriate value of the small parameter. For Model B in the first BO approximation $2E_i \approx E_i^{(0)} + E_i^{(1)}$ is given by the minimal value of the slow subsystem energy $E_1^{\min}(x_s)$ at the equilibrium points $x_s = 0$ (namely, $E_i^{(0)} = \pi^2 n_o^2/(2c)^2$ from Eq. (24) for OSQD and $E_i^{(0)} = \alpha^2_{n_{\rho p}+1}/a^2$ from Eq. (28) for PSQD), and by the corresponding energy values $E_i^{(1)} = \pi(ac)^{-1}n_o(2n_\rho + |m| + 1)$ and $E_i^{(1)} = 2(ac)^{-1}\alpha_{n_{\rho p}+1,|m|}(n_z + 1/2)$ of the 2D and 1D harmonic oscillator, respectively. It is shown in [4] that the terms $E_i(x_s)$ allow high-precision approximation by the Hulten potential. This can be accomplished by means of computer algebra software, e.g., Maple, Mathematica, which allows (in the rude adiabatic approximation) to obtain the lower bound of the spectrum by solving transcendental equations expressed analytically in terms of known special functions, and to use this approach for further development of our SNA project.

The software package developed is applicable to the investigation of impurity and exciton states in semiconductor nanostructure models. Further development of the method and the software package is planned for solving the quasi-2D and quasi-1D BVPs with both discrete and continuous spectrum, which are necessary for calculating the optical transition rates, channeling and transport characteristics in the models like quantum wells and quantum wires.

Authors thank Profs. K.G. Dvoyan, E.M. Kazaryan, and H.A. Sarkisyan for collaboration in the field and Profs. T. Sturm and C. Philips for support of our SNA project. This work was done within the framework of the Protocol No. 3967-3-6-09/11 of collaboration between JINR and RAU in dynamics of finite-dimensional quantum models and nanostructures in external fields. The work was supported partially by RFBR (grants 10-01-00200 and 08-01-00604), and by the grant No. MK-2344.2010.2 of the President of Russian Federation.

References

1. Harrison, P.: Quantum Well, Wires and Dots. In: Theoretical and Computational Physics of Semiconductor Nanostructures. Wiley, New York (2005)
2. Gambaryan, K.M.: Interaction and Cooperative Nucleation of InAsSbP Quantum Dots and Pits on InAs(100) Substrate. Nanoscale. Res. Lett. (2009), doi:10.1007/s11671-009-9510-8
3. Wojs, A., Hawrylak, P., Fafard, S., Jacak, L.: Electronic structure and magneto-optics of self-assembled quantum dots. Phys. Rev. B 54, 5604–5608 (1996)
4. Juharyan, L.A., Kazaryan, E.M., Petrosyan, L.S.: Electronic states and interband light absorption in semi-spherical quantum dot under the influence of strong magnetic field. Solid State Comm. 139, 537–540 (2006)
5. Dvoyan, K.G., Hayrapetyan, D.B., Kazaryan, E.M., Tshantshapanyan, A.A.: Electron States and Light Absorption in Strongly Oblate and Strongly Prolate Ellipsoidal Quantum Dots in Presence of Electrical and Magnetic Fields. Nanoscale Res. Lett. 2, 601–608 (2007)

6. Cantele, G., Ninno, D., Iadonisi, G.: Confined states in ellipsoidal quantum dots. J. Phys. Condens. Matt. 12, 9019–9036 (2000)
7. Trani, F., Cantele, G., Ninno, D., Iadonisi, G.: Tight-binding calculation of the optical absorption cross section of spherical and ellipsoidal silicon nanocrystals. Phys. Rev. B 72, 075423 (2005)
8. Lepadatu, A.-M., Stavarache, I., Ciurea, M.L., Iancu, V.: The influence of shape and potential barrier on confinement energy levels in quantum dots. J. Appl. Phys. 107, 033721 (2010)
9. Vinitsky, S.I., Gerdt, V.P., Gusev, A.A., Kaschiev, M.S., Rostovtsev, V.A., Samoilov, V.N., Tupikova, T.V., Chuluunbaatar, O.: A symbolic-numerical algorithm for the computation of matrix elements in the parametric eigenvalue problem. Programming and Computer Software 33, 105–116 (2007)
10. Chuluunbaatar, O., Gusev, A., Gerdt, V., Kaschiev, M., Rostovtsev, V., Samoylov, V., Tupikova, T., Vinitsky, S.: A Symbolic-numerical algorithm for solving the eigenvalue problem for a hydrogen atom in the magnetic field: cylindrical coordinates. In: Ganzha, V.G., Mayr, E.W., Vorozhtsov, E.V. (eds.) CASC 2007. LNCS, vol. 4770, pp. 118–133. Springer, Heidelberg (2007)
11. Chuluunbaatar, O., Gusev, A.A., Abrashkevich, A.G., Amaya-Tapia, A., Kaschiev, M.S., Larsen, S.Y., Vinitsky, S.I.: KANTBP: A program for computing energy levels, reaction matrix and radial wave functions in the coupled-channel hyperspherical adiabatic approach. Comput. Phys. Commun. 177, 649–675 (2007)
12. Chuluunbaatar, O., Gusev, A.A., Gerdt, V.P., Rostovtsev, V.A., Vinitsky, S.I., Abrashkevich, A.G., Kaschiev, M.S., Serov, V.V.: POTHMF: A program for computing potential curves and matrix elements of the coupled adiabatic radial equations for a hydrogen-like atom in a homogeneous magnetic field. Comput. Phys. Commun. 178, 301–330 (2008)
13. Chuluunbaatar, O., Gusev, A.A., Vinitsky, S.I., Abrashkevich, A.G.: ODPEVP: A program for computing eigenvalues and eigenfunctions and their first derivatives with respect to the parameter of the parametric self-adjoined Sturm-Liouville problem. Comput. Phys. Commun. 180, 1358–1375 (2009)
14. Vinitsky, S.I., Chuluunbaatar, O., Gerdt, V.P., Gusev, A.A., Rostovtsev, V.A.: Symbolic-numerical algorithms for solving parabolic quantum well problem with hydrogen-like impurity. In: Gerdt, V.P., Mayr, E.W., Vorozhtsov, E.V. (eds.) CASC 2009. LNCS, vol. 5743, pp. 334–349. Springer, Heidelberg (2009)
15. Kantorovich, L.V., Krylov, V.I.: Approximate Methods of Higher Analysis. Wiley, New York (1964)
16. Born, M., Huang, X.: Dynamical Theory of Crystal Lattices. The Clarendon Press, Oxford (1954)
17. Abramowitz, M., Stegun, I.A.: Handbook of Mathematical Functions. Dover, New York (1965)
18. Courant, R., Hilbert, D.: Methods of Mathematical Physics, vol. 1. Wiley, Chichester (1989)
19. Strang, G., Fix, G.J.: An Analysis of the Finite Element Method. Prentice-Hall, Englewood Cliffs (1973)
20. Schultz, M.H.: L^2 Error Bounds for the Rayleigh-Ritz-Galerkin Method. SIAM J. Numer. Anal. 8, 737–748 (1971)
21. Harper, P.G.: Single Band Motion of Conduction Electrons in a Uniform Magnetic Field. Proc. Phys. Soc. A 68, 874–878 (1955)

On Reduction of Lagrange Systems

Valentin Irtegov and Tatyana Titorenko

Institute for System Dynamics and Control Theory SB RAS,
134, Lermontov str., Irkutsk, 664033, Russia
irteg@icc.ru

Abstract. We consider nonlinear conservative Lagrange systems with cyclic coordinates, which by means of the Legendre transformation are reduced to linear Routh systems. The latter allows one to reduce the problem of qualitative analysis for the nonlinear systems of above type to linear systems. Such an approach to investigation of the Lagrange systems is demonstrated by an example of a mechanical system with two cyclic coordinates and three positional coordinates. Some results of analysis of the initial system and the reduced one are given. We propose also a procedure of finding and investigation of qualitative properties of invariant manifolds (IMs) for the Lagrange systems with a nonlinear Routh function. The procedure is based on the analysis of stationary conditions of the "extended" Routh function. The efficiency of the proposed approach is demonstrated by an example of analysis of a concrete mechanical system.

Most part of the computations represented in this paper have been conducted with the aid of the computer algebra system "Mathematica".

1 Introduction

The method of reduction is well-known, mainly as a method of simplification in solving mechanical problems [1], problems of control theory [2], variational calculus [3] as well as problems of other areas of natural-scientific knowledge. This paper considers the conservative Lagrange systems with cyclic coordinates, which by means of the Legendre transformation are reduced to the linear Routh systems. It is shown that the inverse problem, which implies reconstruction of the lagrangian corresponding to the given Routh function, is solved algorithmically for these systems. In this case, the whole family of the Lagrange equations may be put in correspondence to the Routh equations. Within the framework of this consideration, the comparison of the sets of stationary solutions and invariant manifolds for the Routh problem and the corresponding Lagrange one is of interest. The comparison of qualitative properties of the corresponding stationary sets is a separate question.

2 On the Systems Reduced to Linear Ones

Let the lagrangian of a mechanical system with m cyclic and $n - m$ positional coordinates be the quadratic function with respect to the velocities:

$$2L = \sum_{i=1}^{n}\sum_{j=1}^{n} F_{ij}(q_{m+1},\ldots,q_n)\dot{q}_i\dot{q}_j - \sum_{i=m+1}^{n}\sum_{j=m+1}^{n} c_{ij}\, q_i q_j. \tag{1}$$

The Lagrange equations corresponding to (1) have the cyclic integrals, which we write in the form:

$$\sum_{j=1}^{m} F_{kj}\,(q_{m+1},\ldots,q_n)\dot{q}_j = p_k - \sum_{j=m+1}^{n} F_{kj}\,(q_{m+1},\ldots,q_n)\dot{q}_j,$$
$$p_k = const \quad (k = 1,\ldots,m). \tag{2}$$

Let us state the problem to define the functions $F_{ij}(q_{m+1},\ldots,q_n)$ $(i,j=1,\ldots,n)$ in expression (1) so that after the Legendre transformation [4] the Routh function (the reduced Lagrange function) corresponding to lagrangian (1) would become a quadratic one with constant coefficients.

For this purpose, find the cyclic velocities $\dot{q}_i$ $(i = 1,\ldots,m)$ from equations (2). It is possible in virtue of the assumption that system (2) is nondegenerate. Next we construct the Routh function

$$R = L - \sum_{k=1}^{m} p_k\dot{q}_k = -\frac{1}{2\Delta}\sum_{i=1}^{m}\sum_{j=1}^{m} a_{ij}\, p_i p_j + \frac{1}{\Delta}\sum_{k=m+1}^{n}\dot{q}_k\sum_{i=1}^{m} p_i\sum_{j=1}^{m} F_{jk}\, a_{ji}$$
$$+\frac{1}{2\Delta}\sum_{l=m+1}^{n}\dot{q}_l\sum_{k=m+1}^{n}\dot{q}_k\,[\Delta F_{lk} - \sum_{j=1}^{m} F_{jl}\sum_{r=1}^{m} F_{rk}\, a_{rj}]$$
$$-\frac{1}{2}\sum_{i=m+1}^{n}\sum_{j=m+1}^{n} c_{ij}\, q_i q_j, \tag{3}$$

which corresponds to the initial Lagrange function (1). Here $\Delta = det\|F_{kj}\|$ is the determinant of system (2), a_{ij} are algebraic complements of the elements F_{ij} in matrix $\|F_{ij}\|$ of system (2). In case when $m = 1$ we shall suppose that $a_{ij} = 1$ and $\Delta = F_{11}$.

The Routh function (3) will be quadratic with respect to the variables q_{m+1}, $\ldots, q_n, \dot{q}_{m+1},\ldots,\dot{q}_n$ when the following conditions

$$\frac{a_{ij}}{\Delta} = (\alpha_{ij\,m+1}q_{m+1} + \ldots + \alpha_{ij\,n}q_n)^2 + r_{ij} = \beta_{ij} \quad (i,j = 1,\ldots,m), \tag{4}$$

$$\sum_{j=1}^{m} F_{jk}\, a_{ji} = \Delta\sum_{l=m+1}^{n}\omega_{ikl}\, q_l = \Delta\, s_{ik} \quad (k = m+1,\ldots,n; i = 1,\ldots,m), \tag{5}$$

$$\frac{1}{\Delta}[\Delta F_{lk} - \sum_{j=1}^{m} F_{jl}\sum_{r=1}^{m} F_{rk}\, a_{rj}] = B_{lk} \quad (l = m+1,\ldots,n; k = m+1,\ldots,n) \tag{6}$$

hold. Here $\alpha_{ij\,k}$, B_{lk}, r_{ij}, and ω_{ikl} are some constants, β_{ij}, s_{ik} are the denotations of expressions in (4) and (5).

From equations (4)–(6) we can find the values of F_{ij}, what allows us to write down the explicit form of lagrangian (1) and, hence, to define the class of lagrangians of type (1), which can be reduced to quadratic ones with the aid of Legendre's transformation.

Having used the known relation between the elements of the second adjunct matrix and the elements of the initial matrix $A_{ij} = F_{ij}\varDelta^{m-2}$ as well as the expression for the determinant of the adjunct matrix

$$det\|a_{ij}\| = \varDelta^{m-1} \tag{7}$$

and (4), we can write F_{ij} $(i, j = 1, \ldots, m)$ as follows:

$$F_{ij} = \frac{A_{ij}}{\varDelta^{m-2}} = \frac{1}{\varDelta^{m-2}} D_{ij}||\varDelta\beta_{ij}|| = \frac{\varDelta^{m-1}}{\varDelta^{m-2}} D_{ij}||\beta_{kl}|| = \varDelta D_{ij}||\beta_{k\,l}||$$
$$(i, j = 1, \ldots, m). \tag{8}$$

Here, by $D_{ij}||.||$ we denote algebraic complements for the elements of the corresponding matrix.

An expression for $\varDelta$ can easily be obtained from (4) and (7). Indeed, according to (4) we have $a_{ij} = \varDelta\beta_{ij}$, and hence, $det\|a_{ij}\| = det\|\varDelta\beta_{ij}\| = \varDelta^m det\|\beta_{ij}\| = \varDelta^{m-1}$. From the latter we can find

$$\varDelta = \frac{1}{det\|\beta_{ij}\|}. \tag{9}$$

When solving equations (5) with respect to F_{kj} $(k = 1, \ldots, m; j = m+1, \ldots, n)$, we obtain the expressions for these coefficients:

$$F_{kj} = \sum_{i=1}^{m} F_{k\,i}\, s_{ij} \quad (k = 1, \ldots, m; j = m+1, \ldots, n). \tag{10}$$

Equations (6) allow one to find the expressions for F_{kl}:

$$F_{kl} = B_{kl} + \sum_{i=1}^{m} F_{il}\, s_{ik} \quad (k = m+1, \ldots, n; l = k, \ldots, n). \tag{11}$$

Hence, we have shown that system (1) with the lagrangian, whose coefficients F_{ij} are defined by formulas (8)–(11), can be reduced to the quadratic Routh function:

$$2R = \sum_{l=m+1}^{n} \dot{q}_l \sum_{k=m+1}^{n} B_{lk}\dot{q}_k + 2\sum_{k=m+1}^{n} \dot{q}_k \sum_{i=1}^{m} p_i s_{ik} - \sum_{i=1}^{m}\sum_{j=1}^{m} \beta_{ij}\, p_i p_j$$
$$- \sum_{i=m+1}^{n}\sum_{j=m+1}^{n} c_{ij}\, q_i q_j. \tag{12}$$

Formulas (8)–(12) can easily be implemented with the aid of some general-purpose computer algebra system, what allows one to construct the Routh function and the corresponding lagrangian for the systems of above class and of desired dimension. The latter gives the possibility to investigate properties of the reduced system and the specificities, which reveal in comparing properties of the initial equations and the reduced ones, for various models.

Now, we consider a mechanical system, the Lagrange function of which and the corresponding Routh function belong to the above class. The mechanical system has two cyclic coordinates ($m = 2$) and three positional coordinates ($n-m = 3$). We use formulas (8)–(12) to construct the lagrangian and the Routh function of this system.

According to formula (12), the Routh function writes:

$$\begin{aligned}2R = {}& 2(\omega_{133}q_3 + \omega_{134}q_4 + \omega_{135}q_5)p_1\dot q_3 + 2(\omega_{233}q_3 + \omega_{234}q_4 + \omega_{235}q_5)p_2\dot q_3 \\ &+B_{33}\dot q_3^2 + 2(\omega_{143}q_3 + \omega_{144}q_4 + \omega_{145}q_5)p_1\dot q_4 + 2(\omega_{243}q_3 + \omega_{244}q_4 + \omega_{245}q_5) \\ &\times p_2\dot q_4 + 2B_{34}\dot q_3\dot q_4 + B_{44}\dot q_4^2 + 2(\omega_{153}q_3 + \omega_{154}q_4 + \omega_{155}q_5)p_1\dot q_5 + 2(\omega_{253}q_3 \\ &+\omega_{254}q_4 + \omega_{255}q_5)p_2\dot q_5 + 2B_{35}\dot q_3\dot q_5 + 2B_{45}\dot q_4\dot q_5 + B_{55}\dot q_5^2 - [2((\alpha_{123}q_3 \\ &+\alpha_{124}q_4 + \alpha_{125}q_5)^2 + r_{12})p_1p_2 + ((\alpha_{113}q_3 + \alpha_{114}q_4 + \alpha_{115}q_5)^2 + r_{11})p_1^2 \\ &+((\alpha_{223}q_3 + \alpha_{224}q_4 + \alpha_{225}q_5)^2 + r_{22})p_2^2] - (c_{33}q_3^2 + c_{44}q_4^2 + c_{55}q_5^2).\end{aligned} \tag{13}$$

Here q_1, q_2 are the cyclic coordinates, q_3, q_4, and q_5 are the positional coordinates, $\beta_{11} = (\alpha_{113}q_3+\alpha_{114}q_4+\alpha_{115}q_5)^2+r_{11},\ \beta_{12} = \beta_{21} = (\alpha_{123}q_3+\alpha_{124}q_4+\alpha_{125}q_5)^2+r_{12},\ \beta_{22} = (\alpha_{223}q_3+\alpha_{224}q_4+\alpha_{225}q_5)^2+r_{22},\ s_{13} = \omega_{133}q_3+\omega_{134}q_4+\omega_{135}q_5,\ s_{14} = \omega_{143}q_3+\omega_{144}q_4+\omega_{145}q_5,\ s_{15} = \omega_{153}q_3+\omega_{154}q_4+\omega_{155}q_5,\ s_{23} = \omega_{233}q_3+\omega_{234}q_4+\omega_{235}q_5,\ s_{24} = \omega_{243}q_3 + \omega_{244}q_4 + \omega_{245}q_5,\ s_{25} = \omega_{253}\omega_3 + \omega_{254}q_4 + \omega_{255}q_5.$

We shall seek the Lagrange function corresponding to the Routh function (13) in the form:

$$\begin{aligned}2L = {}& F_{11}\dot q_1^2 + 2F_{12}\dot q_1\dot q_2 + F_{22}\dot q_2^2 + 2F_{13}\dot q_1\dot q_3 + 2F_{23}\dot q_2\dot q_3 + F_{33}\dot q_3^2 + 2F_{14}\dot q_1\dot q_4 \\ &+2F_{24}\dot q_2\dot q_4 + 2F_{34}\dot q_3\dot q_4 + F_{44}\dot q_4^2 + 2F_{15}\dot q_1\dot q_5 + 2F_{25}\dot q_2\dot q_5 + 2F_{35}\dot q_3\dot q_5 \\ &+2F_{45}\dot q_4\dot q_5 + F_{55}\dot q_5^2 - (c_{33}q_3^2 + c_{44}q_4^2 + c_{55}q_5^2).\end{aligned}$$

According to (9), $\Delta = [((\alpha_{113}q_3 + \alpha_{114}q_4 + \alpha_{115}q_5)^2 + r_{11})((\alpha_{223}q_3 + \alpha_{224}q_4 + \alpha_{225}q_5)^2 + r_{22}) - ((\alpha_{123}q_3 + \alpha_{124}q_4 + \alpha_{125}q_5)^2 + r_{12})^2]^{-1}$.

Using formulas (8) we can write down the expressions for $F_{11} = \Delta((\alpha_{223}q_3 + \alpha_{224}q_4 + \alpha_{225}q_5)^2 + r_{22}), F_{12} = -\Delta((\alpha_{123}q_3 + \alpha_{124}q_4 + \alpha_{125}q_5)^2 + r_{12}), F_{22} = \Delta((\alpha_{113}q_3 + \alpha_{114}q_4 + \alpha_{115}q_5)^2 + r_{11})$. The rest of the coefficients F_{ij} may easily be expressed via the obtained ones with the use of (10), (11).

As a result, we obtain the following Lagrange function:

$$\begin{aligned}2L = {}& F_{11}\dot q_1^2 + 2F_{12}\dot q_1\dot q_2 + F_{22}\dot q_2^2 + 2(s_{13}F_{11} + s_{23}F_{12})\dot q_1\dot q_3 + 2(s_{13}F_{12} + s_{23}F_{22}) \\ &\times\dot q_2\dot q_3 + [B_{33} + s_{13}^2F_{11} + 2s_{13}s_{23}F_{12} + s_{23}^2F_{22}]\dot q_3^2 + 2(s_{14}F_{11} + s_{24}F_{12})\dot q_1\dot q_4 \\ &+2(s_{14}F_{12} + s_{24}F_{22})\dot q_2\dot q_4 + 2[B_{34} + s_{13}s_{14}F_{11} + (s_{14}s_{23} + s_{13}s_{24})F_{12} + s_{23} \\ &\times s_{24}F_{22}]\dot q_3\dot q_4 + [B_{44}+s_{14}^2F_{11}+2s_{14}s_{24}F_{12}+s_{24}^2F_{22}]\dot q_4^2 + 2(s_{15}F_{11}+s_{25}F_{12}) \\ &\times\dot q_1\dot q_5 + 2(s_{15}F_{12} + s_{25}F_{22})\dot q_2\dot q_5 + 2[B_{35} + s_{13}s_{15}F_{11} + (s_{15}s_{23} + s_{13}s_{25})F_{12} \\ &+s_{23}s_{25}F_{22}]\dot q_3\dot q_5 + 2[B_{45} + s_{14}s_{15}F_{11} + (s_{15}s_{24} + s_{14}s_{25})F_{12} + s_{24}s_{25}F_{22}] \\ &\times\dot q_4\dot q_5 + [B_{55} + s_{15}^2F_{11} + 2s_{15}s_{25}F_{12} + s_{25}^2F_{22}]\dot q_5^2 - c_{33}q_3^2 - c_{44}q_4^2 - c_{55}q_5^2.\end{aligned} \tag{14}$$

For the purpose of further investigation, we have constructed the Routh equations of motion, which correspond to the Routh function (13), with the aid of the software package [5]. These write:

$$\begin{aligned}
&B_{33}\ddot{q}_3 + B_{34}\ddot{q}_4 + B_{35}\ddot{q}_5 + A_1 p_1\dot{q}_4 + A_2 p_2\dot{q}_4 + A_3 p_1\dot{q}_5 + A_4 p_2\dot{q}_5 + c_{33}q_3\\
&+2\alpha_{123}(\alpha_{123}q_3 + \alpha_{124}q_4 + \alpha_{125}q_5)p_1p_2 + \alpha_{113}(\alpha_{113}q_3 + \alpha_{114}q_4 + \alpha_{115}q_5)p_1^2\\
&+\alpha_{223}(\alpha_{223}q_3 + \alpha_{224}q_4 + \alpha_{225}q_5)p_2^2 = 0,\\
&B_{34}\ddot{q}_3 + B_{44}\ddot{q}_4 + B_{45}\ddot{q}_5 - A_1 p_1\dot{q}_3 - A_2 p_2\dot{q}_3 + A_5 p_1\dot{q}_5 + A_6 p_2\dot{q}_5 + c_{44}q_4\\
&+2\alpha_{124}(\alpha_{123}q_3 + \alpha_{124}q_4 + \alpha_{125}q_5)p_1p_2 + \alpha_{114}(\alpha_{113}q_3 + \alpha_{114}q_4 + \alpha_{115}q_5)p_1^2\\
&+\alpha_{224}(\alpha_{223}q_3 + \alpha_{224}q_4 + \alpha_{225}q_5)p_2^2 = 0,\\
&B_{35}\ddot{q}_3 + B_{45}\ddot{q}_4 + B_{55}\ddot{q}_5 - A_3 p_1\dot{q}_3 - A_4 p_2\dot{q}_3 - A_5 p_1\dot{q}_4 - A_6 p_2\dot{q}_4 + c_{55}q_5\\
&+2\alpha_{125}(\alpha_{123}q_3 + \alpha_{124}q_4 + \alpha_{125}q_5)p_1p_2 + \alpha_{115}(\alpha_{113}q_3 + \alpha_{114}q_4 + \alpha_{115}q_5)p_1^2\\
&+\alpha_{225}(\alpha_{223}q_3 + \alpha_{224}q_4 + \alpha_{225}q_5)p_2^2 = 0.
\end{aligned} \tag{15}$$

Here $A_1 = \omega_{134} - \omega_{143}, A_2 = \omega_{234} - \omega_{243}, A_3 = \omega_{135} - \omega_{153}, A_4 = \omega_{235} - \omega_{253}, A_5 = \omega_{145} - \omega_{154}, A_6 = \omega_{245} - \omega_{254}$.

Note, the equations of motion obtained, unlike the case of the Routh function (13), do not contain the parameters $\omega_{133}, \omega_{144}, \omega_{155}$, $\omega_{233}, \omega_{244}$, ω_{255}. This is the well-known fact, which means the following: in case of adding the full differential to the integrand in the variational principle the corresponding Euler–Lagrange equations do not change. It is substantial here that above parameters not only enter the Lagrange function (14) but also the corresponding Lagrange equations. We shall call the Routh function the "extended" one if it contains the parameters which do not enter the Routh equations.

2.1 Invariant Sets of the Initial and Reduced System

Let us present some results of analysis related to the initial system and the reduced one for the problem with the characteristic function (14).

Lagrangian (14) is a family of lagrangians parametrized by $\omega_{133}, \omega_{144}, \omega_{155}$, $\omega_{233}, \omega_{244}, \omega_{255}$ (this function has 39 parameters on the whole). After the reduction, each lagrangian of this family with the cyclic integrals

$$\begin{aligned}
V_1 = F_{11}\dot{q}_1 + F_{12}\dot{q}_2 + (s_{13}F_{11} + s_{23}F_{12})\dot{q}_3 + (s_{14}F_{11} + s_{24}F_{12})\dot{q}_4 + (s_{15}F_{11}\\
+s_{25}F_{12})\dot{q}_5 = p_1,\ V_2 = F_{12}\dot{q}_1 + F_{22}\dot{q}_2 + (s_{13}F_{12} + s_{23}F_{22})\dot{q}_3 + (s_{14}F_{12}\\
+s_{24}F_{22})\dot{q}_4 + (s_{15}F_{12} + s_{25}F_{22})\dot{q}_5 = p_2
\end{aligned} \tag{16}$$

corresponds to a family of quadratic Routh functions (13) with the parameters p_1, p_2.

Now, we show that differential equations (15) have a family of IMs with the parameters $\omega_{133}, \omega_{144}, \omega_{155}, \omega_{233}, \omega_{244}, \omega_{255}$, and these manifolds may be obtained algorithmically.

To this end, we write down the stationary conditions R with respect to $\dot q_3, q_3, \dot q_4, q_4, \dot q_5, q_5$:

$$\begin{aligned}
\partial R/\partial \dot q_3 &= B_{33}\dot q_3 + B_{34}\dot q_4 + B_{35}\dot q_5 + D_1 q_3 + D_2 q_4 + D_3 q_5 = 0,\\
\partial R/\partial q_3 &= D_1\dot q_3 + D_4\dot q_4 + D_5\dot q_5 - (c_{33} + \alpha_{113}^2 p_1^2 + 2\alpha_{123}^2 p_1p_2 + \alpha_{223}^2 p_2^2)q_3\\
&\quad -(\alpha_{113}\alpha_{114}p_1^2 + 2\alpha_{123}\alpha_{124}p_1p_2 + \alpha_{223}\alpha_{224}p_2^2)q_4 - (\alpha_{113}\alpha_{115}p_1^2\\
&\quad +2\alpha_{123}\alpha_{125}p_1p_2 + \alpha_{223}\alpha_{225}p_2^2)q_5 = 0,\\
\partial R/\partial \dot q_4 &= B_{34}\dot q_3 + B_{44}\dot q_4 + B_{45}\dot q_5 + D_4 q_3 + D_6 q_4 + D_7 q_5 = 0,\\
\partial R/\partial q_4 &= D_2\dot q_3 + D_6\dot q_4 + D_8\dot q_5 - (c_{44} + \alpha_{114}^2 p_1^2 + 2\alpha_{124}^2 p_1p_2 + \alpha_{224}^2 p_2^2)q_4\\
&\quad -(\alpha_{113}\alpha_{114}p_1^2 + 2\alpha_{123}\alpha_{124}p_1p_2 + \alpha_{223}\alpha_{224}p_2^2)q_3 - (\alpha_{114}\alpha_{115}p_1^2\\
&\quad +2\alpha_{124}\alpha_{125}p_1p_2 + \alpha_{224}\alpha_{225}p_2^2)q_5 = 0,\\
\partial R/\partial \dot q_5 &= B_{35}\dot q_3 + B_{45}\dot q_4 + B_{55}\dot q_5 + D_5 q_3 + D_8 q_4 + D_9 q_5 = 0,\\
\partial R/\partial q_5 &= D_3\dot q_3 + D_7\dot q_4 + D_9\dot q_5 - (c_{55} + \alpha_{115}^2 p_1^2 + 2\alpha_{125}^2 p_1p_2 + \alpha_{225}^2 p_2^2)q_5\\
&\quad -(\alpha_{113}\alpha_{115}p_1^2 + 2\alpha_{123}\alpha_{125}p_1p_2 + \alpha_{223}\alpha_{225}p_2^2)q_3 - (\alpha_{114}\alpha_{115}p_1^2\\
&\quad +2\alpha_{124}\alpha_{125}p_1p_2 + \alpha_{224}\alpha_{225}p_2^2)q_4 = 0. \qquad (17)
\end{aligned}$$

From now on we employ the following denotations: $D_1 = p_1\omega_{133} + p_2\omega_{233}, D_2 = p_1\omega_{134} + p_2\omega_{234}, D_3 = p_1\omega_{135} + p_2\omega_{235}, D_4 = p_1\omega_{143} + p_2\omega_{243}, D_5 = p_1\omega_{153} + p_2\omega_{253}, D_6 = p_1\omega_{144} + p_2\omega_{244}, D_7 = p_1\omega_{145} + p_2\omega_{245}, D_8 = p_1\omega_{154} + p_2\omega_{254}, D_9 = p_1\omega_{155} + p_2\omega_{255}$.

Having equated the determinant of system (17) to zero we can obtain the conditions, under which equations (15) have IMs of interest for us. The system considered is a multiparameter one, and in spite of its small dimension, the determinant of this system is a rather bulky expression, whose complete analysis is complicated. In order to simplify computations it is possible to solve the inverse problem: for a given solution it is necessary to find the conditions imposed on the parameters, under which this solution satisfies equations (17).

One of the solutions possible will be sought in the form:

$$q_3 = x\,\dot q_3,\ \ q_4 = y\,\dot q_4,\ \ q_5 = z\,\dot q_5, \qquad (18)$$

where x, y, z are the coefficients to be determined.

Having removed the variables q_3, q_4, and q_5 from equations (17) with the aid of (18), as a result, we obtain a system of linear equations with respect to $\dot q_3, \dot q_4, \dot q_5$. The conditions, under which the coefficients of these variables vanish, are the desired conditions. These write:

$$\begin{aligned}
&x = -\frac{B_{33}}{D_1},\ y = -\frac{B_{44}}{D_6},\ z = -\frac{B_{55}}{D_9},\\
&B_{34}D_1D_6 = -B_{33}B_{44}Q_1,\ B_{35}D_1D_9 = -B_{33}B_{55}Q_2,\ B_{45}D_6D_9 = -B_{44}B_{55}Q_3,\\
&c_{33} = -\frac{D_1^2}{B_{33}} + Q_4,\ c_{44} = -\frac{D_6^2}{B_{44}} + Q_5, c_{55} = -\frac{D_9^2}{B_{55}} + Q_6,\\
&D_1D_2 = -B_{33}Q_1,\ D_1D_3 = -B_{33}Q_2,\ D_1D_6 = -B_{44}Q_1,\\
&D_6D_7 = -B_{44}Q_3,\ D_5D_9 = -B_{55}Q_2,\ D_8D_9 = -B_{55}Q_3.
\end{aligned}$$

Here $Q_1 = \alpha_{113}\alpha_{114}p_1^2 + (2\alpha_{123}\alpha_{124}p_1 + \alpha_{223}\alpha_{224}p_2)p_2$, $Q_2 = \alpha_{113}\alpha_{115}p_1^2 + (2\alpha_{123}\alpha_{125}p_1 + \alpha_{223}\alpha_{225}p_2)p_2$, $Q_3 = \alpha_{114}\alpha_{115}p_1^2 + (2\alpha_{124}\alpha_{125}p_1 + \alpha_{224}\alpha_{225}p_2)p_2$, $Q_4 = \alpha_{113}^2 p_1^2 + (2\alpha_{123}^2 p_1 + \alpha_{223}^2 p_2)p_2$, $Q_5 = \alpha_{114}^2 p_1^2 + (2\alpha_{124}^2 p_1 + \alpha_{224}^2 p_2)p_2$, $Q_6 = \alpha_{115}^2 p_1^2 + (2\alpha_{125}^2 p_1 + \alpha_{225}^2 p_2)p_2$.

Under above conditions, equations (18) write:

$$D_1\, q_3 + B_{33}\dot{q}_3 = 0,\ D_6\, q_4 + B_{44}\dot{q}_4 = 0,\ D_9\, q_5 + B_{55}\dot{q}_5 = 0. \tag{19}$$

By using the definition of IM [6] we can easily verify that the found solution is a family of IMs with the parameters $\omega_{133}, \omega_{144}, \omega_{155}, \omega_{233}, \omega_{244}, \omega_{255}$. This family can be called stationary because it attributes a stationary value to the characteristic function R of the system.

If we rewrite equations (15) in terms of deviations from the elements of the above family of IMs, while introducing the variables $y_1 = D_1\, q_3 + B_{33}\dot{q}_3,\ y_2 = D_6\, q_4 + B_{44}\dot{q}_4,\ y_3 = D_9\, q_5 + B_{55}\dot{q}_5$, then we can obtain the following equations of perturbed motion, and, respectively, the Routh function:

$$\begin{aligned}
&B_{33}D_1^2\dot{y}_1 - B_{33}^2Q_1\dot{y}_2 - B_{33}^2Q_2\dot{y}_3 - D_1^3y_1 + B_{33}D_1Q_1y_2 + B_{33}D_1Q_2y_3 = 0,\\
&B_{44}^2Q_1\dot{y}_1 - B_{44}D_6^2\dot{y}_2 + B_{44}^2Q_3\dot{y}_3 - B_{44}D_6Q_1y_1 + D_6^3y_2 - B_{44}D_6Q_3y_3 = 0,\\
&B_{55}^2Q_2\dot{y}_1 + B_{55}^2Q_3\dot{y}_2 - B_{55}D_9^2\dot{y}_3 - B_{55}D_9Q_2y_1 - B_{55}D_9Q_3y_2 + D_9^3y_3 = 0,
\end{aligned} \tag{20}$$

$$\begin{aligned}
2R = {}&\frac{D_1^2}{B_{33}}y_1^2 - 2Q_1y_1y_2 + \frac{D_6^2}{B_{44}}y_2^2 - 2Q_2y_1y_3 - 2Q_3y_2y_3 + \frac{D_9^2}{B_{55}}y_3^2\\
&-(r_{11}p_1^2 + 2r_{12}p_1p_2 + r_{12}p_2^2).
\end{aligned}$$

Having considered the expression $\bar{R} = 2R + r_{11}p_1^2 + 2r_{12}p_1p_2 + r_{12}p_2^2$ as a Lyapunov function and computed the derivative due to differential equations (20) for this expression, we obtain

$$\begin{aligned}
\frac{d\bar{R}}{dt} = {}&\frac{D_1^3}{B_{33}^2}y_1^2 - \frac{Q_1(B_{44}D_1 + B_{33}D_6)}{B_{33}B_{44}}y_1y_2 + \frac{D_6^3}{B_{44}^2}y_2^2 - \frac{Q_2(B_{55}D_1 + B_{33}D_9)}{B_{33}B_{55}}y_1y_3\\
&-\frac{Q_3(B_{55}D_6 + B_{44}D_9)}{B_{44}B_{55}}y_2y_3 + \frac{D_9^3}{B_{55}^2}y_3^2.
\end{aligned}$$

When $D_1 < 0 \wedge D_6 < 0 \wedge D_9 < 0 \wedge Q_1 = 0 \wedge Q_2 = 0 \wedge Q_3 = 0 \wedge B_{55} > 0 \wedge B_{44} > 0 \wedge B_{33} > 0$ the quadratic form $\bar{R}$ is positive definite, and its derivative $d\bar{R}/dt$ is negative definite. Hence, due to the well-known theorem of Lyapunov's second method, we conclude that elements of the investigated family of IMs are asymptotically stable under above conditions.

It can easily be verified that elements of the family of IMs (19) are "brought up into" the phase spaces of Lagrange systems related to the Lagrange function (14). In this case, for obtaining the IMs equations it is necessary to add equations (16) to equations of IMs (19).

3 On Stationary Sets of a System with a Nonlinear Routh Function

Here we consider an example of a mechanical system with a nonlinear Routh function, for the analysis of which the above technique is used.

Consider the Lagrange problem related to the motion of a rigid body with a fixed point in a central field of forces (Tisserand's approximation). Here the Lagrange function writes:

$$\begin{aligned}2L &= A\dot{\Theta}^2 + \dot{\psi}^2(A\sin^2\Theta + C\cos^2\Theta) + C(\dot{\varphi}^2 + \dot{\varphi}\dot{\psi}\cos\Theta) - 2z_0\cos\Theta \\ &\quad -\mu(A\sin^2\Theta + C\cos^2\Theta).\end{aligned} \tag{21}$$

The equations of motion admit the following cyclic integrals:

$$\begin{aligned}\frac{\partial L}{\partial\dot{\psi}} &= \dot{\psi}(A\sin^2\Theta + C\cos^2\Theta) + C\dot{\varphi}\cos\Theta = p_2, \\ \frac{\partial L}{\partial\dot{\varphi}} &= C(\dot{\varphi} + \dot{\psi}\cos\Theta) = p_1.\end{aligned}$$

Find $\dot{\psi}, \dot{\varphi}$ from the latter equations

$$\dot{\psi} = \frac{(p_2 - p_1\cos\Theta)}{A\sin^2\Theta}, \quad \dot{\varphi} = \frac{p_1(A\sin^2\Theta + C\cos^2\Theta) - p_2\cos\Theta}{AC\sin^2\Theta}$$

and construct the Routh function

$$\begin{aligned}R = L - p_1\dot{\varphi} - p_2\dot{\psi} &= \frac{1}{2}A\dot{\Theta}^2 - z_0\cos\Theta - \frac{1}{2}\mu(A\sin^2\Theta + C\cos^2\Theta) \\ &\quad -\frac{p_1^2}{2C} - \frac{(p_2 - p_1\cos\Theta)^2}{2A\sin^2\Theta}\end{aligned}$$

with the aid of the Legendre transformation.

Having added to R the full derivative, we obtain the "extended" Routh function

$$\tilde{R} = R + m\dot{\Theta}f(\Theta), \quad m = const.$$

Obviously, the latter does not change the Routh equations.

Let us write down the stationary conditions for $\tilde{R}$:

$$\begin{aligned}\frac{\partial\tilde{R}}{\partial\dot{\Theta}} &= A\dot{\Theta} + mf(\Theta) = 0, \\ \frac{\partial\tilde{R}}{\partial\Theta} &= m\dot{\Theta}f'(\Theta) + z_0\sin\Theta - \frac{1}{2}\mu(A - C)\sin 2\Theta \\ &\quad + \frac{(p_2 - p_1\cos\Theta)(p_2\cos\Theta - p_1)}{A\sin^3\Theta} = 0.\end{aligned}$$

Next, we require that the latter equations be dependent. For this purpose we exclude $\dot{\Theta}$ from the 2nd equation with the aid of the first one. As a result, we obtain the following condition of degeneration

$$z_0 \sin \Theta - \frac{1}{2}\mu(A-C)\sin 2\Theta + \frac{(p_2 - p_1\cos\Theta)(p_2\cos\Theta - p_1)}{A\sin^3\Theta}$$
$$-\frac{m^2}{A} f(\Theta) f'(\Theta) = 0 \tag{22}$$

for the system.

We consider this relation as a differential equation with respect to $f(\Theta)$. It can be rewritten as

$$\frac{df^2(\Theta)}{d\Theta} = \frac{2A}{m^2}\Big(z_0 \sin\Theta - \frac{1}{2}\mu(A-C)\sin 2\Theta + \frac{(p_2 - p_1\cos\Theta)(p_2\cos\Theta - p_1)}{A\sin^3\Theta}\Big).$$

After its integration we have:

$$f^2(\Theta) = \frac{2A}{m^2}(-z_0\cos\Theta - \frac{1}{4}\mu(A-C)\cos 2\Theta + \frac{(2p_2p_1\cos\Theta - p_1^2 - p_2^2)}{2A\sin^2\Theta}) + D,$$
$$D = const. \tag{23}$$

Under this value of $f(\Theta)$, the Routh equation

$$\frac{d}{dt}(\frac{\partial \tilde{R}}{\partial \dot{\Theta}}) - \frac{\partial \tilde{R}}{\partial \Theta} = A\ddot{\Theta} - z_0 \sin\Theta + \frac{1}{2}\mu(A-C)\cos 2\Theta$$
$$-\frac{(p_2 - p_1\cos\Theta)(p_2\cos\Theta - p_1)}{A\sin^3\Theta} = 0, \tag{24}$$

which is the same for R and $\tilde{R}$, as can easily be verified according to the definition, has the family of IMs:

$$A\dot{\Theta} + mf(\Theta) = 0 \tag{25}$$

with parameters m and D. These parameters are not contained in the Routh equation.

Let us reconstruct the Lagrange function corresponding to the "extended" function $\tilde{R}$ according to the algorithm of section 2:

$$2\tilde{L} = (A + \frac{m^2 f^2(\Theta) A\sin^2\Theta}{(p_2 - p_1\cos\Theta)^2})\dot{\Theta}^2 + 2\frac{mf(\Theta)A\sin^2\Theta}{(p_2 - p_1\cos\Theta)}\dot{\Theta}\dot{\psi} + (A\sin^2\Theta$$
$$+C\cos^2\Theta)\dot{\psi}^2 + C\dot{\varphi}^2 + 2C\cos\Theta\dot{\psi}\dot{\varphi} - 2z_0\cos\Theta - \mu(A\sin^2\Theta + C\cos^2\Theta).$$

The cyclic integrals corresponding to $\tilde{L}$ write:

$$\frac{\partial \tilde{L}}{\partial \dot{\psi}} = \frac{mf(\Theta)A\sin^2\Theta}{(p_2 - p_1\cos\Theta)}\dot{\Theta} + \dot{\psi}(A\sin^2\Theta + C\cos^2\Theta) + C\dot{\varphi}\cos\Theta = p_2,$$
$$\frac{\partial \tilde{L}}{\partial \dot{\varphi}} = C(\dot{\varphi} + \dot{\psi}\cos\Theta) = p_1. \tag{26}$$

Differential equations for $\tilde{L}$ have the form:

$$\Big(A+\frac{m^2 f^2(\Theta)A\sin^2\Theta}{(p_2-p_1\cos\Theta)^2}\Big)\ddot{\Theta}+2\frac{mf(\Theta)A\sin^2\Theta}{(p_2-p_1\cos\Theta)}\ddot{\psi}+\frac{m^2 f(\Theta)A\sin\Theta}{(p_2-p_1\cos\Theta)^3}$$
$$\times\big[(f'(\Theta)\sin\Theta+f(\Theta)\cos\Theta)(p_2-p_1\cos\Theta)+f(\Theta)p_1\sin^2\Theta\big]\dot{\Theta}^2$$
$$-\frac{1}{2}(A-C)\sin 2\Theta\dot{\psi}^2+C\sin\Theta\dot{\psi}\dot{\varphi}-z_0\sin\Theta+\frac{1}{2}\mu(A-C)\sin 2\Theta=0,$$

$$\frac{mf(\Theta)A\sin^2\Theta}{(p_2-p_1\cos\Theta)})\ddot{\Theta}+A\sin^2\Theta\ddot{\psi}+\frac{mA\sin\Theta}{(p_2-p_1\cos\Theta)^2}\big[(f'(\Theta)\sin\Theta+2f(\Theta)\cos\Theta)$$
$$\times(p_2-p_1\cos\Theta)-mp_1 f(\Theta)\sin^2\Theta\big]\dot{\Theta}^2+A\sin 2\Theta\dot{\Theta}\dot{\psi}-p_1\dot{\Theta}\sin\Theta=0. \quad (27)$$

These contain parameter m, which does not enter Routh equation (24). The second equation has been obtained here by means of differentiation of the first cyclic integral (26).

After reducing to the normal form the first equation (27) writes:

$$\ddot{\Theta}=\frac{\sin\Theta}{A}\Big[A\cos\Theta\Big(\frac{mf(\Theta)\dot{\Theta}}{p_2-p_1\cos\Theta}+\dot{\psi}\Big)^2-p_1\Big(\frac{mf(\Theta)\dot{\Theta}}{p_2-p_1\cos\Theta}+\dot{\psi}\Big)$$
$$+z_0-\mu(A-C)\cos\Theta\Big].$$

Now, let us verify according to the definition of IM the fact that Lagrange equations in our case assume the family of invariant manifolds:

$$A\dot{\Theta}+mf(\Theta)=0,\ \frac{\partial\tilde{L}}{\partial\dot{\varphi}}=C(\dot{\varphi}+\dot{\psi}\cos\Theta)=p_1,$$
$$\frac{\partial L}{\partial\dot{\psi}}=A\sin^2\Theta\Big(\frac{mf(\Theta)\dot{\Theta}}{p_2-p_1\cos\Theta}+\dot{\psi}\Big)+p_1\cos\Theta=p_2. \quad (28)$$

To this end, we compute the derivative of the first expression (28) due to the Lagrange differential equations:

$$\frac{d}{dt}(A\dot{\Theta}+mf(\Theta))=A\ddot{\Theta}+mf'(\Theta)\dot{\Theta}=\sin\Theta[A\cos\Theta\Big(\frac{mf(\Theta)\dot{\Theta}}{p_2-p_1\cos\Theta}+\dot{\psi}\Big)^2$$
$$-p_1\Big(\frac{mf(\Theta)\dot{\Theta}}{p_2-p_1\cos\Theta}+\dot{\psi}\Big)+z_0-\mu(A-C)\cos\Theta]+mf'(\Theta)\dot{\Theta}. \quad (29)$$

Having excluded $\dot{\Theta}$, $\dot{\psi}$ from expression (29) with the aid of equations (28), we may rewrite (29) as

$$\frac{\cos\Theta}{A\sin^3\Theta}(p_2-p_1\cos\Theta)^2-\frac{p_1}{A\sin\Theta}(p_2-p_1\cos\Theta)+z_0\sin\Theta$$
$$-\mu(A-C)\cos\Theta\sin\Theta-\frac{1}{A}m^2 f'(\Theta)f(\Theta).$$

Next, we replace the addend $A^{-1}m^2 f'(\Theta)f(\Theta)$ with the expression obtained from (22). As a result, we find that the latter expression becomes identically zero.

Hence we have shown that Lagrange differential equations in our case possess the family of IMs defined by equations (28). Therefore, the family of IMs obtained for the Routh equation (24) corresponds to the family of IMs (28) for the Lagrange equations with the characteristic function $\tilde{L}$. Note that equations (28) are obtained when we add the cyclic integrals (26) to the equation of IMs (25). It can easily be verified that the family of manifolds (28) is invariant also for the Lagrange equations corresponding to lagrangian (21) when $f(\Theta)$ is defined by (23).

4 Conclusion

We have considered a class of the conservative Lagrange systems with cyclic coordinates, which are reduced to the linear Routh systems with the aid of the Legendre transformation. The latter allows one to reduce the problem of qualitative analysis for the nonlinear systems of above type to linear systems. An algorithm allowing one to reconstruct the lagrangian for the given Routh function has been proposed.

In the paper, the proposed approach to investigation of the Lagrange systems has been demonstrated by an example of a mechanical system with two cyclic coordinates and three positional coordinates. Some results of analysis of the initial system and the reduced one have been given. In particular, the invariant manifolds of the reduced system, which attribute a stationary value to the "extended" Routh function of this system and possess the property of asymptotic stability, have been obtained. The comparison of IMs and their qualitative properties both for the Routh system and for the initial one has been conducted.

Using the above technique, we have conducted an analysis of a mechanical system with a nonlinear Routh function. The procedure of obtaining and investigation of qualitative properties of IMs both for the Routh system and the Lagrange system have been applied. This procedure is based on the analysis of stationary conditions of the "extended" Routh function.

References

1. Borisov, A.V., Mamayev, I.S.: Poisson Structures and Lie Algebras in Hamiltonian Mechanics. Udmurdsk University, Izhevsk (1999)
2. Elkin, V.I.: Reduction of Nonlinear Control Systems. FAZIS-Computing Center of RAS, Moscow (2003)
3. Griffits, F.: External Differential Forms and Variational Calculus. NFMI, Novosibirsk (1999)
4. Lurier, A.I.: Analytical Mechanics. GIFML, Moscow (1961)
5. Irtegov, V.D., Titorenko, T.N.: Using the system "Mathematica" in problems of mechanics. Mathematics and Computers in Simulation 3-5(57), 227–237 (2001)
6. Olver, P.J.: Applications of Lie Groups to Differential Equations, 2nd edn. Springer, New York (1993)

Series Transformations to Improve and Extend Convergence

G.A. Kalugin and D.J. Jeffrey

The University of Western Ontario, Department of Applied Mathematics, London, Ontario, Canada

Abstract. We consider a new invariant transformation of some previously known series for the Lambert W function. The transformations contain a parameter p which can be varied, while retaining the basic series structure. The parameter can be used to expand the domain of convergence of the series. The speed of convergence, that is the accuracy for a given number of terms, can increase or decrease with p. Theoretical and experimental investigations that rely heavily on the computer-algebra system MAPLE are described.

1 Introduction

The Lambert W function is the inverse of the mapping $z \mapsto ze^z$. The inverse is a multivalued function denoted W_k, and the branches of this multivalued function are fixed by defining W_k through the equations [1]

$$\forall z \in \mathbb{C}, W_k(z)\exp(W_k(z)) = z \ , \tag{1}$$

$$W_k(z) \to \ln_k z \text{ for } \Re z \to \infty \ . \tag{2}$$

Here, $\ln_k z$ is the kth branch of logarithm, namely $\ln_k z = \ln z + 2\pi i k$, where $\ln z$ is the principal branch of natural logarithm [2]. Lambert W and its branches are important in the study of delay-differential equations. The simplest delay equation, using the notation $\dot{y} = \frac{dy}{dt}$ for the derivative with respect to time, is

$$\dot{y}(t) = ay(t-1) \ ,$$

subject to the condition on $-1 \le t \le 0$ that $y(t) = f(t)$, a known function. The solution can be expressed as the sum [3]

$$y(t) = \sum_{k=-\infty}^{\infty} c_k \, \exp\left(W_k(a)t\right) \ ,$$

where the c_k can be determined from the initial conditions. One sees immediately that the solution will grow exponentially if any of the $W_k(a)$ has a positive real part, which leads to important stability theorems in the theory of delay equations. Other applications are given in [1].

In this paper, we use the computer-algebra system MAPLE to investigate the properties of series expansions for W. We focus on a number of asymptotic expansions for large z; these are also valid for non-principal branches around $z = 0$. One practical application of the series is to provide initial estimates for the numerical evaluation of W; these estimates can then be refined using iterative schemes to provide the arbitrary precision computations used in computer algebra systems. The series also have intrinsic interest. For example, the definition above of the branches W_k is based on partitioning the plane using the asymptotic series. Another interest is the fact that the asymptotic series are also convergent, and the nature of the convergence is one particular interest of this paper. In this paper, we shall mostly be concerned with the principal branch $k = 0$, which is the only branch that is finite at the origin. We shall abbreviate W_0 to W for the rest of the article.

The first asymptotic series is that found by de Bruijn [4] and Comtet [5] as

$$W(z) = \ln z - \ln\ln z + u \ , \tag{3}$$

where u has the series development

$$u = \sum_{n=1}^{\infty}\sum_{m=1}^{n}(-1)^{n-m}\left[\begin{matrix} n \\ n-m+1\end{matrix}\right]\frac{\sigma^{n-m}\tau^m}{m!} \ , \tag{4}$$

where $\sigma = 1/\ln z$ and $\tau = \ln\ln z/\ln z$, and where $\left[\begin{smallmatrix} n \\ n-m+1\end{smallmatrix}\right]$ denotes Stirling Cycle Numbers, also called the unsigned Stirling numbers of the first kind [6,7]. This series was rearranged in [8] by introducing the new variable $\zeta = 1/(1+\sigma)$ to get

$$u = \sum_{m=1}^{\infty}\frac{\tau^m}{m!}\sum_{k=0}^{m-1}\left\{\begin{matrix} k+m-1 \\ k\end{matrix}\right\}_{\geq 2}(-1)^{k+m-1}\zeta^{k+m} \ , \tag{5}$$

where the 2-associated Stirling Subset Numbers [6,7] appear. Two further expansions introduce the variables $L_\tau = \ln(1-\tau)$ and $\eta = \sigma/(1-\tau)$.

$$u = -L_\tau + \sum_{n=1}^{\infty}(-\eta)^n\sum_{m=1}^{n}(-1)^m\left[\begin{matrix} n \\ n-m+1\end{matrix}\right]\frac{L_\tau^m}{m!}, \tag{6}$$

$$u = -L_\tau + \sum_{m=1}^{\infty}\frac{1}{m!}L_\tau^m\eta^m\sum_{k=0}^{m-1}\left\{\begin{matrix} k+m-1 \\ k\end{matrix}\right\}_{\geq 2}\frac{(-1)^{k+m-1}}{(1+\eta)^{k+m}} \ . \tag{7}$$

All of these expansions are limited in their domain of applicability by the fact that σ and τ are each singular at $z = 1$, restricting their utility to $z > 1$. In addition to the domain of validity of the variables, there is the question of the domain of convergence of the series. For example, we show below that for $z \in \mathbb{R}$, series (4) is convergent only for $z > e$.

In this paper, we consider transformations of the above series. We shall concentrate on the properties of the series for $z \in \mathbb{R}$. Our aims are to improve the convergence properties with respect to domain of convergence and with respect to rate of convergence. We shall do this using theoretical and experimental methods.

2 Computer Algebra Tools

We shall be using a number of tools from MAPLE in the work below. The coefficients appearing in the expansions (4) and (5) can be computed from their generating functions as follows. The 2-associated Stirling subset numbers are defined by the generating function

$$(e^z - 1 - z)^m = m! \sum_{n\geq 0} \frac{z^n}{n!} \left\{ {n \atop m} \right\}_{\geq 2} .$$

Given numerical values for n and m, we expand the left-hand side symbolically up to the term of nth order and then extract the appropriate numerical coefficient. The next lines show an implementation of this procedure with examples in Maple.

```
> StirlingSubset2:=proc(n::integer, m::integer)
                     option remember;
                     local f,z;
                     f:=series( (exp(z)-1-z)^m , z , n+1);
                     if n<2*m then
                          0
                       else
                          coeff(f,z,n)*n!/m!;
                       end if;
                     end proc;

> StirlingSubset2(6,3),StirlingSubset2(9,4),StirlingSubset2(12,5);
                  15,  1260,  190575
```

It can be noted that a similar method to this is used in the standard MAPLE library for Stirling Cycle numbers, which are used in (4). In practice, it is more efficient to store all of the coefficients from any series expansion, but this level of detail is not shown here. Similar techniques can be used for the Eulerian numbers used below in (23).

Another important tool from MAPLE for this paper is computation to arbitrary precision. It is a standard topic in numerical analysis that summing series requires a close watch on the effects of working precision, otherwise one runs the risk of generating 'numerical monsters' which are completely artificial effects of the computation and do not reflect any actual mathematical properties [9]. In all of the calculations below, the MAPLE environment variable `Digits` was set and monitored to ensure that the results were reliable.

3 An Invariant Transformation

We reconsider the derivation of (4), replacing (3) with the *ansatz*

$$W = \ln z - \ln(p + \ln z) + u \ . \tag{8}$$

Substituting into the defining equation $We^W = z$, we obtain

$$\left(\ln z - \ln(p + \ln z) + u\right)\frac{ze^u}{p + \ln z} = z$$

From this, it is clear that if we define

$$\sigma = \frac{1}{p + \ln z} \text{ and } \tau = \frac{p + \ln(p + \ln z)}{p + \ln z} \ , \tag{9}$$

then we recover the equation originally given by de Bruijn for u.

$$1 - \tau + \sigma u - e^{-u} = 0 \ . \tag{10}$$

The remarkable property is that (10) is invariant with respect to p, with only the definitions of σ and τ being changed. From (10), the expansion (4) is derived [5].

We now consider the properties of the transformations for $z \in \mathbb{R}$. We shall start with $p \in \mathbb{R}$ and later consider briefly one complex value of p. Both σ and τ are singular at $z_s = e^{-p}$, with the special case $p = 0$ recovering the previous observations regarding the singularities at $z = 1$. We note σ is monotonically decreasing on $z > z_s$. For τ, we have $\tau(z_0) = 0$ at $z_0 = \exp(z_s - p)$, with τ positive for larger z and negative for smaller. Also we note that τ has a maximum at $z = \exp(ez_s - p)$. In Figure 1, we plot σ and τ, defined by (9), for different values of p. We see that for all $z > z_s$, σ decreases with increasing p, but τ increases. In view of the form of the double sums above it is not obvious whether convergence is increased or decreased as a result of these opposed changes. This is what we wish to investigate here.

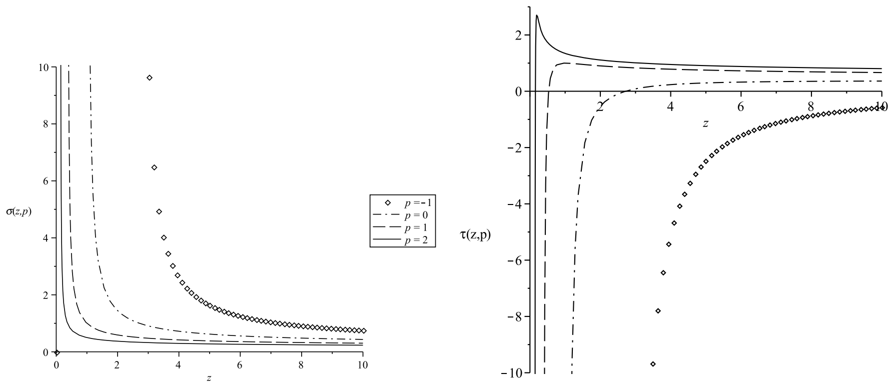


Fig. 1. Dependence σ and τ on z for different values of parameter p

4 Domain of Convergence

We wish to investigate first the domains of $z \in \mathbb{R}$ for which the various series above converge, and how the domains vary with p. We begin with a theoretical result for $p = 0$.

Theorem 1. *The series (4) converges for $p = 0$ for all $z \geq e$.*

Proof. For $p = 0$, we have $\tau = -\sigma \ln \sigma$. We write (10) in the form

$$1 - \tau + \sigma u - e^{-u} = g(u) + f(u; \sigma, \tau) = 0 \ , \qquad (11)$$
$$g(u) = 1 - e^{-u} \quad \text{and} \quad f(u; \sigma, \tau) = \sigma u - \tau \ .$$

We now consider this equation in the complex plane of u. For any analytic function $F(\zeta)$ with a single isolated zero at $\zeta = u$ inside a contour C, we can use Cauchy's integral formula to write

$$u = \frac{1}{2\pi i} \int_C \frac{F'(\zeta)}{F(\zeta)} \, \zeta \, \mathrm{d}\zeta \ . \qquad (12)$$

Thus for our case, we have

$$u = \frac{1}{2\pi i} \int_C \frac{e^{-\zeta} + \sigma}{1 - e^{-\zeta} + \sigma\zeta - \tau} \zeta \, \mathrm{d}\zeta = \frac{1}{2\pi i} \int_C \frac{e^{-\zeta} + \sigma}{g(\zeta) + f(\zeta; \sigma, \tau)} \, \zeta \, \mathrm{d}\zeta \ , \qquad (13)$$

provided we can find the contour C.

For $z \approx e$ while $z > e$, we have $\sigma \approx 1$ and $\sigma < 1$. We define $\delta > 0$ by $\sigma = (1 - \delta)$. A contour which satisfies the requirements is the rectangular contour

$$\zeta = \begin{cases} \delta + it \ , & -2\delta^{1/2} \leq t \leq 2\delta^{1/2} \ , \\ t + 2\delta^{1/2} i \ , & -2 \leq t \leq \delta \ , \\ -2 + it \ , & -2\delta^{1/2} \leq t \leq 2\delta^{1/2} \ , \\ t - 2\delta^{1/2} i \ , & -2 \leq t \leq \delta \ . \end{cases} \qquad (14)$$

It is straightforward for MAPLE show that on this contour $|g| > |f|$. Rouché's theorem states that g and $f + g$ have the same number of zeros within C. Since $g(u) = 0$ for $u = 0$, the function $f + g$ has a single isolated zero as desired.

In addition to satisfying the conditions of the integration, the contour allows us to evaluate the integral by expanding the denominator of the integrand as an absolutely and uniformly convergent power series in f/g.

$$\frac{1}{1 - e^{-\zeta} + \sigma\zeta - \tau} = \sum_{k=0}^{\infty} \sum_{m=0}^{\infty} (1 - e^{-\zeta})^{-k-m-1} \zeta^k \sigma^k \tau^m (-1)^{k+1} C_m^{m+k} \ . \qquad (15)$$

Substituting this expansion into (13) and integrating term by term, we obtain u as the sum of an absolutely convergent double power series in σ and τ, provided $z > e$.

The domain of convergence cannot be extended to $z < e$, because the series for du/dz diverges at $z = e$. This can be seen by noting that $\tau = 0$ at $z = e$ (for $p = 0$). All terms reduce to zero except $m = 1$ which gives the sum

$$\frac{1}{e}\sum_{k=0}^{\infty}(-1)^k \ ,$$

which is divergent. □

In general, the precise domain of convergence is not of high importance, although its characterization remains an interesting mathematical challenge. The important point is to establish whether the domain of convergence increases or decreases, so that numerical procedures can be designed accordingly. Therefore, rather than devote space here to accumulating formal proofs for all the different cases, we can use numerical means as a rapid method to ascertain trends in the domains of convergence for all series. The method is simply to compute the partial sum of a series to a high number of terms, using extended floating-point precision as necessary, and then to plot the ratio of the partial sum to the exact value (the exact value is obtained by means other than series summation). The edge of the domain of convergence is then signaled by rapid oscillations and by marked deviations from the desired ratio of 1. Thus for the series just discussed, namely (4), we have plotted in Figure 2 the partial sum to 40 terms for different values of p. For $p = 0$, we see a nice illustration of theorem 1, with the partial sum becoming unstable in the vicinity of $z = e$. For positive p, we see the domain of convergence increased and for negative p it is decreased.

Similar effects can be seen for (5), we plot in Figure 3 the partial sums for 40 terms as p varies. The domain of convergence for each p is clearly seen, and confirms that the point of divergence moves to larger z for decreasing p and to the left for increasing p.

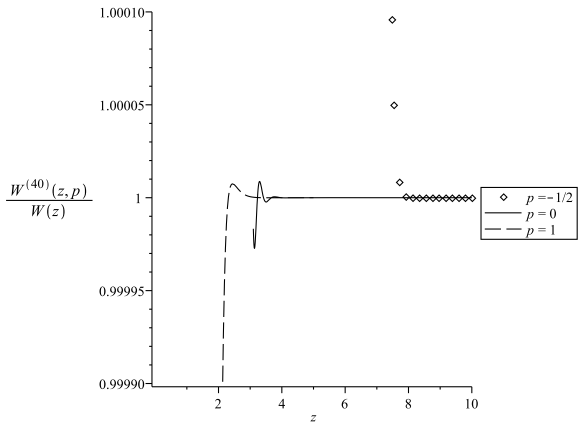


Fig. 2. For series (4), the ratio $W^{(40)}(z,p)/W(z)$ as functions of z for $p = -1/2, 0, 1$

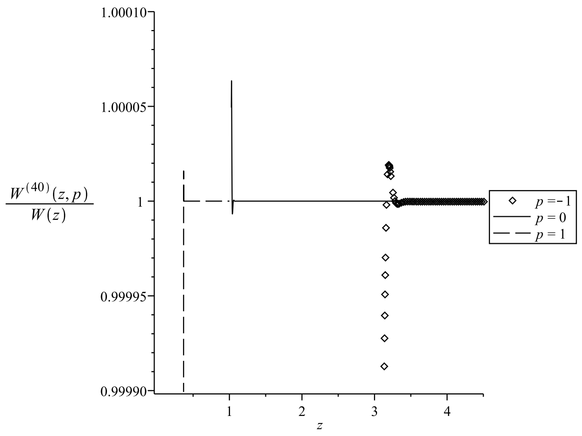


Fig. 3. For series (5), the ratio $W^{(40)}(z,p)/W(z)$ as functions of z for $p = -1, 0, 1$. Compared with Figure 2, this shows convergence down to smaller z.

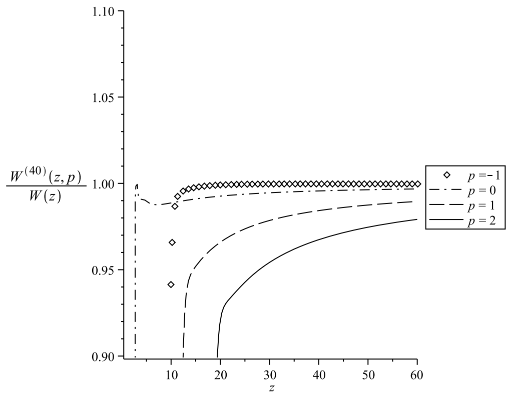


Fig. 4. For series (6), the ratio $W^{(40)}(z,p)/W(z)$ against z for $p = -1, 0, 1, 2$. Compared with figures 2 and 3, the changes in convergence are no longer monotonic in p.

A similar investigation of series (6) shows an interesting non-monotonic change in the domain of convergence. In Figure 4 the partial sums are plotted and the boundary of the domain of convergence moves to the right for $p \neq 0$.

We can summarize these findings by noting that series (5) has the widest domain of convergence, and the best behaviour with p, while the domains of convergence for series (4) and (6) become worse in that order.

5 Rate of Convergence

By rate of convergence, we are referring to the accuracy obtained by partial sums of a series. Given two series, each summed to N terms, the series giving on

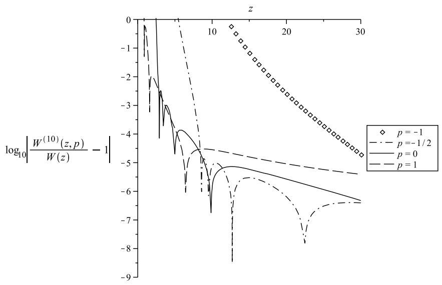


Fig. 5. For series (4) with $N = 10$, changes in accuracy in z for $p = -1, -1/2, 0$ and 1

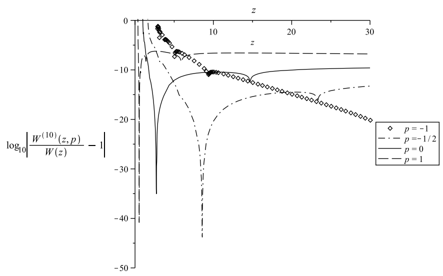


Fig. 6. For series (5) with $N = 10$, changes in accuracy in z for $p = -1, -1/2, 0$ and 1

average a closer approximation to the converged value is said to converge more quickly. The qualification 'on average' is needed because it will be seen in the plots below that the error regarded as a function of z can show fine structure which confuses the search for a general trend. Further, the comparison of rate of convergence between different series can vary with z and p. For some ranges of z, one series will be best, while for other ranges of z a different series will be best. Although one series may converge on a wider domain than another, there is no guarantee that the same series will converge more quickly on the part of the domain they have in common. The practical application of these series is to obtain rapid estimates for W using a small number of terms, and for this the quickest convergence is best, but this will be dependent on the domain of z.

The previous section showed that positive values of the parameter p extend the domain of convergence of the series, but its effect on rate of convergence is different. Figures 5, 6 and 7 show the dependence on z of the accuracy of computations of the series (4),(5) and (7) respectively with $N = 10$ for $p = -1, -1/2, 0$ and 1. One can see that the behaviour of the accuracy is non-monotone with respect to both z and p although some particular conclusions can be made. For example, one can observe that for the series (4) at least for $z < 30$ within the common domain of convergence the accuracy for $p = -1/2, 0$ and 1 is higher than for $p = -1$. The series (5) and (7) have the same domain of convergence and a very similar behaviour of the accuracy. Specifically, for these series an increase of positive values of p reduces a rate of convergence within the common

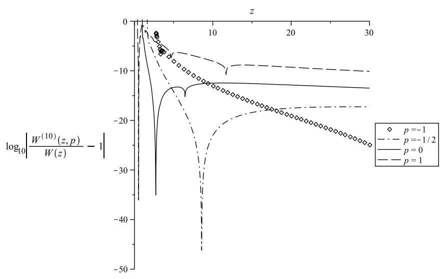


Fig. 7. For series (7) with $N = 10$, changes in accuracy in z for $p = -1, -1/2, 0$ and 1

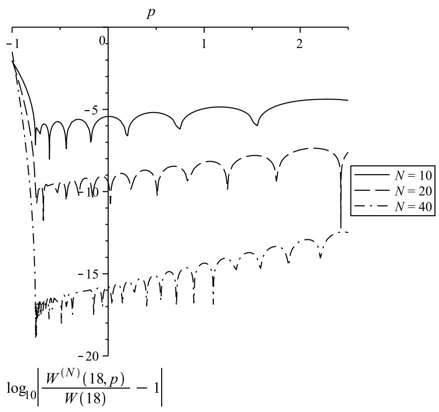


Fig. 8. For series (4), the accuracy as a function of p at fixed point $z = 18$ for $N = 10, 20$ and 40

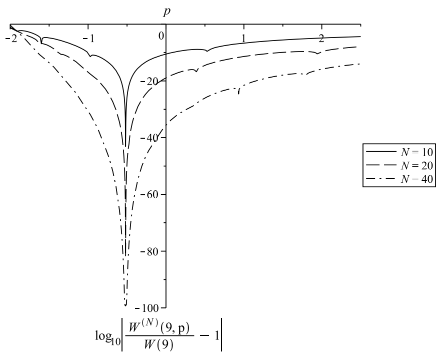


Fig. 9. For series (5), the accuracy as a function of p at fixed point $z = 9$ for $N = 10, 20$ and 40

domain of convergence i.e. for $z > 1.5$. However, at the same time for $z > 11$ computations with $p = -1$ are more accurate than those with positive p and for $5 < z < 18$ the highest accuracy occurs when $p = -1/2$.

The next two figures 8 and 9 display the dependence of convergence properties of the series (4) and (5) respectively on parameter p for different numbers of terms $N = 10, 20$ and 40. Again, the curves in these figures confirm that the accuracy strongly depends on parameter p and is non-monotone and show that on the whole an increase of the number of terms improves the accuracy. It is also interesting that there exists a value of p for which the accuracy at the given point is maximum; this value depends very slightly on N and approximately is $p \approx -0.75$ in Figure 8 and $p \approx -0.5$ in Figure 9.

The explained behaviour of the accuracy depending on parameter p shows that introducing parameter p in the series can result in significant changes in accuracy. The pointed out non-monotone effects of parameter p on a rate of convergence can be due to the aforementioned non-monotone behaviour of τ.

6 Branch −1 and Complex p

The above discussion has considered only real values for the parameter p. We briefly shift our consideration to complex p and to branch -1. For z in the domain $-1/e < z < 0$, we have that $W_{-1}(z)$ takes real values in the range $[-1, -\infty)$. The general asymptotic expansion (2) takes the form

$$W_{-1}(z) = \ln(z) - 2\pi i - \ln(\ln(z) - 2\pi i) + u \ . \tag{16}$$

This will clearly be very inefficient for $z \in [-1/e, 0)$ because each term in the series will be complex, and yet the series must sum to a real number. If, however, we utilize the parameter p, we can improve convergence enormously.

We again adopt the *ansatz* used above to write

$$W_k(z) = [\ln_k z + p] - [p + \ln(p + \ln_k z)] + \frac{p + \ln(p + \ln_k z)}{p + \ln_k z} + v \ , \tag{17}$$

where v stands for the remaining series which will not be pursued here. By setting $p = i\pi$, we can rewrite $[\ln_{-1} z + i\pi]$ as $\ln(-z)$. A numerical comparison of partial sums can be used to show the improvement. We compare

$$W_{-1}^{(1)} = \ln(z) - 2\pi i - \ln(\ln(z) - 2\pi i) + \frac{\ln(\ln(z) - 2\pi i)}{\ln(z) - 2\pi i} \ , \tag{18}$$

$$\hat{W}_{-1} = \ln(-z) - \ln(-\ln(-z)) + \frac{\ln(-\ln(-z))}{\ln(-z)} \ . \tag{19}$$

The results are shown in table 1. We note that the transformed series is exactly correct at $z = -1/e$ and asymptotically correct as $z \to 0$, and therefore the error is a maximum somewhere in the domain. In contrast the untransformed series has an error that increases as $z \to -1/e$.

Table 1. Numerical comparison of series transformation with $p = i\pi$

z	$W_{-1}(z)$	$\hat{W}_{-1}(z)$	$W_{-1}^{(1)}(z)$
-0.01	-6.4728	-6.4640	$-6.3210 - 0.04815i$
-0.1	-3.5772	-3.4988	$-3.4124 - 0.3223i$
-0.2	-2.5426	-2.3810	$-2.5182 - 0.5153i$
-0.3	-1.7813	-1.5438	$-2.0087 - 0.6621i$
$-1/e$	-1	-1	$-1.7597 - 0.7450i$

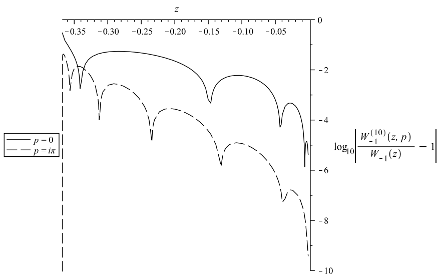


Fig. 10. Errors in approximations (18) and (19) for W_{-1}

The accuracy is also shown graphically in figure 10. Notice that although the approximation $\hat{W}_{-1}$ given in (19) is exactly equal to W_{-1} at $z = -e^{-1}$, the local behaviour is different. We know that W_{-1} has a square-root singularity, while $\hat{W}_{-1}$ is regular there. This is why the maximum error occurs at $z = -e^{-1}$.

7 Taylor Series

We have seen that the transformation allows us to obtain series valid for a wider range of z. We now observe that the Taylor series for $W(z)$ around $z = 0$ is well known [1]

$$W(z) = \sum_{n=1}^{\infty} \frac{(-n)^{n-1}}{n!} z^n. \tag{20}$$

This converges for $|z| < e^{-1}$. We can bridge the gap between the series above and the Taylor series by a series around $z = 1$. We have [7,10] with $\omega = W(1)$

$$W(x) = \omega + \sum_{n=1}^{\infty} a_n (\ln x)^n \tag{21}$$

or by setting $t = \ln x$

$$W(e^t) = \omega + \sum_{n=1}^{\infty} a_n t^n, \tag{22}$$

where

$$a_n = \frac{1}{n!(1+\omega)^{2n-1}} \sum_{k=0}^{n-1} \left\langle\!\!\left\langle {n-1 \atop k} \right\rangle\!\!\right\rangle (-1)^k \omega^{k+1} \ . \tag{23}$$

This formula represents the expansion coefficients in terms of the second-order Eulerian numbers [6,7]. We now show that these coefficients can also be represented through the unsigned associated Stirling numbers of the first kind $d(m,k)$ given by [5]

$$[\ln(1+v) - v]^k = k! \sum_{m=2k}^{\infty} (-1)^{m+k} \, d(m,k) \frac{v^m}{m!} \tag{24}$$

and the 2-associated Stirling subset numbers used in the series (5).

Both representations can be obtained on the basis of a relation [2]

$$W(e^t) + \ln W(e^t) = t \tag{25}$$

and the Lagrange Inversion Theorem [11]. To apply this theorem it is convenient to introduce a function that is zero at $t = 0$. We consider the function

$$v = v(t) = W(e^t)/\omega - 1 \tag{26}$$

and write (25) as

$$t = \omega \, v + \ln(1+v). \tag{27}$$

Then by the Lagrange Inversion Theorem we obtain

$$v = \sum_{n=1}^{\infty} \frac{t^n}{n} [v^{n-1}] \left(\omega + \frac{\ln(1+v)}{v} \right)^{-n} \tag{28}$$

where the operator $[v^p]$ represents the coefficient of v^p in a series expansion in v. Comparing (26),(22) and (28) leads to a formula for the coefficients a_n, which after applying the binomial theorem becomes

$$a_n = \frac{\omega}{n(1+\omega)^n} [v^{n-1}] \sum_{k=0}^{\infty} (-1)^k \binom{n-1+k}{n-1} \frac{[\ln(1+v) - v]^k}{v^k (1+\omega)^k} \tag{29}$$

or by (24)

$$a_n = \frac{\omega}{n!} \sum_{k=0}^{n-1} \frac{(-1)^{n+k-1} \, d(n+k-1,k)}{(1+\omega)^{n+k}}. \tag{30}$$

If instead of function (26), we take

$$h = h(t) = W(e^t) - \omega - t \tag{31}$$

and apply the Lagrange Inversion Theorem to invert a relation

$$t = (e^{-h} - 1)\, \omega - h \tag{32}$$

coming from (25), then we find in a similar way

$$a_n = \frac{1}{n!}\sum_{k=0}^{n-1} \left\{ {n+k-1 \atop k} \right\}_{\geq 2} \frac{(-1)^{k+1}\omega^k}{(1+\omega)^{n+k}}. \tag{33}$$

Finally, one more representation for the coefficients a_n can be found in the following way. Let us consider a function

$$\psi = \psi(t) = W(e^t) - t \tag{34}$$

which is a simplified version of functions (26) and (31): now one does not need to provide the zero function value at $t = 0$ and here $\psi(0) = \omega$. Then it follows from (25) that

$$t = e^{-\psi} - \psi. \tag{35}$$

This equation can also be obtained from the fundamental relation (10) by transformation $u = \psi + \ln t, \sigma = 1/t$ and $\tau = \ln t/t$.

Differentiating (35) in t and excluding the term $e^{-\psi}$ from the result again using (35) result in an initial value problem for ordinary differential equation

$$\frac{d\psi}{dt} = -\frac{1}{1+t+\psi}. \tag{36}$$

Searching a solution in the form of series

$$\psi(t) = \omega + \sum_{n=1}^{\infty} c_n t^n \tag{37}$$

by substituting it into the differential equation and equating coefficients of the same power in t one can finally find

$$c_1 = -\frac{1}{1+\omega}, \quad c_n = -\frac{1}{n(1+\omega)}\left((n-1)c_{n-1} + \sum_{k=1}^{n-1} k c_k c_{n-k}\right), \text{ for } n \geq 2\ . \tag{38}$$

At length combining (37),(34) and (22) gives

$$a_1 = 1 + c_1, \quad a_n = c_n \text{ for } n \geq 2. \tag{39}$$

In practice, computing the coefficients using (38) and (39) is found to be more effective than using other representations. However, we have found some remarkable combinatorial identities. For example, equating the right-hand sides of (23) and (33) we obtain

$$\frac{1}{(1+\omega)^{n-1}}\sum_{k=0}^{n-1} \left\langle\!\!\left\langle {n-1 \atop k} \right\rangle\!\!\right\rangle (-1)^{k+1}\omega^{k+1} = \sum_{k=0}^{n-1} \left\{ {n+k-1 \atop k} \right\}_{\geq 2} \frac{(-1)^k\omega^k}{(1+\omega)^k}.$$

8 Concluding Remarks

We found an invariant transformation defined by the parameter p and applied it to the series for the Lambert W function to obtain a family of series. We studied an effect of parameter p on convergence properties of the transformed series. It is shown that an increase of p results in an extension of the domain of convergence of the series and thus the series obtained under the transformation with positive values of p have a wider domain of convergence than the original series does. However, at the same time a rate of convergence can be found to be reduced when the parameter p increases. Therefore in such a case within the common domain of convergence of the series with different positive values of p the series with the minimum value of p would be the most effective. The found relationships can be used, e.g. in evaluating of the Lambert W function in computer algebra systems.

We also considered the well-known expansion of $W(x)$ in powers of $\ln x$ and found three more forms for a represenation of the expansion coefficients in terms of the associated Stirling numbers of the first kind (30), the 2-associated Stirling subset numbers (33) and iterative formulas (39)-(38). As a consequence some combinatorial identities are obtained.

References

1. Corless, R.M., Gonnet, G.H., Hare, D.E.G., Jeffrey, D.J., Knuth, D.E.: On the Lambert W Function. Advances in Computational Mathematics 5, 329–359 (1996)
2. Jeffrey, D.J., Hare, D.E.G., Corless, R.M.: Unwinding the branches of the Lambert W function. Mathematical Scientist 21, 1–7 (1996)
3. Heffernan, J.M., Corless, R.M.: Solving some delay differential equations with computer algebra. Mathematical Scientist 31(1), 21–34 (2006)
4. de Bruijn, N.G.: Asymptotic Methods in Analysis. North-Holland, Amsterdam (1961)
5. Comtet, L.: C. R. Acad. Sc., Paris 270, 1085–1088 (1970)
6. Graham, R.L., Knuth, D.E., Patashnik, O.: Concrete Mathematics. Addison-Wesley, Reading (1994)
7. Corless, R.M., Jeffrey, D.J., Knuth, D.E.: A Sequence of Series for The Lambert W Function. In: Proceedings of the ACM ISSAC, Maui, pp. 195–203 (1997)
8. Jeffrey, D.J., Corless, R.M., Hare, D.E.G., Knuth, D.E.: On the inversion of $y^\alpha e^y$ in terms of associated Stirling numbers. C. R. Acad. Sc., Paris 320, 1449–1452 (1995)
9. Essex, C., Davison, M., Schulzky, C.: Numerical monsters. SIGSAM Bulletin 34(4), 16–32 (2000)
10. Corless, R.M., Jeffrey, D.J.: The wright ω function. In: Calmet, J., Benhamou, B., Caprotti, O., Hénocque, L., Sorge, V., et al. (eds.) AISC 2002 and Calculemus 2002. LNCS (LNAI), vol. 2385, pp. 76–89. Springer, Heidelberg (2002)
11. Carathéodory, C.: Theory of Functions of a Complex Variable, Chelsea (1954)

Differential Resultant, Computer Algebra and Completely Integrable Dynamical Systems

Zoia Kostova[1], Nikolay Kostov[2], and Vladimir Gerdjikov[2]

[1] Meridian 22 Private School, Block 227, Mladost 2, 1799 Sofia
zoia.1959@abv.bg
[2] Institute of Nuclear Research and Nuclear Energy, Bulgarian Academy of Sciences, Blvd. Tsarigradsko shousse 72, Sofia 1784, Bulgaria
nakostov@inrne.bas.bg, gerjikov@inrne.bas.bg

Abstract. For a pair of differential operators A and B with periodic coefficients we construct their differential resultant and derive condition for their commutativity. By considering this condition as a stationary Lax representation we are able to treat completely integrable dynamical systems. As special cases we obtain Hénon-Heiles dynamical systems. We propose algorithms to do this by using the powerful methods of computer algebra and performing symbolic calculations in Maple13 and Reduce4.

Keywords: computer algebra, Lax representation, Baker-Akhiezer function, differential resultant, algebra of commuting differential operators.

1 Introduction

The method of the differential resultant (see the monograph [2] and the references therein) is an important tool for analyzing the algebras of differential operators. It has been extensively used for developing algorithms for analytic calculations for REDUCE, MAPLE and other high level packages. The present paper is an extension of [3] and [23] and is organized as follows: section 2 describes the basic facts about Lax representation, Baker-Akhiezer (BA) function $\Psi(x,t,\lambda)$ theory for finite–gap integration method. Next we study differential resultant of ordinary differential operators, section 3. Section 4 is dedicated to extracting left and right prime divisors of a given LODE and to the related Riccati type equations. In section 5 we analyze classes of LODE allowing exponential and polynomial solutions, as well as equations taking the form of exact differentials. We close our paper with two examples analyzed earlier by Fordy [8]. He relates the stationary KdV eq. of degree 5 and the stationary Sawada-Kotera equation to the Hénon-Heiles completely integrable systems interpreted as commutativity condition for two ordinary differential operators.

2 Preliminaries

We start with some basic facts about Baker–Akhiezer function $\Psi(x,t,\lambda)$ which will be used in the sequel. For systematic treatments and proofs, we refer the reader to [25,4,21,6].

Consider the following linear differential equations

$$A\Psi(x,t,\lambda) = z\Psi(x,t,\lambda), \tag{1}$$

$$B\Psi(x,t,\lambda) = \lambda\Psi(x,t,\lambda). \tag{2}$$

where the differential operators A and B are defined by

$$A = \sum_{k=0}^{n} a_k(x,t)D^k, \tag{3}$$

$$B = \sum_{l=0}^{m} b_l(x,t)D^l, \qquad D = \frac{d}{dx}. \tag{4}$$

λ is the spectral parameter and $z = z(\lambda)$. We consider the case when the coefficient functions $a_k(x,t)$ and $b_l(x,t)$ are periodic functions of x.

We assume that A and B satisfy the Lax representation [1] of the form:

$$A_t = [B, A]. \tag{5}$$

Eq. (5) means that the system (3) and

$$\left(\frac{d}{dt} - B\right)\Psi(x,t,\lambda) = \lambda\Psi(x,t,\lambda). \tag{6}$$

have a common set of fundamental solutions. Of course this is possible only if the spectral parameters z and λ are functionally related. Using Lax representations with different choices of the operators A and B has resulted in discovering an immense number of completely integrable systems, see [6,30,1,14].

The common fundamental solution $\Psi(x,t,\lambda)$ for the class of periodic potentials is known as the Baker-Akhiezer (BA) function. The BA functions have been constructed for a wide class of operators A and B in terms of elliptic theta-functions. They have become an important tool in the theory of finite-gap integration method for constructing explicit periodic and quasi-periodic solutions of the nonlinear evolution equations possessing Lax representations (5).

The BA function $\Psi(x,t,\lambda)$, which is general solution of the system (5), is meromorphic function on the Riemann surface and has essential singularities of prescribed form near fixed points on Riemann surface which is defined by

$$\det(\mathrm{RRres}(A(x,t)B(x,t)) = 0. \tag{7}$$

where $\mathrm{RRes}A(x,t)B(x,t))$ is the differential resultant of operators A and B defined below in Section 3.

Another important topic is to study the stationary equations

$$[B, A] = 0, \tag{8}$$

which allows one to analyze finite dimensional dynamical systems.

In what follows we will outline on several examples how one can construct these Riemann surfaces (or algebraic curves) using the 'algcurve' package of Maple13, see [5].

3 Differential Resultant of Two Differential Operators

Following [2,9] let us consider the differential operators A (3) and B (4) with coefficients $a_k \in C_I^k$ and $b_l \in C_I^l$ where C_I^k (resp. C_I^l) is the set of k-th (resp. l-th) differentiable functions on the interval I of the real variable x.

We start by defining right resultant $\mathrm{RRes}\,(A, B)$ and left resultant $\mathrm{LRes}\,(A, B)$ as follows: we act by the operators $D^{m-1}, \dots, D$, $D^0 = 1$ on the left on A and by the operators $D^{n-1}, \dots, D$, $D^0 = 1$ on the left on B. Thus we obtain the systems:

$$\sum_{k=0}^{n+s} a_{k,s} y^{(k)} = 0, \qquad s = 0 \div (m-1), \tag{9}$$

$$\sum_{l=0}^{m+p} b_{l,p} y^{(l)} = 0, \quad p = 0 \div (n-1), \tag{10}$$

where the coefficients $a_{k,s}$ and $b_{l,p}$ are computed by:

$$a_{k,s} = \sum_{i=0}^{s} \binom{s}{i} a_{k-i}^{(s-i)}, \qquad b_{l,p} = \sum_{j=0}^{p} \binom{p}{j} b_{l-j}^{(p-j)}. \tag{11}$$

The right resultant of operators A and B $\mathrm{RRes}\,(A, B)$ is called the determinant of the following resultant matrix R of degree $(m+n)$:

$$\mathrm{RRes}\,(A,B) = \begin{vmatrix} a_{n+m-1,m-1} & a_{n+m-2,m-1} & \cdots & \cdots & \cdots & a_{0,m-1} \\ 0 & a_{n+m-1,m-2} & \cdots & \cdots & \cdots & a_{0,m-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & a_{n,0} & \cdots & a_{0,0} \\ b_{n+m-1,n-1} & b_{n+m-2,n-1} & \cdots & \cdots & \cdots & b_{0,n-1} \\ 0 & b_{n+m-2,n-2} & \cdots & \cdots & \cdots & b_{0,n-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & b_{m,0} & \cdots & b_{0,0} \end{vmatrix}. \tag{12}$$

In order to introduce the left resultant we will need the the conjugated operators A^* and B^* which are defined by:

$$A^* = \sum_{r=0}^{n} \sum_{k=r}^{n} (-1)^k \binom{k}{r} a_k^{(k-r)} D^k, \tag{13}$$

$$B^* = \sum_{r=0}^{n} \sum_{k=r}^{n} (-1)^k \binom{k}{r} b_k^{(k-r)} D^k. \tag{14}$$

Then the left resultant of operators A and B $\mathrm{LRes}\,(A, B)$ is called the resultant matrix R^* of degree $(m+n)$, i.e. $\mathrm{RRes}\,(A, B) = \det(R)$, $\mathrm{LRes}\,(A, B) = \det(R^*)$. By definition $\mathrm{LRes}\,(A, B) = \mathrm{RRes}\,(A^*, B^*)$, where A^* and B^* are the conjugated operators. The differential resultant answers to the question when the operators

A and B has right(left) divisor i.e. when the overdetermined system $Ay = 0$, $By = 0$ (or $A^*y = 0$, $B^*y = 0$) has solution.

We will say that the systems $Ay = 0$, $By = 0$ are consistent if $\det(R) = \mathrm{RRes}\,(A, B) = 0$; otherwise the systems are not consistent.

For applications of differential resultant see for example [2]. The most important ones are the following: i) the criterion of existence of the greatest right and left divisor of the system of operators; ii) the criterion of consistency of the system of linear ordinary differential equations with one unknown function; iii) the criterion of commutation of two linear differential operators; iv) the criterion of existence of polynomial and exponential solutions of linear differential equations with variable coefficients; v) the criterion of factorization of operators of degree n into products of operators of degrees $n-1$ and 1.

Example 1. Let us introduce the following equations

$$L_1 \equiv Ay = x^2y'' + xy' - (x^2 + 1/4)y = 0, \tag{15}$$
$$L_2 \equiv By = 2xy'' + (3 - 4x)y' + (2x - 3)y = 0. \tag{16}$$

Using (12) we obtain

$$\begin{aligned}\mathrm{RRes}\,(A, B) &= \begin{vmatrix} x^2 & 3x & -x^2 + 3/4 & -2x \\ 0 & x^2 & x & -x^2 - 1/4 \\ 2x & 5x - 4 & 2x - 7 & 2 \\ 0 & 2x & 3 - 4x & 2x - 3 \end{vmatrix} \\ &= \frac{9}{4}x^2(x - 1)(4x - 1)(4x^2 - 4x + 3).\end{aligned} \tag{17}$$

Problem 1. Find the differential resultant of operators A and B introduced above in eqs. (3) and (4).

Algorithm 1. We solve problem 1 by the procedure DIFRESULT(a, n, b, m, x); Next we outline the algorithm 1. Using (11) with given coefficients of the differential operators A and B we compute the elements of the resultant matrix, after that we find the determinant. The output is $\mathrm{RRes}\,(A, B) = \det(R)$, where R is the resultant matrix.

Input

a is the array of coefficients of differential operator A;

n is the degree of differential operator A;

b is the array of coefficients of differential operator B;

m is the degree of differential operator B;

x is independent variable.

Output: The resultant of two differential operators A and B, (17).

4 The Generalized Riccati Equation

4.1 Prime Right Divisor of Operator *L*

Let us consider the differential operator A of degree n ([3]). A necessary and sufficient conditions for the following factorization:

$$A = A_2(D-\alpha), \qquad \operatorname{ord} A_2 = n-1$$

is given by $\operatorname{RRes}(A, D-\alpha) = 0$.

$$\operatorname{RRes}(A, D-\alpha) = \begin{vmatrix} a_n & a_{n-1} & \dots & a_2 & a_1 & a_0 \\ 1 & -\alpha & \dots & -\binom{n-1}{n-3}\alpha^{(n-3)} & -\binom{n-1}{n-2}\alpha^{(n-2)} & -\alpha^{(n-1)} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & -\alpha & -\alpha' \\ 0 & 0 & \dots & 0 & 1 & -\alpha \end{vmatrix}. \quad (18)$$

Then we find the following generalized Riccati equation of first kind $N(\alpha)$ of degree $(n-1)$:

$$N(\alpha) = L(\alpha) + M(\alpha),$$

where

$$L(\alpha) = \sum_{k=0}^{n} a_k \alpha^k,$$

and $L(\alpha)$ is called the pseudo-characteristic Riccati equation of first kind

$$M(\alpha) = \sum_{k=1}^{n} a_k M_{k-1}, \qquad M_{k-1} = (D+\alpha)^{k-1} - \alpha^{(k-1)}$$

and $M(\alpha) = 0$ is called the reduced Riccati equation of first kind. When $n = 2$ the generalized Riccati equation of first kind has the form

$$N(\alpha) \equiv a_2\alpha' + a_2\alpha^2 + a_1\alpha + a_0 = 0.$$

$N(\alpha) = 0$ **-generalized Riccati** $M(\alpha) = 0$ **-reduced Riccati**

4.2 Prime Left Divisor of Operator *L*

Consider again the differential operator A of degree n ([3]). The necessary and sufficient conditions for the factorization of the form:

$$L = (D-\alpha)L_1, \qquad \operatorname{ord} L_1 = n-1$$

is $\operatorname{LRes}(L, D-\alpha) = \operatorname{RRes}(L^*, D+\alpha)$, i.e.

$$\begin{vmatrix} a_n^* & a_{n-1}^* & \dots & a_2^* & a_1^* & a_0^* & \\ 1 & \alpha & \dots & \binom{n-1}{n-3}\alpha^{(n-3)} & \binom{n-1}{n-2}\alpha^{(n-2)} & \alpha^{(n-1)} & \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & \alpha & \alpha' & \\ 0 & 0 & \dots & 0 & 1 & \alpha & \end{vmatrix} = 0.$$

Thus we obtain the generalized Riccati equation $N^*(\alpha)$ of second kind and degree $(n-1)$:

$$N^*(\alpha) \equiv L^*(\alpha) + M^*(\alpha) = 0,$$

where

$$L^*(\alpha) \equiv L = \sum_{r=0}^{n} \sum_{k=r}^{n} (-1)^k \binom{k}{r} a_k^{(k-r)} \alpha^k = L_1^*(D+\alpha). \tag{19}$$

Then $L^*(\alpha) = 0$ is the pseudo-characteristic Riccati equation of second kind and $M^*(\alpha) = 0$ is the reduced Riccati equation of second kind, where

$$M^*(\alpha) = \sum_{r=1}^{n} \sum_{k=r}^{n} (-1)^k \binom{k}{r} a_k^{(k-r)} M_{r-1}, \qquad M_r = (D+\alpha)^r - \alpha^{(r)}, \tag{20}$$

When $n = 2$ the generalized Riccati equation of second kind has the form,

$$N^*(\alpha) \equiv a_2\alpha' + a_2\alpha^2 + (-a_1 + 2a_2')\alpha + a_0 - a_1' + a_2'' = 0$$

As a consequence of problem 1 we have:

Problem 2. Let the operators L_1 and L_1^* are defined by the right hand sides of eqs. (3) and (13). Find the generalized Riccati equation of first kind,

$$N_1(\alpha) = \mathrm{RRes}\,(L_1, D-\alpha) = 0$$

and the generalized Riccati equation of second kind,

$$N_2^*(\alpha) = \mathrm{RRes}\,(L_1^*, D+\alpha)) = 0.$$

Algorithm 2. We solve the problem 2 by the procedure RICCATI(a, n, pp, x).

```
Input
$a$ is the array of coefficients of differential operator $L_1$;
$n$ is the degree of differential operator $L_1$;
$pp$ is the kind of the Riccati equation ($pp = 1$ and $pp = 2$);
$x$ is the independent variable.
Output: The generalized Riccati equation of first kind ($pp = 1$) and of second
    kind ($pp = 2$).
R1: If $pp = 1$ then go to R2 else go to R3;
R2: $\ll L2 = (D-\alpha)y$;
    $b$ is the array of coefficients of operator $L2$;
    $m := 1$ ;  $N_1(\alpha) :=$ DIFRESULT$(a, n, b, m, x)$;  return $N_1(\alpha) \gg$;
R3: $\ll L_1^*$ is
        $$L_1^* \equiv \sum_{r=0}^{n} \sum_{k=r}^{n} (-1)^k \binom{k}{r} a_k^{(k-r)} y^{(k)},$$
$a^*$ is the array of coefficients of differential operator $L_1^*$,  $L_2^* = (D+\alpha)y$;
$b^*$ is the array of the coefficients of differential operator $L_2^*$;
$m := 1$ ;  $N_2(\alpha) :=$ DIFRESULT$(a^*, n, b^*, m, x)$;  return $N_2(\alpha) \gg$;
```

5 ODE Resolved by Algebraic Means

5.1 Linear Differential Equations with Exponential Solutions of Type $y = \exp(-\alpha x)$, $\alpha = \text{const}$

The necessary and sufficient conditions for existence of such solutions is the pseudo-characteristic Riccati equation of first kind $L(\alpha) = 0$ and of second kind $L^*(\alpha)$ to have solution $\alpha = \text{const}$, i.e. $\text{RRes}\,(L, D+\alpha) = 0$ or $\text{RRes}\,(L^*, D-\alpha) = 0$. Then the linear differential equation

$$Ly = \sum_{k=0}^{n} a_k D^k y = 0, \qquad a_n \neq 0, a_k \in C_I^k. \tag{21}$$

takes the form $Ly = L_2(D + \alpha)y$, where L_2 is differential operator, $\text{ord}\,(L_2) = n - 1$, and the differential equation has exponential solution of the type $y = \exp(-\alpha x)$. In the case of linear equation of second order

$$Ly = a_2(x)y'' + a_1(x)y' + a_0(x)y = 0, \qquad a_k \neq 0, \quad a_k \in C_I^k, \quad k = 0 \div 2, \tag{22}$$

the problem is to find the coefficients of factorization α_1, α_2 in $Ly = (D - \alpha_1)(D - \alpha_2)y = 0$, where $\alpha_1 = \text{const}$ or $\alpha_2 = \text{const}$. i.e. we consider the classes of equations for which the associated Riccati equation of first type or of second type in terms of α has constant solutions.

Problem 3. Consider eq. (21) with LODE of degree n. Find conditions which ensure that eq. (21) has exponential solution $y = \exp(-\alpha x)$, where $\alpha = \text{const}$..

Algorithm 3. In our program the problem 3 is solved by the procedure

LIVDIF(a, d, n, x).

Input

a is the array of coefficients of operator L;
d is nonhomogeneous part of equation Ly;
n is the degree of differential operator L;
x is the independent variable.

Output If we find the particular solution of the form $y = e^{-\alpha x}$, $\alpha = \text{const}$ and the differential equation is of degree two, then the coefficients of factorization α_1, α_2 and the fundamental system of solutions $y_1(x)$, $y_2(x)$ are obtained. If the degree of LODE is $n > 2$, then the degree of LODE is reduced by 1. If the equation has no exponential solution the message is received.

D1: Algorithm 2 for finding the generalized Riccati equation of first type $N_1(\alpha)$, $N_1(\alpha) := \text{RICCATI}(a, n, 1, x)$;

D2: $L_1(\alpha) = \sum_{k=0}^{n} a_k \alpha^{(k)} = 0$ is the pseudo-characteristic equation of first kind;

D3: $M_1(\alpha) = N_1(\alpha) - L_1(\alpha) = 0$; $M_1(\alpha)$ - reduced equation of first kind;

D4: Subalgorithm for finding constant solution of algebraic equation $L_1(\alpha)$ in terms of α.
Output: constant solution α_0, or
obtain message that the solution of this type does not exist,

D5: If α_0 exists and is solution of $M_1(\alpha) = 0$ then go to D11 else go to D6;
D6: Algorithm 2 for finding the generalized Riccati equation of second type $N_2(\alpha) := \text{RICCATI}(a, n, 2, x)$;
D7:

$$L_2(\alpha) = \sum_{r=0}^{n} \sum_{k=r}^{n} (-1)^k \binom{k}{r} a_k^{(k-r)} \alpha^k \tag{23}$$

($L_2(\alpha)$-pseudo-characteristic equation of second type)
D8:

$$M_2(\alpha) = N_2(\alpha) - L_2(\alpha) \tag{24}$$

($M_2(\alpha)$-reduced Riccati equation of second type)
D9: Subalgorithm of finding constant solution of algebraic equation $L_2(\alpha)$ in terms of α
Output: the constant solution α_0, or a message that such a solution does not exist)
D10: If α_0 exists and is solution of $M_2(\alpha) = 0$ then go to D11 else go to D13;
D11: If $n = 2$ then go to D12 else go to D13;
D12: $\ll \alpha_2 = \alpha_0$; $\alpha 1 = -(a_1 + \alpha_2)$;

$$y_1 = \exp(D^{-1}\alpha_1), \qquad y_2 = y_1 D^{-1} \exp(D^{-1}(\alpha_2 - \alpha_1)); \tag{25}$$

Message "The coefficients of factorization α_1 and α_2 and fundamental solutions y_1 and y_2 are obtained ";
D13: $\ll y_1 = \exp(D^{-1}\alpha_0)$, Message "The degree of differential equation is reduced by 1"; New LODE := Sub($y = y_1 D^{-1} z, Ly$); In equation

$$Ly = \sum_{k=0}^{n} a_k D^k y = 0, \quad a_n \neq 0, \quad a_k \in C_I^k \tag{26}$$

we make the change of variables $y = y_1 D^{-1} z$ where z is new variable) Return New LODE ;
D14: Message "The solution of exponential type does not exist".

Algorithm 4. Subalgorithm for finding of constant solution of algebraic equation $L(\alpha)$ in terms of α. In our program this subalgorithm is realized by procedure HAREQ($L(\alpha), \alpha, x$);
Output: The constant solution α_0 of the equation $L(\alpha)$, or a message that such solution does not exists.

Example 2

$$Ly = y'' + (8 + \sin(x)^2)y' + 8\sin(x)^2 y = 0; \quad (') = d/dx \tag{27}$$

In the array $a(n)$, $n = 0 \div 2$, we write the coefficients of the equation

$$a(2) := 1; \quad a(1) := 8 + \sin(x)^2; \quad a(0) := 8\sin(x)^2; \quad n := 2;\, d := 0; \tag{28}$$

Using the procedure LIVDIF(a, d, n, x); we seek for a particular solution of the following type $y = \exp(-\alpha x)$, where $\alpha = \text{const.}$, and if the differential equation is of degree $n = 2$, then we find the coefficients of factorization α_1, α_2 and the fundamental system of solutions $y_1(x)$, $y_2(x)$, also the general solution. Then we have the solution of differential equation of type $y = \exp(-8x)$, and the coefficients of factorization are

$$\alpha_1 = 8; \quad \alpha_2 = \sin(x)^2, \tag{29}$$

and due to the fact that the equation is of second degree we may also find a second solution

$$y_2 = y_1 D^{-1} \exp(D^{-1}(\alpha_2 - \alpha_1)), \tag{30}$$

and general solution

$$y = C_1 y_1 + C_2 y_2. \tag{31}$$

5.2 LODE with Polynomial Solutions

A necessary and sufficient conditions which ensures that the equation (21) allows polynomial solutions of degree $p \leq m$ is $\text{RRes}(L, D^{m+1}) = 0$. Thus using the differential resultant of operators L and D^{m+1} we find the degree of the polynomial solution of LODE (it it exists). After that by the method of indefinite coefficients we find the coefficients of this polynomial and if the degree of LODE is $n = 2$ we may obtain the general solution; when $n > 2$ we reduce by 1 the degree of LODE.

Problem 4. Consider eq. (21) with LODE of degree n and find its polynomial solution.

Algorithm 5. To solve the problem 4 we apply the procedure
POLDIF(a,n,x,hs);

Input
a is the array of coefficients of LODE $Ly = 0$,
n is the degree of differential operator L,
x is independent variable,
hs is the maximal degree of the polynomial solution,
Output:

If there exists polynomial solution of degree $p \leq hs$, then in the case of $n = 2$ we find the general solution, and when $n > 2$ the degree is reduced by 1. If there is no solution one gets the message that polynomial solution does not exist.

```
P1: $i := 1$;
P2: Algorithm 1 for finding the differential resultant $\text{RRes}\,(L, D^i)$;
P3: If $\text{RRes}\,(L, D^i) = 0$, then go to P6 else go to P4;
P4: If $i > hs$ then go to P12 else go to P5;
P5: $\ll i := i + 1;$ go to P2 $\gg$;
P6: P:= $i - 1$;
```

P7: Message "P is the degree of polynomial solution";

P8: Subalgorithm for finding the coefficients of the polynomial solution (Output-polynomial solution)

P9: If $n = 2$ then go to P10 else go to P11;

P10: Find the fundamental system of LODE.
$\ll y_1 =$ polynomial solution;

$$y_2 = y_1 D^{-1} \exp\left(-D^{-1} a_1\right) y_1^{-2}$$

Return $y_2, y_1 \gg$;

P11: The degree of LODE is reduced by 1.
$\ll y_1 =$ polynomial solution;
New LODE := Sub($y = y_1 D^{-1} z, Ly$), where in equation

$$Ly = \sum_{k=0}^{n} a_k D^k y = 0, \quad a_n \neq 0, \quad a_k \in C_I^k \tag{32}$$

we make the change of variables $y = y_1 D^{-1} z$ where z is the new variable
Return New LODE $\gg$;

P12: Message "The polynomial solution of the LODE ";

Algorithm 6. Subalgorithm for finding the coefficients of the polynomial solution of the LODE.

In our program this subalgorithm is realized by the procedure DEPOLDIF(a, n, x, p);

Input:
a is the array of coefficients of the differential equation $Ly = 0$
n is the degree of differential operator L,
x is independent variable,
p is the degree of polynomial solution of $Ly = 0$.

Output: The coefficients of the polynomial solution of by the method of indefinite coefficients.

Example 3

$$Ly = x^2(\ln(x) - 1)y'' - xy' + y = 0,$$

The array $a(n), n = 0 \div 2$, contain the coefficients of the given LODE.

$$a(2) := x^2(\ln(x) - 1); \qquad a(1) := -x; \qquad a(0) := 1; \quad n := 2; \quad hs := 3;$$

By using the procedure POLDIF(a, n, x, hs) we find the fundamental system of solution $y1 := x; y2 := -\ln(x)$. Then the general solution is given by: $y := C_1 x + C_2 \ln(x)$;

5.3 The Equations in Terms of Exact Differentials

The equation (21) will be expressed in terms of exact differentials if it has the following form

$$Ly = D\sum_{k=0}^{n-1} b_k D^k y = 0, \tag{33}$$

where

$$a_n = b_{n-1}(x), \quad a_0 = b_0'(x), \quad a_{k+1} = (b_k + b_{k+1}'(x)), \tag{34}$$

and k take values from 0 to $n-2$. The necessary and sufficient conditions that the left hand side of the equation $Ly = 0$ becomes exact differential is

$$\sum_{k=0}^{n} (-1)^k a_k^{(k)}(x) = 0. \tag{35}$$

If the equation (21) is expressed in terms of exact differentials, and if $n = 2$ we may find the general solution of this equation. For $n > 2$ we may find first integral.

Problem 5. Consider eq. (21) with LODE of degree n. Check if this LODE takes the form of exact differentials. If this is true then find the general solution (when $n = 2$) or a first integral (when $n > 2$).

Algorithm 7. To solve the problem 5 we apply the procedure
EQDIF(a,n,f(x),x);

Input:
a is the array of coefficients of LODE
n is the degree of differential operator L,
dl is the nonhomogeneous part of given LODE
x is independent variable,
Output:

Check if the given equation is in exact differentials. If this is true find the general solution in the case $n = 2$ and first integral if $n > 2$; otherwise output the message that the equation is not in exact differentials

E1:

$$\boldsymbol{\partial} := \sum_{k=0}^{n} (-1)^k a_k^{(k)}(x) = 0, \tag{36}$$

E2: If $\boldsymbol{\partial} = 0$ then go to E4 else go to E3;
E3: Message: "This equation is not in exact differentials".
E4: $\ll b(n-1) := a(n)$; For $k := (n-2)$ Step (-1) Until 0 Do

$$b(k) := a(k+1) - b'(k+1). \gg \tag{37}$$

E5: If $n = 2$ Then Go To E6 Else Go To E7;

E6:

$$y = \exp\left(-D^{-1}a(1))(C_2 + D^{-1}(D^{-1}f(x) + C_1)\exp(D^{-1}a(1))\right); \tag{38}$$

(C_1, C_2 - are constants of integration, y - general solution of LODE)
E7:

$$L_2 y = D\sum_{k=0}^{n-1} b_k D^k y. \tag{39}$$

6 Differential Resultant and Algebra of Commuting Differential Operators

The important results about the algebra of commuting differential operators (ACDO) are outlined in [25,4,21,6,2,9]. The relation of ACDO with the hierarchy of stationary soliton equations are discussed in [30,29,24,26,27,28]. For basic introductory course to hierarchy of stationary soliton equations we highly recommend the new book [14]. The fundamental relation of completely integrable dynamical systems and algebraic curves are outlined in [7,10,1,16,12,13,15]. The relation of a great number of completely integrable dynamical systems to Hill's equation are discussed in [22]. The investigation of soliton equations using computer algebra is given in great details in [19,20,17,18]. We follow these results in considering our basic examples.

Example 4. Fordy example, stationary KdV equation of degree 5 [8,14].

$$H = \frac{1}{2}\left(p_1^2 + p_2^2\right) + \frac{1}{2}q_1 q_2^2 + q_1^3. \tag{40}$$

The Hénon-Heiles system with Hamiltonian (40) allows stationary Lax representation (8) with

$$\begin{aligned} &A = D^2 + q_1 - z, \qquad\qquad (41)\\ &B = 16D^5 + 40q_1D^3 + 60p_1D^2 - (120q_1^2 + 25q_2^2)D - 60q_1p_1 - 15q_2p_2 - \lambda. \end{aligned}$$

The relevant algebraic curve is

$$z^2 = -256\lambda^5 - 32E\lambda^2 + 8K\lambda \tag{42}$$

$$E = \frac{1}{2}\left(p_1^2 + p_2^2\right) + q1^3 + \frac{1}{2}q_1q_2^2, \qquad K = q_2p_1p_2 - q_1p_2^2 + q_2^4 + 1/2q_1^2q_2^2.$$

This example show the abilities of the differential resultant to find the first integrals and algebraic curve associated to given linear differential operators.

Example 5. Fordy example, stationary Sawada-Kotera equation [8].

$$H = \frac{1}{2}\left(p_1^2 + p_2^2\right) + \frac{1}{2}q_1q_2^2 - \frac{1}{6}q_1^3. \tag{43}$$

We associate to this Hamiltonian (43) the following linear differential operators

$$\begin{aligned} A &= D^3 + q_1 D - z, \qquad (44)\\ B &= 9D^5 + 15q_1 D^3 + 15p_1 D^2 - (5q_1^2 + 10q_2^2)D - 60q_1p_1 - 15q_2p_2 - \lambda, \end{aligned}$$

The associated algebraic curve obtained by differential resultant method has the form

$$\begin{aligned} & z^3 = 729\lambda^5 - 162E\lambda^3 + K^2\lambda, \\ & E = \frac{1}{2}\left(p_1^2 + p_2^2\right) + \frac{1}{6}q1^3 + \frac{1}{2}q_1q_2^2, \qquad K = 3p_1p_2 + \frac{3}{2}q_2q_1^2 + \frac{1}{2}q_2^3. \end{aligned}$$

7 Conclusion

We have described briefly the solutions by REDUCE4 and MAPLE13 to several problems of constructing commuting LODE. More details can be found by downloading the file https://stardust.inrne.bas.bg/code/diff-res-pack.pdf or https://stardust.inrne.bas.bg/code/diff-res-pack.tex.

Acknowledgements

We are grateful to anonymous referees for careful reading of the manuscript and for useful suggestions.

References

1. Ablowitz, M.J., Clarkson, P.A.: Solutions, Nonlinear Evolution Equations and Inverse Scattering. London Mathematical Society Lecture Notes on Mathematics, vol. 149. Cambridge University Press, Cambridge (1991)
2. Berkovich, L.M., Berkovich, F.L.: Transformation and factorization of second order linear ordinary differential equations and its implementation in Reduce, Samara (2002)
3. Berkovich, L.M., Gerdt, V.P., Kostova, Z.T., Nechaevsky, M.L.: Second Order Reducible Linear Differential Equations. In: Applied Packages. Software for Methematical Simulation, pp. 9–24. Nauka Publishers, Moscow (1992)
4. Cherednik, I.V.: Differential equations of the Baker–Akhiezer functions of algebraic curves. Funct. Anal. Appl. 12, 195–203 (1978)
5. Deconinck, B., Heil, M., Bobenko, A., van Hoeij, M., Schmies, M.: Computing Riemann Theta Functions. Mathematics of Computation 73, 1417–1442 (2004)
6. Dubrovin, B.A., Matveev, V.B., Novikov, S.P.: Nonlinear equations of the KdV type, finite gap linear operators and abelian varieties. Uspekhi Matem. Nauk 31(1), 55–135 (1976)
7. Enol'skii, V.Z., Kostov, N.A.: On the geometry of elliptic solitons. Acta Appl. Math. 36, 57–86 (1994)
8. Fordy, A.P.: The Hénon-Heiles system revisited. Physica 52D, 201–210 (1991)
9. Ganzha, V.G., Vorozhtsov, E.V.: Computer-aided analysis of difference schemes for partial differential equations. Wiley, Chichester (1996)

10. Gesztesy, F.: On the Modified Korteweg–de-Vries Equation. In: Goldstein, J.A., Kappel, F., Schappacher, W. (eds.) Differential Equations with Applications in Biology, Physics, and Engineering, pp. 139–183. Marcel Dekker, New York (1991)
11. Gesztesy, F., Weikard, R.: Spectral deformations and soliton equations. In: Ames, W.F., Harrell II, E.M., Herod, J.V. (eds.) Differential Equations with Applications to Mathematical Physics, pp. 101–139. Academic Press, Boston (1993)
12. Gesztesy, F., Weikard, R.: Treibich-Verdier potentials and the stationary (m)KdV hierarchy. Math. Z. 219, 451–476 (1995)
13. Gesztesy, F., Weikard, R.: On Picard potentials. Diff. Int. Eqs. 8, 1453–1476 (1995)
14. Gesztesy, F., Holden, H.: Soliton equations and their algebro-geometric solutions. In: (1+1)-dimensional continuous models Cambridge Studies in Advanced Mathematics, vol. 79. Cambridge University Press, Cambridge (2003)
15. Gesztesy, F., Weikard, R.: Floquet theory revisited. In: Knowles, I. (ed.) Differential Equations and Mathematical Physics, pp. 67–84. International Press, Boston (1995)
16. Gesztesy, F., Weikard, R.: Lamé potentials and the stationary (m)KdV hierarchy. Math. Nachr. 176, 73–91 (2006)
17. Hereman, W., Angenent, S.: The Painleve test for nonlinear ordinary and partial differential equations. MACSYMA Newsletter 6, 11–18 (1989)
18. Hereman, W., Zhuang, W.: Symbolic computation of solution with MACSYMA. In: Ames, W.F., van der Houwen, P.J. (eds.) Computational and Applied Mathematics II: Differential Equations, pp. 287–296. North-Holland, Amsterdam (1992)
19. Hereman, W.: Review of symbolic software for the computation of Lie symmetries of differential equations. Euromath Bulletin 1(2), 45–82 (1994)
20. Hereman, W.: Symbolic software for Lie symmetry analysis. In: Ibragimov, N.H. (ed.) CRC Handbook of Lie Group Analysis of Differential Equations, ch. 13, vol. 3. CRC Press, Boca Raton (1995) (in Press)
21. Its, A.R.: Inversion of hyperelliptic integrals, and integration of nonlinear differential equations. Vest. Leningr. Gos. Univ. 7(2), 37–46 (1976)
22. Kostov, N.A.: Quasi-periodical solutions of the integrable dynamical systems related to Hill's equation. Lett. Math. Phys. 17, 95–104 (1989)
23. Kostov, N.A., Kostova, Z.T.: Nonlinear waves, differential resultant, computer algebra and completely integrable dynamical systems, arXiv:solv-int/9904015
24. Krichever, I.M.: Integration of nonlinear equations by the methods of algebraic geometry. Funct. Anal. Appl. 11, 12–26 (1977)
25. Krichever, I.M.: Methods of algebraic geometry in the theory of non-linear equations. Russ. Math. Surv. 32(6), 185–213 (1977)
26. Krichever, I.M.: Elliptic solutions of the Kadomtsev-Petviashvili equation and integrable systems of particles. Funct. Anal. Appl. 14, 282–290 (1980)
27. Krichever, I.M.: Nonlinear equations and elliptic curves. Revs. Sci. Technology 23, 51–90 (1983)
28. Krichever, I.M.: Elliptic solutions of nonlinear integrable equations and related topics. Acta Appl. Math. 36, 7–25 (1994)
29. Li, Z., Schwarz, F., Tsarev, P.: Factoring systems of linear PDEs with finite-dimensional solution spaces. Journal of Symbolic Computation, Special issue: International symposium on symbolic and algebraic computation (ISSAC 2002) archive 36(3-4), 443–471 (2003)
30. Mikhailov, A.V., Shabat, A.B., Sokolov, V.V.: The symmetry approach to classification of integrable equations. Uspekhi Math. Nauk 42, 3–53 (1987)

Generic, Type-Safe and Object Oriented Computer Algebra Software

Heinz Kredel and Raphael Jolly

IT-Center, University of Mannheim, Germany and Databeans, Paris, France
kredel@rz.uni-mannheim.de, raphael.jolly@free.fr

Abstract. Advances in computer science, in particular object oriented programming, and software engineering have had little practical impact on computer algebra systems in the last 30 years. The software design of existing systems is still dominated by ad-hoc memory management, weakly typed algorithm libraries and proprietary domain specific interactive expression interpreters. We discuss a modular approach to computer algebra software: usage of state-of-the-art memory management and run-time systems (e.g. JVM) usage of strongly typed, generic, object oriented programming languages (e.g. Java) and usage of general purpose, dynamic interactive expression interpreters (e.g. Python) To illustrate the workability of this approach, we have implemented and studied computer algebra systems in Java and Scala. In this paper we report on the current state of this work by presenting new examples.

1 Introduction

The great success of computer algebra systems like Maple or Mathematica in engineering and science has lead to reduced efforts in the construction of software components suitable for algorithmic and software engineering research. Maple and Mathematica and numerical systems like Matlab, are currently required in the daily work of many engineers and scientists. Therefore, the focus, marketing and presentation of these systems is oriented to the needs of engineers and scientists: tool boxes with strong emphasis on documentation and education. On the other side there is demand for software systems suitable for algorithmic and software engineering research. For this research it is essential to control every aspect of the software. The critique on systems like Mathematica or Maple from this research groups vary from inappropriate or insufficient implementations of algorithms without the ability to repair or extend the code by better implementations [1] to insufficient data type support [2,3], and requirements for object oriented implementations [4]. The design chosen 30 years ago makes it moreover difficult to evolve these systems according to current needs or ideas [5,6]. Furthermore, the concepts of Maple and Mathematica are challenged by Google web-application developments [7] in the same way as Microsoft Word is challenged by Google web documents and toolkits [8]. The Eclipse Rich Client Platform [9] is yet another alternative [10]. Additionally the software architecture of these systems may not be suited for multi-CPU and Grid-Computing with 10.000 to 100.000 processing nodes.

Contemporary open source computer algebra software like Singular ([11,12]), Kant, PariGP, Gap and others partially suffer from similar software architectural origins as Maple or Mathematica. These systems are monolithic, highly integrated software systems with nearly no public interfaces besides the command line shell. The low-level run-time system which provides mainly memory management with automatic garbage collection is tightly coupled with the next level of arithmetic algorithm implementations like integer arithmetic or polynomial arithmetic. This structure is a great obstacle in the reuse of parts of the implementations in other systems or projects. Nevertheless there is a very clever attempt to provide a common interface to some of these systems with the Sage project [13].

In this situation we undertook the experiment to rewrite major portions of a computer algebra system in object oriented programming languages. We could have joined efforts with Axiom/Aldor to a certain extent for our object oriented approach. However, our two main motivations to use Java and Scala are: first, to explore to which extent these programming languages are realy suited for this task and second, have platforms which support Cloud computing (e.g. [14]) and Smart devices (e.g. [15]).

1.1 Related Work

There is not much work published on object oriented programming for algebraic or symbolic algorithm implementation. There is more work published on type systems for computer algebra or abstract data type (ADT) approaches to computer algebra. The main question is the expressiveness required to implement algebraic algorithms. However, the requirements for libraries and interactive parts are constantly mangled and not cleanly separated as in our approach. A first paper on CAS with SmallTalk [16] and the ongoing work of the Axiom developers can to some extend be viewed as object oriented [3]. Newer approaches start with [17] in Common Lisp, then using C++ [1,18]. Early considerations of Java in computer algebra and symbolic computation [19,20,21]. Newer approaches using Java are [22,23] or [24,25], or our approaches starting with [26,27]. An object oriented but non-Java approach to computer algebra exists as part of the Focalize project [28]. Type-safe design considerations in computer algebra are described in [29,30,3,31,32,17]. Generic programming issues are discussed for example in [33,34,35] and the references therein. Interoperability via OpenMath is discussed in [36,37,38,39,40]. Further thoughts on the future of computer algebra systems see [6,5]. Further related work is mentioned in the paper as required, and in the section on future work.

1.2 Outline

In section 2 we discuss design considerations for object oriented computer algebra and symbolic computation software. Examples for the construction of such systems are presented in section 3. Section 4 shows some future work and the final section draws some conclusions.

2 Design Considerations

The proposed software architecture builds on other software projects as much as possible. Only the parts specific to computer algebra are to be implemented. We identify three major parts for computer algebra software.

– run-time infrastructure with memory management,
– statically typed object oriented algorithm libraries,
– dynamic interactive scripting interpreters.

We discuss the first points in the following subsections. For the third point see our articles [41,42].

2.1 Run-Time Systems

Run-time systems with *automatic memory management* can be taken from virtual machines, like the Java JVM or the .NET CLR. Advantages of virtual machines:

– constant maintenance and improvements,
– more opportunities for code optimization with just-in-time compilers,
– memory management with automatic garbage collection,
– exception and security constraint handling,
– independence of computer hardware and optimization requirements,
– suitable for multi-CPU and distributed computing.

Disadvantages of virtual machines are the dependency on the chosen platform.

Software development in computer algebra must deal with unpredictable dynamic memory requirements of the algebraic and symbolic objects occurring in a computation, so a crucial necessity is the availability of automatic memory management in the run-time system.

2.2 Object Oriented Software

The second building block for the design and implementation of computer algebra systems, is *object oriented programming methodology.* It can be characterized as follows

– usage of contemporary (object oriented) software engineering principles,
– modular software architecture, consisting of
  • usage of existing implementations of basic data structures like integers or lists
  • generic type safe algebraic and symbolic algorithm libraries
  • thread safe and multi-threaded library implementations
  • algebraic objects transportable over the network
– high performance implementations of algorithms with state of the art asymptotic complexity but also fast and efficient for small problem sizes,
– minimizing the 'abstraction penalty' which occurs for high-level programming languages compared to low-level assembly-like programming languages.

Our research in this new direction is the design and the construction of object oriented computer algebra libraries in Java and Scala, called JAS and ScAS. The main concepts and achievements have recently been presented and published in a series of computer science and computer mathematics conferences [27,43,44,45,46,47,48,49,50] and on the projects Web-pages [51,26].

3 Examples

In this section we discuss some examples of the object oriented approach to computer algebra software. First we discuss the design of the basic interfaces and classes for algebraic structures, namely rings and polynomials. We present the ScAS design, as the similar JAS design has been presented elsewhere. In the next sub-section, we discuss the design of algorithms for polynomial factorization. For the performance of Java implementations of multivariate polynomial arithmetic, greatest common divisor computation and comprehensive Gröbner bases construction see [45,47,48]. Problems of object oriented programming for which we have not found satisfactory solutions yet will be covered in a subsequent publication. Some knowledge of Java and Scala is required for the understanding of this section.

3.1 Ring Elements and Polynomials

The type-safe design of the basic structural interfaces in ScAS makes use of "traits", the Scala equivalent of Java's interfaces (i.e. with multiple inheritance), but with the added feature that some methods can be implemented, thus taking code re-use a step further. These traits are type-parametrized, which provides modularity while forbidding arithmetic operations on incompatible types at compile time. The root of the hierarchy is the trait `Element`. Each trait in the hierarchy comes with a `Factory` which is used to obtain new instances of its type and do various operations on these.

```
object Element {
  trait Factory[T <: Element[T]] {
    def random(numbits: Int)(
        implicit rnd: scala.util.Random): T
  }
}
trait Element[T <: Element[T]] extends Ordered[T] { this: T =>
  val factory: Element.Factory[T]
  def equals(that: T) = this.compare(that) == 0
  def ><(that: T) = this equals that
  def <>(that: T) = !(this equals that)
}
```

The notation `this: T =>` is called a self-type and is used to specify the future type of "this" which is currently not known as the type is abstract and cannot be instantiated. The definitions of `><` and `<>` must be made because the operators `==` and `!=`, which in Scala are normally routed to `equals`, are not type-parametrized and have an argument of type `Any`. Next we have traits for commutative (abelian) additive groups, (multiplicative) semi-groups, and monomials. Scala provides operator overloading, resulting in a natural mathematical notation.

```
object AbelianGroup {
  trait Factory[T <: AbelianGroup[T]] extends Element.Factory[T] {
    def zero: T
  }
}
trait AbelianGroup[T <: AbelianGroup[T]]
    extends Element[T] { this: T =>
  override val factory: AbelianGroup.Factory[T]
  def isZero = this >< factory.zero
  def +(that: T): T
  def -(that: T): T
  def unary_+ = this
  def unary_- = factory.zero - this
  def abs = if (signum < 0) -this else this
  def signum: Int
}
```

Some methods like `abs` can already be implemented in terms of other ones which remain abstract for now (`signum`, `-`). If `signum` is meaningful in the respective ring depends on the ring. It is a design decision, explained for example in [47], to let interfaces define methods which may fail in certain rings.

```
trait SemiGroup[T <: SemiGroup[T]] extends Element[T] { this: T =>
  def *(that: T): T
}
```

`Semigroup` has no corresponding `Factory` as there is no additional feature compared to `Element`. Thus `Monoid.Factory` below will inherit directly from `Element.Factory`, whereas `Monoid` inherits from `SemiGroup`:

```
object Monoid {
  trait Factory[T <: Monoid[T]] extends Element.Factory[T] {
    def one: T
  }
}
trait Monoid[T <: Monoid[T]] extends SemiGroup[T] { this: T =>
  override val factory: Monoid.Factory[T]
  def isUnit: Boolean
  def isOne = this >< factory.one
  def pow(exp: BigInt) = {
    assert (exp >= 0)
    (factory.one /: (1 to exp.intValue)) {
      (l, r) => l * this
    }
  }
}
```

The method `isOne` tests exactly for the 1 in the ring, `isUnit` tests if the element is an associate of 1 (whence if it is invertible). `pow` is implemented here, which will save these lines of code everywhere Monoid will be inherited. Note, such elegant solutions are not possible with Java interfaces and Java abstract classes are also of no help in such situations. Binary exponentiation is not shown to simplify the example.

```
object Ring {
  trait Factory[T <: Ring[T]]
      extends AbelianGroup.Factory[T] with Monoid.Factory[T] {
    def characteristic: BigInt
  }
}
trait Ring[T <: Ring[T]] extends AbelianGroup[T]
    with Monoid[T] { this: T =>
  override val factory: Ring.Factory[T]
}
```

`Ring` inherits multiply from `AbelianGroup` and `Monoid`, and declares a `characteristic`. Below we outline the implementation of a `Polynomial` class from the basic structures we have just defined.

```
object Polynomial {
  class Factory[C <: Ring[C]](val ring: C,
      val variables: Array[Variable],
      val ordering: Comparator[Int])
      extends Ring.Factory[Polynomial[C]] {
    def generators: Array[Polynomial[C]]
    def apply(value: SortedMap[Array[Int], C])
       = new Polynomial(this)(value)
    override def toString: String
  }
}
class Polynomial[C <: Ring[C]](
    val factory: Polynomial.Factory[C])(
    val value: SortedMap[Array[Int], C])
    extends Ring[Polynomial[C]] {
  def elements: Iterator[Pair[Array[Int], C]]
  def headTerm = elements.next
  def degree: Int
  def isUnit = this.abs.isOne
  override def toString: String
}
```

To be really accurate, the above Polynomial definition in ScAS is abstract and is in fact a trait, just like Ring from which it inheritates. This is be able to sublclass it to a SolvablePolynomial (i.e. non-commutative) in addition to a regular Polynomial. The abstract Polynomial has a self-type parameter like the other abstract types, in order to preserve type-safety. The definitions are then sketched as follows:

```
object Polynomial {
  trait Factory[T <: Polynomial[T, C],
              C <: Ring[C]] extends Ring.Factory[T] {
    def multiply(w: T, x: Array[Int], y: C) = { // commutative
    }
  }
}
trait Polynomial[T <: Polynomial[T, C], C <: Ring[C]]
      extends Ring[T]

object SolvablePolynomial {
  trait Factory[T <: Polynomial[T, C], C <: Ring[C]]
        extends Polynomial.Factory[T, C] {
    override def multiply(w: T, x: Array[Int], y: C) = {
                // non-commutative case
    }
  }
}
```

The various mechanisms above allow multiple options to be combined mostly independently with minimal code duplication. Among the possible dimensions of parametrization, we have implemented:

- the coefficient type (`C`, above)
- the underlying data structure (array, list, tree) of the polynomial, involving the type of the “value” field (`SortedMap[Array[Int], C]` above) which can be parametrized trough a Scala abstract type member
- the type of the exponents (`P` in ScAS code, not shown)
- we have made a prototype implementation of polynomials with different algorithms for gcd computation : the type of the polynomial (factory) tells whether the gcd algorithm is of simple, primitive, or subresultant kind. This is intended as an improvement on JAS, see the next section for how this is implemented there.
- whether or not the polynomial is a `SolvablePolynomial`

We are studying the implementation for:

- an improved parametrization of the type of the exponents. It is typically `Int` or `Long`, so this is a Java primitive type. Unlike Java, Scala allows to parametrize over these. But this is currently made through boxing and unboxing, with performance impact. Scala 2.8 will provide “type specialization”

whereby a class with a primitive type as parameter will be implemented in an optimized way.
– the list of variables and the ordering. Currently it is possible to arithmetically mix polynomials in different sets of variables, which is not optimal. Parameterizing this item requires some kind of dependent type, as described in section 7.3 of [47].

3.2 Unique Factorization Domains

To further exemplify the usefulness of object oriented computer algebra software development we studied the implementation of some non-trivial examples for library design. We limit the discussion to algorithms for multivariate polynomials in (constructive) unique factorization domains. Examples, like Gröbner bases, comprehensive Gröbner bases, univariate power series and elementary integration of rational functions are discussed elsewhere [51,49,48]. For the mathematical background see some text books on the topic, like [52]. We give an overview of the respective interfaces and classes of JAS and then compare it to the approach of Scratchpad (now Axiom) [29,30].

Figure 1 shows an UML overview diagram of the involved interfaces and classes for the computation of greatest common divisors of multivariate polynomials. In case an algorithm is only meaningful for univariate polynomials, the

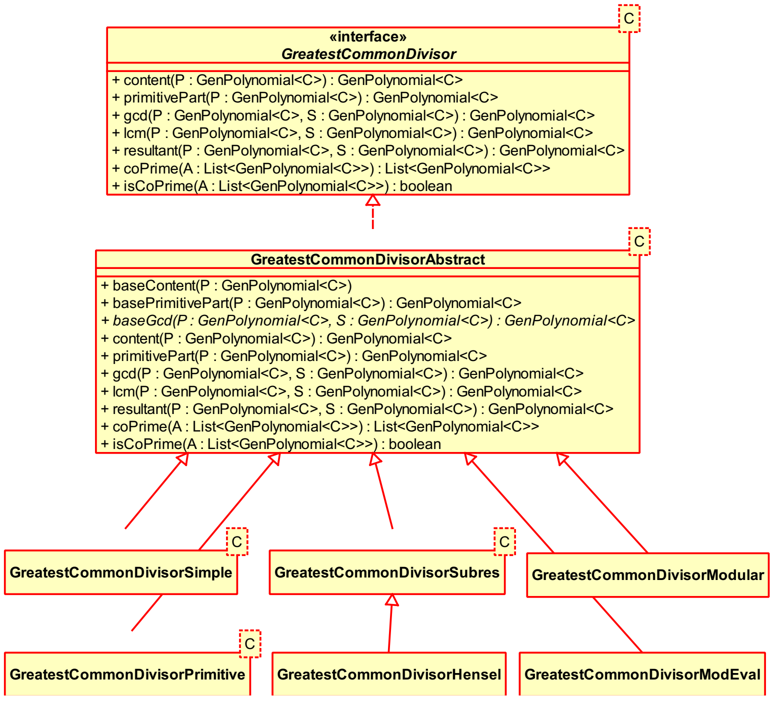

C
«interface»
GreatestCommonDivisor
+ content(P : GenPolynomial<C>) : GenPolynomial<C>
+ primitivePart(P : GenPolynomial<C>) : GenPolynomial<C>
+ gcd(P : GenPolynomial<C>, S : GenPolynomial<C>) : GenPolynomial<C>
+ lcm(P : GenPolynomial<C>, S : GenPolynomial<C>) : GenPolynomial<C>
+ resultant(P : GenPolynomial<C>, S : GenPolynomial<C>) : GenPolynomial<C>
+ coPrime(A : List<GenPolynomial<C>>) : List<GenPolynomial<C>>
+ isCoPrime(A : List<GenPolynomial<C>>) : boolean
C
GreatestCommonDivisorAbstract
+ baseContent(P : GenPolynomial<C>)
+ basePrimitivePart(P : GenPolynomial<C>) : GenPolynomial<C>
+ baseGcd(P : GenPolynomial<C>, S : GenPolynomial<C>) : GenPolynomial<C>
+ content(P : GenPolynomial<C>) : GenPolynomial<C>
+ primitivePart(P : GenPolynomial<C>) : GenPolynomial<C>
+ gcd(P : GenPolynomial<C>, S : GenPolynomial<C>) : GenPolynomial<C>
+ lcm(P : GenPolynomial<C>, S : GenPolynomial<C>) : GenPolynomial<C>
+ resultant(P : GenPolynomial<C>, S : GenPolynomial<C>) : GenPolynomial<C>
+ coPrime(A : List<GenPolynomial<C>>) : List<GenPolynomial<C>>
+ isCoPrime(A : List<GenPolynomial<C>>) : boolean
C
GreatestCommonDivisorSimple
C
GreatestCommonDivisorSubres
GreatestCommonDivisorModular
C
GreatestCommonDivisorPrimitive
GreatestCommonDivisorHensel
GreatestCommonDivisorModEval

Fig. 1. Greatest common divisor classes

respective implementations convert the polynomials to univariate polynomials in the main variable with multivariate coefficients. There are implementations for such recursive representations, which are not shown. We start with an interface `GreatestCommonDivisor`. It defines the method names for a ring with gcd algorithm. First there is the method `gcd()` itself, together with the method `lcm()` to compute the least common multiple of two polynomials. With the help of `gcd()` the algorithms for the content `content()` and the primitive part `primitivePart()` computation can be implemented. The methods `coPrime()` compute lists of co-prime polynomials from given lists of polynomials.

The abstract super class for the implementations is called `GreatestCommon-DivisorAbstract`. It implements nearly all methods defined in the `GreatestCommonDivisor` interface. The abstract methods are `baseGcd()` and `recursiveUnivariateGcd()`. The method `gcd()` first checks for the recursion base, and eventually calls `baseGcd()`. Otherwise it converts the input polynomials to recursive representation, as univariate polynomials with multivariate polynomial coefficients, and calls method `recursiveUnivariateGcd()`.

The concrete implementations come in two flavors. The first flavor implements only the methods `baseGcd()` and `recursiveUnivariateGcd()`, using the setup provided by the abstract super class. There are implementations for various polynomial remainder sequence (PRS) algorithms: simple, monic, primitive and the sub-resultant algorithm (in the respective classes). These implementations are generic for any (UFD) coefficient ring. The second flavor directly implements `gcd()` without providing `baseGcd()` and `recursiveUnivariateGcd()`. The algorithms compute gcds first modulo some suitable prime numbers and then interpolate the result using Chinese remainder algorithms, or in the Hensel case via powers of the prime number. The later versions are only valid for integer or modular coefficient classes `BigInteger`, `ModInteger` or `ModLong`.

An overview for the classes for the squarefree decomposition of multivariate polynomials is shown in figure 2. Again, there is an interface `Squarefree` and an abstract class `SquarefreeAbstract`. The other classes are for coefficients from rings or fields of characteristic zero, and for finite and infinite fields of characteristic $p > 0$. Method `squarefreeFactors()` of the interface decomposes a polynomial into squarefree parts. Method `squarefreePart()` computes the squarefree part of a polynomial. Method `isSquarefree()` test the respective property for a polynomial. `isFactorization()` tests if a given map or list is actually a squarefree decomposition for a given polynomial. The interface has additionally methods `coPrimeSquarefree()` to compute lists of co-prime and squarefree polynomials from given lists of polynomials.

The abstract class `SquarefreeAbstract` implements most of the methods specified in the interface. There are four methods which remain abstract and have to be implemented in sub-classes. For multivariate polynomials the methods `squarefreeFactors()` and `squarefreePart()` and for univariate polynomials the methods `baseSquarefreeFactors()` and `baseSquarefreePart()`. The implementation has to distinguish three major and two sub cases: coefficient fields of characteristic zero, class `SquarefreeFieldChar0`, and coefficients not from a

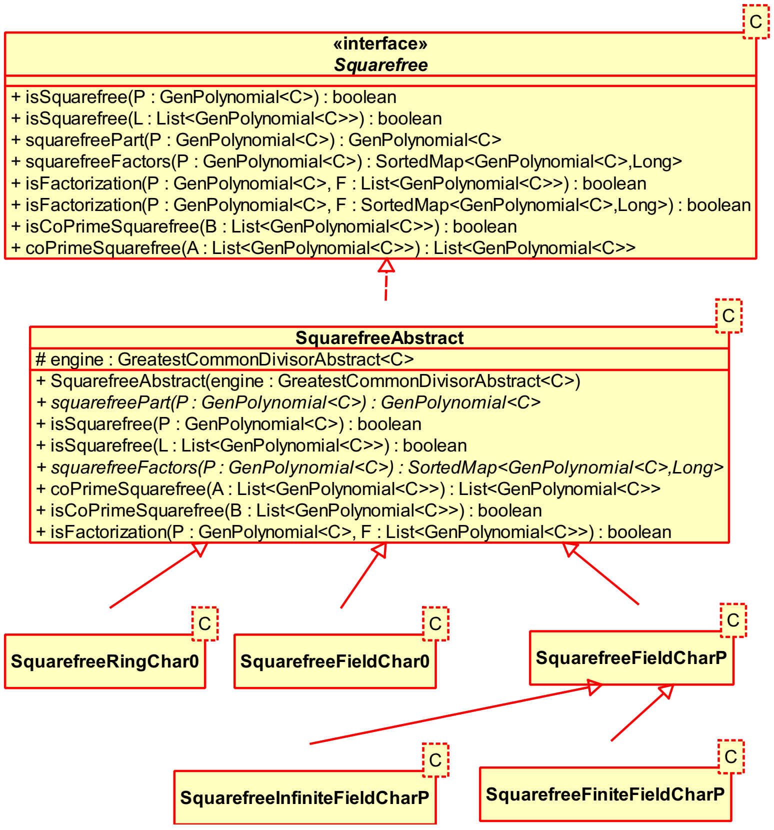

C
«interface»
Squarefree
+ isSquarefree(P : GenPolynomial<C>) : boolean
+ isSquarefree(L : List<GenPolynomial<C>>) : boolean
+ squarefreePart(P : GenPolynomial<C>) : GenPolynomial<C>
+ squarefreeFactors(P : GenPolynomial<C>) : SortedMap<GenPolynomial<C>,Long>
+ isFactorization(P : GenPolynomial<C>, F : List<GenPolynomial<C>>) : boolean
+ isFactorization(P : GenPolynomial<C>, F : SortedMap<GenPolynomial<C>,Long>) : boolean
+ isCoPrimeSquarefree(B : List<GenPolynomial<C>>) : boolean
+ coPrimeSquarefree(A : List<GenPolynomial<C>>) : List<GenPolynomial<C>>
C
SquarefreeAbstract
# engine : GreatestCommonDivisorAbstract<C>
+ SquarefreeAbstract(engine : GreatestCommonDivisorAbstract<C>)
+ squarefreePart(P : GenPolynomial<C>) : GenPolynomial<C>
+ isSquarefree(P : GenPolynomial<C>) : boolean
+ isSquarefree(L : List<GenPolynomial<C>>) : boolean
+ squarefreeFactors(P : GenPolynomial<C>) : SortedMap<GenPolynomial<C>,Long>
+ coPrimeSquarefree(A : List<GenPolynomial<C>>) : List<GenPolynomial<C>>
+ isCoPrimeSquarefree(B : List<GenPolynomial<C>>) : boolean
+ isFactorization(P : GenPolynomial<C>, F : List<GenPolynomial<C>>) : boolean
C
SquarefreeRingChar0
C
SquarefreeFieldChar0
C
SquarefreeFieldCharP
C
SquarefreeInfiniteFieldCharP
C
SquarefreeFiniteFieldCharP

Fig. 2. Squarefree decomposition classes

field `SquarefreeRingChar0`. For coefficient fields of characteristic non-zero, class `SquarefreeFieldCharP`, we need two sub-classes for finite and infinite fields, classes `SquarefreeFiniteFieldCharP` and `SquarefreeInfiniteFieldCharP`.

Figure 3 shows the class layout of an interface, two abstract and several concrete classes for multivariate polynomial factorization. The interface `Factorization` defines the most useful factorization methods. The method `factors()` computes a complete factorization with no further preconditions and returns a `SortedMap`, which maps polynomials to the exponents of the polynomials occurring in the factorization. The method `factorsSquarefree()` factors a squarefree polynomial. It returns a list of polynomials since the exponents will all be 1. Method `factorsRadical()` computes a complete factorization, but returns a list of polynomials, that is, all exponents are removed. Methods `isIrreducible()` and `isReducible()` test the respective properties for a polynomial. `isFactorization()` tests if a given map is actually a factorization for a given polynomial.

The abstract class `FactorAbstract` implements all of the methods specified in the interface. Only `baseFactorSquarefree()` for the factorization of a

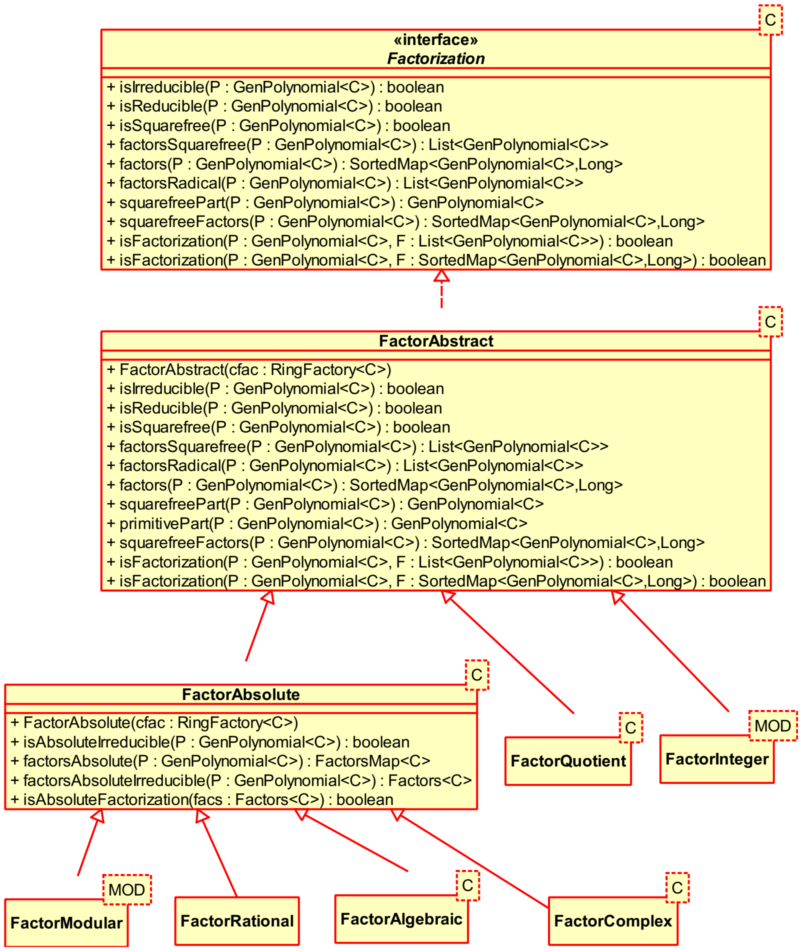

C
«interface»
Factorization
+ isIrreducible(P : GenPolynomial<C>) : boolean
+ isReducible(P : GenPolynomial<C>) : boolean
+ isSquarefree(P : GenPolynomial<C>) : boolean
+ factorsSquarefree(P : GenPolynomial<C>) : List<GenPolynomial<C>>
+ factors(P : GenPolynomial<C>) : SortedMap<GenPolynomial<C>,Long>
+ factorsRadical(P : GenPolynomial<C>) : List<GenPolynomial<C>>
+ squarefreePart(P : GenPolynomial<C>) : GenPolynomial<C>
+ squarefreeFactors(P : GenPolynomial<C>) : SortedMap<GenPolynomial<C>,Long>
+ isFactorization(P : GenPolynomial<C>, F : List<GenPolynomial<C>>) : boolean
+ isFactorization(P : GenPolynomial<C>, F : SortedMap<GenPolynomial<C>,Long>) : boolean
C
FactorAbstract
+ FactorAbstract(cfac : RingFactory<C>)
+ isIrreducible(P : GenPolynomial<C>) : boolean
+ isReducible(P : GenPolynomial<C>) : boolean
+ isSquarefree(P : GenPolynomial<C>) : boolean
+ factorsSquarefree(P : GenPolynomial<C>) : List<GenPolynomial<C>>
+ factorsRadical(P : GenPolynomial<C>) : List<GenPolynomial<C>>
+ factors(P : GenPolynomial<C>) : SortedMap<GenPolynomial<C>,Long>
+ squarefreePart(P : GenPolynomial<C>) : GenPolynomial<C>
+ primitivePart(P : GenPolynomial<C>) : GenPolynomial<C>
+ squarefreeFactors(P : GenPolynomial<C>) : SortedMap<GenPolynomial<C>,Long>
+ isFactorization(P : GenPolynomial<C>, F : List<GenPolynomial<C>>) : boolean
+ isFactorization(P : GenPolynomial<C>, F : SortedMap<GenPolynomial<C>,Long>) : boolean
C
FactorAbsolute
+ FactorAbsolute(cfac : RingFactory<C>)
+ isAbsoluteIrreducible(P : GenPolynomial<C>) : boolean
+ factorsAbsolute(P : GenPolynomial<C>) : FactorsMap<C>
+ factorsAbsoluteIrreducible(P : GenPolynomial<C>) : Factors<C>
+ isAbsoluteFactorization(facs : Factors<C>) : boolean
C
FactorQuotient
MOD
FactorInteger
MOD
FactorModular
FactorRational
C
FactorAlgebraic
C
FactorComplex

Fig. 3. Polynomial factorization classes

square-free univariate polynomial is declared abstract and must be implemented for each coefficient ring. For multivariate polynomials Kronecker's algorithm is used to reduce this case to a univariate problem and to reassemble multivariate factors from univariate ones. This algorithm is not particularly fast and will be accompanied by a multivariate Hensel algorithm in the future.

The absolute factorization of a polynomial, that is, a factorization over an algebraically closed coefficient field, is implemented in class `FactorAbsolute`. The main method is `baseFactorsAbsoluteIrreducible` for the factorization of a polynomial which is irreducible over the given coefficient field. This method constructs a field extension of the ground field from the given irreducible polynomial. The given polynomial is then factored in this algebraic field extension with the algorithms from class `FactorAlgebraic`. Then there are classes which implement factorization for particular coefficient rings, for example modular integers, integers and rational numbers as well as generic algorithms for algebraic numbers and quotients of polynomials.

We come to a comparison with the "categorical view of factorization" in Scratchpad and Axiom. There factorization is attached to the generic univariate polynomial 'class', named `SparseUnivariatePolynomial`. The 'method' called `factorPolynomial()` computes the factorization of a given polynomial. 'Categorical' then means, that it will compute a factorization of multivariate polynomials over any coefficient ring which has itself a constructive implementation of factorization of univariable polynomials over itself.

The building blocks for factorization in JAS, the computation of greatest common divisors and the squarefree decomposition are fully generic in this sense (except the special implementations for integers and integers modulo primes). They only require coefficient rings which themselves have constructive greatest common divisors (for univariate polynomials over itself). Also for squarefree decomposition all cases for coefficient fields of characteristic zero, respectively finite and infinite fields of characteristic $p > 0$ are implemented generically.

Multivariate polynomial factorization reduction to univariate polynomial factorization is fully generic by the Kronecker substitution algorithm. Univariate polynomial factorization for specific coefficient rings, namely `BigInteger`, `ModInteger` and `BigRational` is obviously not generic. However, `FactorAlgebraic` and `FactorQuotient` is generic for coefficients rings which allow constructive univariate polynomial factorization. The selection of appropriate factorization algorithms (as well as squarefree decomposition and gcd algorithms) is provided in class `FactorFactory` (respectively `SquarefreeFactory` and `GCDFactory`) by method `getImplementation(RingFactory r)`. It takes a factory for the coefficient ring `r` as parameter and returns a suitable implementation (if one is available or exists [30]). The returned implementation is of type `FactorAbstract<C>` which must at least contain a method to factor univariate polynomials over the ring `r`. For example we can factor polynomials with coefficients from the ring $\mathbb{Q}(\sqrt{2})(x)(\sqrt{x})$. In summary we have shown, that very general approaches like the categorical approach of Scratchpad can be implemented in a generic, type-safe way in Java.

4 Future Work

With the described software stack, computer algebra software can make use of many modern computing and presentation environments. For example, using existing rich client platforms like Eclipse by MathEclipse [10]. Or for using web-service and Cloud computing platforms, like Google App Engine by Symja [53]. This approach is moreover interesting as it has a parser for the Mathematica programming language. A further example is computer algebra on portable devices: we have a version of the JSCL library compiled for the Palm platform [26], and we plan to use the scripting ability of Android to make computer algebra available on this system [54].

The JVM infrastructure enables moreover a new lower level of interaction between computer algebra systems. It complements the most general high level interoperation between computer algebra systems using OpenMath [37] and the intermediate level interoperation using Python like the Sage project [13]. OpenMath builds on very general interfaces defining algebraic and symbolic expressions whereas Sage takes an agnostic approach and completely ignores common interfaces for the integrated computer algebra systems.

Examples for a tight interoperation on the JVM are the linear algebra library of Apache Commons Math [55] or the linear algebra library JLinAlg [56]. Using adaptors for ring elements it is possible to use generic implementations of linear algebra algorithms with JAS algebraic objects and vice-versa. The various adaptor classes between generic computer algebra libraries could be avoided if we could define (and agree on) a common interface for ring elements and implement it in each system.

A more loose integration is based on the Java scripting framework specified in JSR 223 [57] and was outlined in jscl-meditor to combine several scripting engines accessed from a common mathematical editor [26]. Further interoperation of computer algebra systems is investigated with a Maxima port to ABCL common lisp on the JVM [58]. There is also a Reduce on the JVM [59,21] which could be accessed via JSR 223 (with some effort in code rearrangement). Combining OpenMath and Java is studied in Popcorn and Wupsi [36,39,40,37,38].

5 Conclusions

We have shown how computer algebra software can be designed and implemented leveraging 30 years of advances in computer science. Our approach concentrates on mathematical aspects by re-using software components developed by other projects. Namely, the Java programming language together with the JVM runtime system and interactive scripting languages, for example Python. The JVM infrastructure opens new ways of interoperability of computer algebra systems on Java byte-code level. This infrastructure gives also new opportunities to provide computer algebra software on new computing devices, software as a service, distributed or cloud computing. Our object oriented approach with the Java and Scala programming languages makes it possible to implement non-trivial algebraic structures in a type-safe way which can be stacked and plugged together in unprecedented ways.

Acknowledgments

We thank Thomas Becker for discussions on the implementation of a polynomial template library. JAS itself has improved by requirements from Axel Kramer and Georg Thimm. We thank our colleagues W. K. Seiler, Dongming Wang and Thomas Sturm for various discussions and for encouraging our work. Thanks also to the referees for the suggestions to improve the paper.

References

1. Frink, A., Bauer, C., Kreckel, R.: Introduction to the GiNaC framework for symbolic computation within the C++ programming language. J. Symb. Comput. (2002)
2. Stansifer, R., Baumgartner, G.: A Proposal to Study Type Systems for Computer Algebra. Technical Report 90-07, Johannes Kepler University, Linz, Austria (1990)
3. Jenks, R., Sutor, R. (eds.): Axiom The Scientific Computation System. Springer, Heidelberg (1992)
4. Calmet, J., Seiler, W.M.: Computer algebra and field theories. Mathematics and Computers in Simulation 45, 33–37 (1998)
5. Watt, S.M.: On the future of Computer Algebra Systems at the threshold of 2010. In: Proceedings ASCM-MACIS 2009, pp. 422–430. Kyushu University, Fukuoka (2009)
6. Wolfram, S.: WolframAlpha. Technical report (2009), `http://www.wolframalpha.com/` (accessed January 2010)
7. Certik, O.: SymPy Python library for symbolic mathematics. Technical report (since 2006), `http://code.google.com/p/sympy/` (accessed November 2009)
8. GWT Developers: Google Web Toolkit consists of a Java - to - JavaScript compiler, user interface API, and related tools. Technical report (2008), `http://code.google.com/webtoolkit/` (accessed November 2009)
9. Eclipse Developers: Eclipse rich client platform (RCP). Technical report (2008), `http://www.eclipse.org/` (accessed November 2009)
10. Kramer, A.C.: MathEclipse is usable as an online Java computer algebra system or Eclipse plugin. Technical report (2009, since 2002), `http://www.matheclipse.org/` (accessed November 2009)
11. Greuel, G., Pfister, G., Schönemann, H.: Singular - A Computer Algebra System for Polynomial Computations. In: Computer Algebra Handbook, pp. 445–450. Springer, Heidelberg (2003)
12. Greuel, G., Pfister, G.: A Singular introduction to commutative algebra. Springer, Heidelberg (2007)
13. Stein, W.: SAGE Mathematics Software (Version 2.7). The SAGE Group (since 2005), `http://www.sagemath.org` (accessed November 2009)
14. AppEngine Developers: Google App Engine enables you to build and host web apps on the same systems that power Google applications. Technical report (2010), `http://code.google.com/appengine` (accessed June 2010)
15. Android Developers: Android is a software stack for mobile devices including an operating system, middleware and key applications. Technical report (2008), `http://code.google.com/android/` (accessed November 2009)
16. Abdali, S.K., Cherry, G.W., Soiffer, N.: An object-oriented approach to algebra system design. In: Char, B.W. (ed.) Proc. SYMSAC 1986, pp. 24–30. ACM Press, New York (1986)
17. Zippel, R.: Weyl computer algebra substrate. In: Miola, A. (ed.) DISCO 1993. LNCS, vol. 722, pp. 303–318. Springer, Heidelberg (1993)
18. Parisse, B.: Giac/Xcas, a free computer algebra system. Technical report, University of Grenoble (2008)
19. Bernardin, L., Char, B., Kaltofen, E.: Symbolic computation in Java: an appraisement. In: Dooley, S. (ed.) Proc. ISSAC 1999, pp. 237–244. ACM Press, New York (1999)
20. Bernardin, L.: A Java framework for massively distributed symbolic computing. SIGSAM Bull. 33(3), 20–21 (1999)

21. Norman, A.C.: Further evaluation of Java for symbolic computation. In: ISSAC 2000: Proc. International Symposium on Symbolic and Algebraic Computation 2000, pp. 258–265. ACM, New York (2000)
22. Niculescu, V.: A design proposal for an object oriented algebraic library. Technical report, Studia Universitatis "Babes-Bolyai" (2003)
23. Niculescu, V.: OOLACA: an object oriented library for abstract and computational algebra. In: OOPSLA Companion, pp. 160–161. ACM, New York (2004)
24. Whelan, C., Duffy, A., Burnett, A., Dowling, T.: A Java API for polynomial arithmetic. In: Proc. PPPJ 2003, pp. 139–144. Computer Science Press, New York (2003)
25. Platzer, A.: The Orbital library. Technical report, University of Karlsruhe (2005), http://www.functologic.com/
26. Jolly, R.: jscl-meditor - Java symbolic computing library and mathematical editor. Technical report (since 2003), http://jscl-meditor.sourceforge.net/ (accessed November 2009)
27. Kredel, H.: A systems perspective on A3L. In: Proc. A3L: Algorithmic Algebra and Logic 2005, pp. 141–146. University of Passau (April 2005)
28. Focalize Developers: Focalize is a software distribution for program certification. Technical report (2005-2010), http://focalize.inria.fr/ (accessed June 2010)
29. Davenport, H.J., Trager, B.M.: Scratchpad's view of algebra I: Basic commutative algebra. In: Miola, A. (ed.) DISCO 1990. LNCS, vol. 429, pp. 40–54. Springer, Heidelberg (1990)
30. Davenport, H.J., Gianni, P., Trager, B.M.: Scratchpad's view of algebra II: A categorical view of factorization. In: Proc. ISSAC 1991, Bonn, pp. 32–38 (1991)
31. Davenport, H.J.: Abstract data types in Computer Algebra. In: Nielsen, M., Rovan, B. (eds.) MFCS 2000. LNCS, vol. 1893, pp. 21–35. Springer, Heidelberg (2000)
32. Bronstein, M.: Sigma$^{\mathrm{it}}$ - a strongly-typed embeddable computer algebra library. In: Limongelli, C., Calmet, J. (eds.) DISCO 1996. LNCS, vol. 1128, pp. 22–33. Springer, Heidelberg (1996)
33. Musser, D., Schupp, S., Loos, R.: Requirement oriented programming - concepts, implications and algorithms. In: Jazayeri, M., Musser, D.R., Loos, R.G.K. (eds.) Dagstuhl Seminar 1998. LNCS, vol. 1766, pp. 12–24. Springer, Heidelberg (2000)
34. Schupp, S., Loos, R.: SuchThat - generic programming works. In: Jazayeri, M., Musser, D.R., Loos, R.G.K. (eds.) Dagstuhl Seminar 1998. LNCS, vol. 1766, pp. 133–145. Springer, Heidelberg (2000)
35. Dragan, L., Watt, S.: Performance Analysis of Generics in Scientific Computing. In: Proceedings of Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, pp. 90–100. IEEE Computer Society, Los Alamitos (2005)
36. Freundt, S., Horn, P., Konovalov, A., Linton, S., Roozemond, D.: Symbolic computation software composability protocol (SCSCP) specification, version 1.3. Technical report, SCIEnce Consortium (2009)
37. OpenMath Consortium: OpenMath, version 2.0. Technical report, OpenMath Consortium (2004), http://www.openmath.org/standard/om20-2004-06-30/omstd20html-0.xml (accessed January 2010)
38. SCIEnce Consortium: Symbolic computation infrastructure for Europe. Technical report, SCIEnce Consortium (2009)
39. Horn, P., Roozemond, D.: The Popcorn OpenMath representation, version 1.0. Technical report, SCIEnce EU Project (2009)

40. Horn, P., Roozemond, D.: WUPSI universal Popcorn SCSCP interface, version 1.0. Technical report, SCIEnce EU Project (2009)
41. Jolly, R., Kredel, H.: How to turn a scripting language into a domain specific language for computer algebra. Technical report (2008), http://arXiv.org/abs/0811.1061
42. Jolly, R., Kredel, H.: Symbolic script programming for Java. Technical report (2009), http://arXiv.org/abs/0906.2315
43. Kredel, H.: On the Design of a Java Computer Algebra System. In: Proc. PPPJ 2006, pp. 143–152. University of Mannheim (2006)
44. Kredel, H.: Evaluation of a Java Computer Algebra System. In: Kapur, D. (ed.) ASCM 2007. LNCS (LNAI), vol. 5081, pp. 59–62. Springer, Heidelberg (2008)
45. Kredel, H.: Evaluation of a Java computer algebra system. In: Kapur, D. (ed.) ASCM 2007. LNCS (LNAI), vol. 5081, pp. 121–138. Springer, Heidelberg (2008)
46. Kredel, H.: Multivariate greatest common divisors in the Java Computer Algebra System. In: Proc. Automated Deduction in Geometry (ADG), pp. 41–61. East China Normal University, Shanghai (2008)
47. Kredel, H.: On a Java Computer Algebra System, its performance and applications. Science of Computer Programming 70(2-3), 185–207 (2008)
48. Kredel, H.: Comprehensive Gröbner bases in a Java Computer Algebra System. In: Proceedings ASCM 2009, pp. 77–90. Kyushu University, Fukuoka (2009)
49. Kredel, H.: Distributed parallel Gröbner bases computation. In: Proc. Workshop on Engineering Complex Distributed Systems at CISIS 2009. University of Fukuoka, Japan (2009), CD–ROM
50. Kredel, H.: Distributed hybrid Gröbner bases computation. In: Proc. Workshop on Engineering Complex Distributed Systems at CISIS 2010. University of Krakow, Poland (2010), CD–ROM
51. Kredel, H.: The Java algebra system (JAS). Technical report (since 2000), http://krum.rz.uni-mannheim.de/jas/
52. Geddes, K.O., Czapor, S.R., Labahn, G.: Algorithms for Computer Algebra. Kluwer Academic Publishers, Dordrecht (1993)
53. Kramer, A.C.: Symja a symbolic math system written in Java based on the MathEclipse libraries. Technical report (since 2009), http://code.google.com/p/symja/ (accessed January 2010)
54. Android Scripting Developers: Android Scripting brings scripting languages to android. Technical report (2009), http://code.google.com/p/android-scripting/ (accessed June 2010)
55. Apache Software Foundation: Commons-Math: The Jakarta mathematics library. Technical report (2003-2010), http://commons.apache.org/ (accessed November 2009)
56. Keilhauer, A., Levy, S.D., Lochbihler, A., Ökmen, S., Thimm, G.L., Würzebesser, C.: JLinAlg: a Java-library for linear algebra without rounding errors. Technical report (2003-2010), http://jlinalg.sourceforge.net/ (accessed January 2010)
57. Sun Microsystems, Inc.: JSR 223: Scripting for the Java platform. Technical report (2003-2006), http://scripting.dev.java.net/ (accessed November 2009)
58. ABCL Developers: Armed bear common lisp (ABCL) - common lisp on the JVM. Technical report (2003-2010), http://common-lisp.net/project/armedbear/ (accessed January 2010)
59. Reduce Developers: REDUCE interactive system for general algebraic computations. Technical report (1968-2010), http://www.reduce-algebra.com/ (accessed January 2010)

Implementing Data Parallel Rational Multiple-Residue Arithmetic in Eden[⋆]

Oleg Lobachev and Rita Loogen

Philipps–Universität Marburg, Fachbereich Mathematik und Informatik
Hans–Meerwein–Straße, D–35032 Marburg, Germany
`{lobachev,loogen}@informatik.uni-marburg.de`

Abstract. Residue systems present a well-known way to reduce computation cost for symbolic computation. However most residue systems are implemented for integers or polynomials. This work combines two known results in a novel manner. Firstly, it lifts an integral residue system to fractions. Secondly, it generalises a single-residue system to a multiple-residue one. Combined, a rational multi-residue system emerges. Due to the independent manner of single "parts" of the system, this work enables progress in parallel computing. We present a complete implementation of the arithmetic in the parallel `Haskell` extension `Eden`. The parallelisation utilises algorithmic skeletons. A non-trivial example computation is also supplied.

Keywords: residue system, rational reconstruction, EEA, CRT, homomorphism, parallelisation, functional programming, parallel functional software, implementation report.

1 Introduction

A common approach to reduce computation complexity of symbolic computations, i.e. intermediate expression swell, is a residue arithmetic. We regard residues of integers in this paper. A residue system w.r.t. some prime m is the field $\mathbb{Z}/\langle m\rangle$. The addition and multiplication are the usual operations in $\mathbb{Z}$, combined with an integer division by m. The actual result is the residue of the division. The focus of this paper lies on systems with a) using multiple residues at the same time, b) capable of representing fractions. If we have an *a priori* upper bound on the result of our computation, then we can perform it in a certain residue class with the result remaining exact. The benefit is reduced computation time, especially for intermediate expressions, which might be significantly larger than the bound [13]. Using the Chinese Residue Theorem we can split a *single* large residue class into multiple smaller residue classes. The latter can be designed to fit into a machine word, making the single operation in a small residue class a constant time one. Thus we can obtain arbitrary precision by increasing the numbers of small residue classes.

[⋆] Supported by DFG grant LO 630-3/1.

The approach mentioned above is traditionally implemented for *integer* arithmetic. However, it is possible to represent certain subsets of *rational* numbers as integers in a residue class—and to recover the rational numbers from integers. This property holds for a bound on the input rational numbers and output residue class [5].

Eden. This paper presents a parallel `Haskell` programmer's approach to a *rational multiple-residue arithmetic*. We develop it in the broader context of the SPICA project[1], an implementation of selected computer algebra algorithms using novel parallelisation techniques, i.e. algorithmic skeletons. Such skeletons implement common patterns of parallel computation like process farms, divide-&-conquer schemes, etc. The source code is written in `Haskell` [10] with GHC extensions. The parallelism constructs reside in a controlled subset of the code base. It is written in the parallel `Haskell` extension called `Eden` [9]. The latter incorporates explicit process creation and implicit communication. An *algorithmic skeleton* library is available for `Eden` [8]. Contrary to the typical imperative approach, the skeletons are implemented in `Eden` itself, as they are just higher-order functions. The skeleton library contains, for instance, a parallel process `farm`, implemented as a statically load-balanced parallel `map`. This function applies its first argument which is a function to each element of its second argument which is a list. The farm skeleton divides the input list into packages of almost equal size which are processed in parallel. Using only skeleton calls for parallelism, we can refrain from using parallel primitives in `Eden` programs. `Eden` is implemented as a distributed memory language, but it performs also well on multicore SMP machines. `Haskell` and thus `Eden` is a statically typed language with polymorphism. Each language object has a type, obtained with Hindley-Milner type inference. We denote the type with `object :: type`.

Plan of the Paper. The next section presents the rational-to-integer and integer-to-rational mappings. Section 3 presents in short an integer multiple-residue arithmetic. Section 4 is the actual focus of the paper. Subsections 4.1 and 4.2 present the rational multiple-residue arithmetic, whereas Subsection 4.3 describes our approach to parallelism. Section 5 presents an example implementation using the arithmetic. Section 6 presents related work. Section 7 concludes.

2 From Fractions to Integers and Back

Definition 1 (Residues and division). *We denote integer division with* $a = cm + r$ *as* $c = a \operatorname{div} m$ *and* $r = a \bmod m$*. The latter forms a* residue class *for given* m*. We define* $(\mathbb{Z}_m, \oplus, \odot) := (\mathbb{Z}, +, \cdot)/\langle m\rangle$*. If* m *prime,* $(\mathbb{Z}_m, \oplus, \odot)$ *is a field with* $x \circledcirc y = (x \circ y) \bmod m$ *for* $x, y \in \mathbb{Z}_m$ *and* $\circ \in \{+, -, \cdot, /\}$*. An element of* $\mathbb{Z}_m = \{0, \ldots, m-1\}$*, denoted with* $|a|_m$*, is* $a \bmod m$*. With a small abuse of notation, we write in further simply* $\mathbb{Z}_m$ *for* $(\mathbb{Z}_m, \oplus, \odot)$*. Further, if in the division above* $r = 0$*, then we write* $m \mid a$*, else* $m \nmid a$*. A residue class modulo*

[1] `http://www.mathematik.uni-marburg.de/~lobachev/`

multiple *residues* $\beta = [m_1, m_2, \dots, m_n]$ *is defined as* $\mathbb{Z}_\beta := \mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_n}$. *The corresponding arithmetic operations will be defined later.*

The well-known notion of an *extended euclidean algorithm* (EEA) in a matrix-vector form can be represented as follows.

Algorithm 1 (Standard EEA)
Input: seed matrix $\begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \end{pmatrix}$.

```
1. If $a_2 = 0$, return $[a_1, b_1]$.
2. Let $t = a_1 \operatorname{div} a_2$. Set pairwise $(a_1, a_2) \leftarrow (a_2, a_1 - ta_2)$ and $(b_1, b_2) \leftarrow (b_2, b_1 - tb_2)$. Go to step 1.
```

Output: $[a_1, b_1]$.

We implement Algorithm 1 in Figure 7 in the appendix, see top of the figure. The 2×2 matrix is represented by a nested pair of pairs. The `Integral` type class is a `Haskell` notion for ring $\mathbb{Z}$, which abstracts from the integer representation.

Definition 2. *We define the residue-based representation of the rationals* $|a/b|_m$ *as elements of* $\hat{\mathbb{Q}}_m$ *per [5]. The elements of* $\hat{\mathbb{Q}}_m$ *are integers in the* m*-modular residue arithmetic. The notion of* $|a/b|_m$ *stands for an integer modulo* m*, congruent to* $|a|_m \odot |b^{-1}|_m$. *Here* $\odot$ *denotes the multiplication modulo* m.

We can compute $|a/b|_m$ efficiently, using EEA.

Algorithm 2 (Rational-to-integer mapping)
Input: A fraction a/b*, an integer* m *with* $m \nmid a$, $m \nmid b$.
Start Algorithm 1 with input matrix

$$\begin{pmatrix} m & 0 \\ b & a \end{pmatrix}.$$

Return the second element of the output vector.
Output: an integer, representing $|a/b|_m$.

Algorithm 2 returns the desired *recoverable* result if a *bound* on m and a/b holds. It is rigorously discussed [4,14,15,6,5]. We summarise.

Definition 3 (Farey fractions). *All vulgar fractions* a/b *satisfying* $|a| \leq N$, $|b| \leq N$ *are called* Farey fractions *of order* N.

Proposition 4. *If*

$$N \leq \sqrt{\frac{m}{2}} \qquad (1)$$

holds, it is possible to recover the original Farey fraction a/b *of order* N *from integer* $|a/b|_m$.

```
data Mod a = Z a a
makeZ :: Integral a ⇒ a → a → Mod a
lift2z :: Integral a ⇒ (a → a → a) → Mod a → Mod a → Mod a
lift2z f (Z a p) (Z b q)
  | p ≠ q = error "Different residue classes!"
  | otherwise = makeZ (f a b) p

-- P.+ denotes (+) instances for Integral type class
-- from the Prelude. It corresponds to the ring ℤ.
instance (Integral a) ⇒ Num (Mod a) where
    (+) = lift2z (P.+)
    (−) = lift2z (P.-)
    (∗) = lift2z (P.∗)
instance (Integral a) ⇒ Fractional (Mod a) where
    (/) (Z a p) (Z b q) = -- use EEA
```

Fig. 1. Required function types for single-residue arithmetic in `Eden`

Algorithm 3 (Integer to Farey fraction)
Input: an integer $x = |a/b|_m$, m.

1. *Compute N from m per (1).*
2. *Start Algorithm 1 with seed matrix*

$$\begin{pmatrix} m & 0 \\ x & 1 \end{pmatrix}.$$

3. *In each step of the algorithm check, whether $|a_1|$ and $|b_1|$ are both smaller than N. If so, return the fraction b_1/a_1 (sic). If Algorithm 1 terminates without producing such a pair of numbers, fail.*

Output: either a Farey fraction a/b or a failure.

The correctness of Algorithm 3, the uniqueness of the fraction b_1/a_1, and the criteria for the input of the algorithm, needed to succeed, are proved in [5]. The first proof known to us is in [15]. We name the mapping "rational reconstruction" per [13]. We show `Eden` implementations of Algorithms 2 and 3 on Figure 7. Further, we need a way to refer to a single residue arithmetic in $\mathbb{Z}/\langle m\rangle = \mathbb{Z}_m$. The details are well-known, we present source code signatures in Figure 1.

3 An Integer-Based Multiple-Residue Arithmetic

Let us consider a *multiple*-residue system $\mathbb{Z}_\beta$ with more than one residue. Hence, β is a vector. For the sake of simplicity we consider elements of β to be prime numbers. Then for $\beta = [m_1, m_2, \dots, m_n]$ single residue classes are $\mathbb{Z}_{m_1}, \dots, \mathbb{Z}_{m_n}$. With $M = m_1 \cdots m_n$, it holds that

$$\mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_n} = \mathbb{Z}_\beta \cong \mathbb{Z}_M. \qquad (2)$$

The equation (2), read from left to right, is widely known as Chinese Residue Theorem, which we abbreviate to CRT. There are many different proofs of the

```
type IMods a = [Mod a]
makeIZ′ :: (Integral a, Integral b) ⇒ a → [a] → IMods b
makeIZ′ value primes = map (makeZ value) primes
instance (Integral a) ⇒ Num (IMods a) where
    (+) = zipWith (+)
    -- and so on...
```

Fig. 2. Implementing the multiple-residue integer arithmetic

CRT; some of the constructive ones allow the algorithmic construction of the "large" residue. We call such proofs *implementations* of CRT, the other name in the literature is "Chinese Residue Algorithm", *cf.* [13].

Further (2) facilitates a background for forth and backwards mappings between $\mathbb{Z}_\beta$ and $\mathbb{Z}_M$ as well as for defining the arithmetic. We will present this known result with a notion from functional programming.

Definition 5 (Map function). *For all functions operating on single elements:* `f :: a → b`*, we define a function* `map`*, which takes as its arguments such* `f` *and a* collection *of type* `[a]` *of elements of type* `a`*. The function* `map` *applies* `f` *to each element of its input collection and combines the results of each such application to its output collection of type* `[b]`*. Hence,* `map` *has the type* `(a → b) → [a] → [b]` *and a partial application* `map f` *has the type* `[a] → [b]`*. So we write*

```
map :: (a → b) → [a] → [b]
map f xs = [ f x | x ∈ xs ]
```

Corollary 6 (ZipWith function). *We define a binary version of* `map`*.*

```
zipWith :: (a → b → c) → [a] → [b] → [c]
zipWith f xs ys = [ f x y | x ∈ xs | y ∈ ys ]
```

Now we can define all four integral multiple-residue arithmetic operation on $\mathbb{Z}_\beta$ as a kind of `map` of their single-residue counterparts. Because `map` applies `f` to each single element in an independent manner, such definition of a multiple-residue arithmetic underlines its strength for vectorisation. All of the computation within a single "residue element" can be done independent from other residue elements. This will be the basis for the parallel implementation in Subsection 4.3. The implementation of the arithmetic falls back to `zipWith`—a variant of `map` for binary functions. Hence, the type for the multiple-residue system is just a list of single-residues. See Figure 2 for details. Now we can sketch the following.

Algorithm 4 (To integral multiple-residue)
Input: vector of primes β, integer x with no common factors with elements of β.
Compute $|x|_{m_i}$ for all elements m_i of β.
Output: $|x|_\beta$

Algorithm 5 (From integral multiple-residue)
Input: $|x|_\beta$.
Compute $|x|_M$ with an implementation of CRT.
Output: $|x|_M$.

We omit the implementation here for the sake of brevity.

```
data FSingleMod a = FM (Mod a) a
type FMods a = [FSingleMod a]

nFromM, mFromN :: Integral i ⇒ i → i
-- convert M to n per (1)
makeFZ :: Integral i ⇒ Fraction i → FMods i
-- forth mapping, see Figure 4
restoreFZ :: Integral i ⇒ FMods i → Maybe (Fraction i)
-- backwards mapping, see Figure 5
```

Fig. 3. Basic outline of rational multiple-residue implementation in `Eden`

4 A Rational Multiple-Residue System

4.1 The Mappings

Definition 7 (Elements). *We define an element of a rational multiple-residue system $\mathbb{W}_\beta$ as follows. Let n be the length of prime list β. Then such element is a list of pairs*

$$[(u_i, v_i) : i = 1, \dots, n] \,.$$

Here the components u_i are the residues and v_i are the powers of corresponding elements of β.

The implementation is in Figure 3. The authors of [5] define a similar residue system, we call $\mathbb{M}_\beta$ here. Its elements are similar to those of $\mathbb{W}_\beta$ and the arithmetical operations definitions coincide. The major difference lies in how the forth and backwards mappings are defined. Noteworthy, [4,6] define a yet another residue system, which differs from both $\mathbb{M}_\beta$ and $\mathbb{W}_\beta$ in not separating out the powers of primes v_i from the residues u_i. It is the predecessor of $\mathbb{M}_\beta$.

Given a fraction a/b and $\beta = [m_1, m_2, \dots, m_n]$, satisfying (1) and (3), we can state the following.

Algorithm 6 (Outline of forth mapping)
Input: fraction a/b, residues $\beta = [m_1, \dots, m_n]$.

1. *Extract common factors v_i of all m_i and a/b. Remember $v_1, \dots, v_n$.*
2. *Convert the resulting fraction to an integer modulo $M = m_1 \cdots m_n$ with Algorithm 2.*
3. *Convert the resulting integer to a multiple-residue system modulo β with Algorithm 4. Store the results in a list $[u_1, \dots, u_n]$.*

Output: rational multiple-residue representation of a/b being $[(u_1, v_1), \dots, (u_n, v_n)]$.

The key difference between our approach and $\mathbb{M}_\beta$ is in

$$\frac{a^{(1)}}{b^{(1)}} = \frac{a}{b} m_1^{v_1} \quad \frac{a^{(2)}}{b^{(2)}} = \frac{a}{b} m_2^{v_2} \quad \dots \quad \frac{a^{(n)}}{b^{(n)}} = \frac{a}{b} m_n^{v_n} \,. \tag{3}$$

With these equations, the forth mapping for $\mathbb{W}_\beta$ takes in step 2 the value $a^{(i)}/b^{(i)}$ for i-th residue class in the system. The forth mapping for $\mathbb{M}_\beta$ extracts *all* factors

```
detectPower :: (Integral i, Num n) ⇒ i → i → (n, i)
-- Code omitted. Example: detectPower 40 2 = (3, 5)

convertFraction :: (Integral i) ⇒ i → i → i → Mod i
-- Code omitted. Converts fraction to integer modulo m

extractFactors :: (Integral i, Num n) ⇒ i → [i] → [(n, i)]
extractFactors x ps = map (detectPower x) ps

makeFZ′ :: Integral i ⇒ i → i → [i] → FMods i
makeFZ′ a b ps | gcd a b == 1
  = let (ws, ys) = unzip $ extractFactors a ps
        (qs, zs) = unzip $ extractFactors b ps
        vs = zipWith (−) ws qs -- well-defined
        cs = zipWith3 convertFraction ys zs ps
    in zipWith FM (cs) vs
               | otherwise = -- recursive call

makeFZ :: Integral i ⇒ Fraction i → FMods i
makeFZ = -- a trivial constructor expansion
```

Fig. 4. Forward mapping (Algorithm 7)

from the input fraction for all residue classes. Unfortunately, this leads to an instable addition. Our approach does not have such a problem. We will show an example, underlying the difference of both approaches after the definition of arithmetic operations as Example 12 on page 186.

Algorithm 7 (Forth mapping)
Input: fraction a/b, residues $\beta = [m_1, \ldots, m_n]$.
Execute Algorithm 6 with (3) in Step 2.
Output: rational multiple-residue representation of a/b as an element of $\mathbb{W}_\beta$.

The implementation of this algorithm is on Figure 4. Note that in Algorithm 7 we convert each fraction $a^{(i)}/b^{(i)}$ separately, resulting in up to n calls of Algorithm 2. The backward mapping is defined as follows.

Algorithm 8 (Backward mapping)
Input: $[(u_1, v_1), \ldots, (u_n, v_n)] \in \mathbb{W}_\beta$, $\beta = [m_1, \ldots, m_n]$

1. *Compute $M = m_1 \cdots m_n$ and $N = \sqrt{M/2}$.*
2. *Compute $a'/b' = m_1^{v_1} \cdots m_n^{v_n}$.*
3. *For $i \in \{1, \ldots, n\}$ distort the values of u_i. Let*
$$\hat{u}_i := u_i / \prod_{j \neq i} m_j^{v_j}.$$
4. *Regard $[\hat{u}_1, \ldots, \hat{u}_n]$ an integer multiple-residue value in $\mathbb{Z}_\beta$. Find its representation q in $\mathbb{Z}_M$ with an implementation of CRT (Algorithm 5).*
5. *Find fraction a/b of order N, such that $|a/b|_M = q$ with Algorithm 3. If it succeeds, continue. Else fail.*

Output: aa'/bb' or failure.

```
getM :: Integral i ⇒ FMods i → Integer
-- returns the product of all primes in the system

stripPowers :: Integral i ⇒ FMods i → (i, i, FMods i)
-- set v_i = 0 for all i and compensate

restoreFZ′ :: Integral i ⇒ FMods i → (Maybe (i,i), (i,i))
restoreFZ′ x = let m = getM x
                   n = nFromM m
                   (nom, denom, strips) = stripPowers x
                   z = convertToIntResidues strips
                   r = toIntegral $ restoreIZ′ z
                   e = eeaSearch ((m, r), (0, 1)) n
               in (e, (nom, denom))

restoreFZ :: Integral i ⇒ FMods i → Maybe (Fraction i)
restoreFZ =-- compute in Maybe monad the product of fraction e with nom/denom
```

Fig. 5. The outline of the backwards mapping (Algorithm 8)

Implementation of the latter algorithm is presented on Figure 5. How does the input set of Algorithm 7 look like? This set consists of Farey fractions of corresponding order N *and* their products with powers $v_1, \ldots, v_n$ of $m_1, \ldots, m_n$. For the restricted values of v_i the shape of this set is shortly discussed in [7]. If we do not restrict the values of v_i, then it is infinite. Further questions on the shape of this set are open.

4.2 The Arithmetic

Now we have to define the actual arithmetic on $\mathbb{W}_\beta$. Each operation is defined for a single residue (use `map`! The rational system is still independent in its components). These definitions coincide with ones for $\mathbb{M}_\beta$ from [5], but not with ones from [4]. We begin with the definition of multiplication, since it is the simplest operation in the system.

Definition 8 (Multiplication). *The product of* (u, v) *and* (μ, ν) *modulo* m *is defined as* $(|u\mu|_m, v + \nu)$.

The implementation is straightforward:

```
(FM u1 v1) * (FM u2 v2) = FM (u1*u2) (v1+v2)
```

Definition 9 (Multiplicative Inverse). *The inverse of* (u, v) *is* $(|u|_m^{-1}, -v)$.

Note, it is easy and well-known, how to compute $|u|_m^{-1}$, the multiplicative inverse of u modulo m with EEA, Algorithm 1, for such u and m, that $\gcd(u, m) = 1$. It coincides with computing an integer representation of a fraction $1/u$ with Algorithm 2, a standard approach in residue rings. The sum of (u, v) and (μ, ν) modulo m is $(|u + \mu|_m, v)$ if $v = \nu$ and just (u, v) for $|v| < |\nu|$ with a single exception for sum of something with zero being the non-zero summand, regardless of the power of m. A more formal definition follows.

Definition 10 (Addition). *The sum of* (u, v) *and* (μ, ν) *modulo* m *is defined as follows. Let* $u \oplus \mu = |u + \mu|_m$*. We write in this table* v *for positive values,* $-v$ *for negative and* 0 *for zero.*

$+$	$(0,z)$	(u,v)	$(u,0)$	$(u,-v)$
$(0,\zeta)$	$(0,0)$	(u,v)	$(u,0)$	$(u,-v)$
(μ,ν)	(μ,ν)	A	$(u,0)$	$(u,-v)$
$(\mu,0)$	$(\mu,0)$	$(\mu,0)$	$(u\oplus\mu,0)$	$(u,-v)$
$(\mu,-\nu)$	$(\mu,-\nu)$	$(\mu,-\nu)$	$(\mu,-\nu)$	B

The two subcases are:

$$A = \begin{cases} (u,v) & \text{if } v < \nu \\ (u \oplus \mu, v) & \text{if } v = \nu \\ (\mu,\nu) & \text{if } v > \nu \end{cases} \quad B = \begin{cases} (u,-v) & \text{if } -v < -\nu \\ (u \oplus \mu, v) & \text{if } v = \nu \\ (\mu,\nu) & \text{if } -v > -\nu \end{cases}$$

It is $z, \zeta \in \mathbb{Z}$*. The zero element is not unique because of* $(0, z)$ *with* $z \neq 0$*, but we* norm *it to the standard representation* $(0, 0)$*. The Theorem 14 shows that this causes no problems.*

Definition 11 (Additive Inverse). *The additive inverse of* (u, v) *modulo* m *is* $(|-u|_m, v)$*.*

The actual arithmetic operations on $\mathbb{W}_\beta$ are defined by lifting the above single-element operations with `zipWith` to lists:

```
instance (Integral a) ⇒ Num (FMods a) where
    (+) = zipWith (+)
    (−) = zipWith (−)
    (∗) = zipWith (∗)
instance (Integral a) ⇒ Fractional (FMods a) where
    (/) = zipWith (/)
```

The code for addition, the most complicated operation even for single-element inputs, is presented in Figure 8 in Appendix.

Now, given the arithmetic, we can show that our approach is better than $\mathbb{M}_\beta$ from [5]. Regard an example computation [7].

Example 12 (Counterexample for $\mathbb{M}_\beta$). *Let* $a = 1/21$ *and* $b = 1/3$*. We compute in* $\mathbb{M}_\beta$ *modulo* $\beta = [5, 7, 11, 13]$*. Per (1), all fractions of order 50 are on the safe side. As* $\mathbb{M}_\beta$ *needs to extract all factors of elements of* β *from all elements of the residue system, we obtain representations* $[(2, 0), (5, -1), (4, 0), (9, 0)]$ *for* a *and* $[(2, 0), (5, 0), (4, 0), (9, 0)]$ *for* b*. The sum is* $[(4, 0), (5, -1), (8, 0), (5, 0)]$*, we obtain* $2/21$ *as the result, contrary to the correct result* $8/21$*. The same example with* $\mathbb{W}_\beta$ *of the same scale results in* $[(1, 0), (5, -1), (10, 0), (5, 0)]$ *for* a *and* $[(2, 0), (5, 0), (4, 0), (9, 0)]$ *for* b*. The sum is* $[(3, 0), (5, -1), (3, 0), (1, 0)]$*, yielding the correct result* $8/21$*.*

Theorem 13 (Well-definiteness). *The arithmetic operations in* $\mathbb{W}_\beta$ *produce correct results.*

The proof is in the appendix.

Theorem 14. $\mathbb{W}_\beta$ *is a field.*

The mainly technical proof is given in the appendix.

Theorem 15 (Correctness). *The algorithms 7 and 8 are correct.*

Proof (Idea). Each of the two algorithms is a composition of homomorphic mappings. The mappings defined by Algorithms 2, 3, 4 and 5 are known to be homomorphic. Show that the mapping

$$\frac{a}{b} m^v \to \left(\left| \frac{a}{b} \right|_m , v \right)$$

with arithmetic from Section 4.2 is a homomorphism. Further, this mapping is invertible for Farey fractions a/b. Their order is bound by (1). †

4.3 Parallelism

Multiple-residue arithmetic is known for its data parallelism potential. We compute with different residues in a fully independent manner, without a need for a communication in-between. As our implementation of rational multiple-residue arithmetic conforms to this principle, we can immediately make a step from a (sequential) `Eden` implementation to (parallel) `Eden` code.

Suppose, we have some function `f :: FMods Int` → `FMods Int`. This function could be implemented in `Eden` as `f = map g`, where `g :: FSingleMod Int` → `FSingleMod Int`. It suffices to write `f = farm g` to obtain a parallel `Eden` implementation. The skeleton `farm` implements a parallel `map` behaviour and is part of `Eden`'s skeleton library.

A further advantage is provided by the `Eden` type system. As both `FMods` and `FSingleMod` are instances of the standard `Num` and `Fractional` type classes, we could use the standard arithmetical notation of $+, -, \cdot, /$ in the implementation of the function `g` from above. Even more: the generalised type of `g` is `g :: (Num a, Fractional a)` ⇒ `a` → `a`. This means, that we can use `g` for *any* arithmetic of our choice: be it the standard one, or the one presented above. In terms of computer algebra, one says that `g` is symbolic.

5 Testing the Arithmetic

In order to have a large enough task, we use matrix computations for testing the arithmetic. We choose the *LU* decomposition of matrices as our test problem.

We have implemented distributed determinant computation of permuted, scaled with 1/3 Pascal matrices, using the above approach. As a Pascal matrix is unimodular, the final result is always known. The arithmetic of a right scale always performed correctly in our tests. A visualisation of the parallel program execution with EdenTV [1] is depicted in Figure 6. In the diagram, the horizontal axis indicates the time, the vertical axis shows PEs. The bars show process activity over time. Multiple processes can be placed on one PE. The

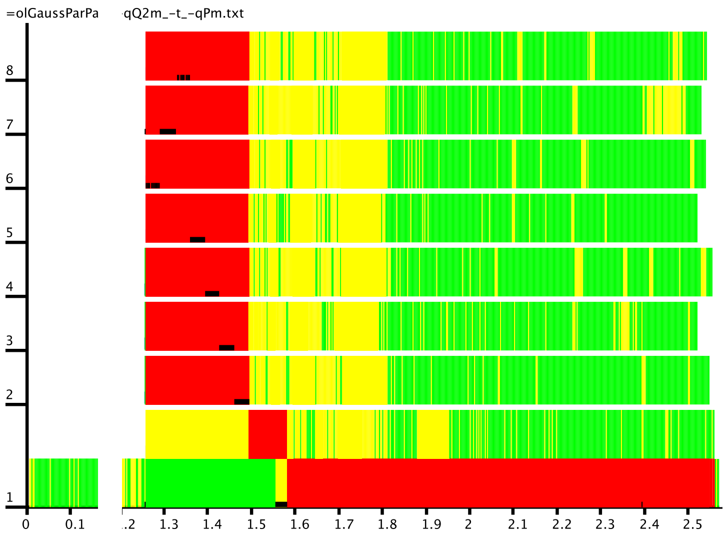


Fig. 6. A fragment of the runtime diagram of the test executable

colours correspond to a traffic light: red ● (dark grey in a black and white version) is blocked, i.e. waiting for input. Yellow ● (light grey) is "runnable", but not running, typical causes are garbage collection and communication in progress. Green ● (grey) stands for running.

The initial delay of 1.3–1.5 second is due to the generation of the input matrix. Probably, some part of this computation is also due to the boilerplate code for parallelisation. Each parallel residue computation, seen from 1.25 second to 2.5–2.6 second perform for about 1.3 seconds. Then, the needed results—only the diagonal of the whole matrix!—are sent back to the PE 1. After a very short post-processing phase the program terminates. The visualised run was performed on a eight-core Intel Xeon machine with a 2.5 GHz CPU and 16 GB RAM. We used no special memory management and fixed the message buffer at 2 MB pro PE. The input matrix size was 100×100, we used 8 residues with primes of size $\approx 5 \cdot 10^4$. The bound on fraction size was $\approx 10^{17}$, the determinant was a Farey fraction of order $\approx 5 \cdot 10^{47}$. We stress that the matrix operations' implementation is a prototype one. The essence of this example is not to make performance records, but to show that the arithmetic produces correct results in practice.

6 Related Work

Any decent system implements residue-based approach for reducing the intermediate expression swell. A lot of work on rational residue systems was done in [5] and preceding papers [4,14,15,6,11]. The earliest approaches to rational residues known to us are [12] and [2]. Our own work on this topic is [7].

7 Conclusions and Future Work

We have presented a rational multiple-residue arithmetic and its parallel implementation in `Eden`. We constructed a test case and provided a visualisation of the program execution. We present in this paper concise source code for almost all operations. The omitted parts are straightforward. Nevertheless, the complete source code package is available from the project homepage[2].

Further direction is the development of a parallel benchmark, based on the presented implementation, coupled with certain test input. Another point for future work would be an attempt for an adaptive computation. It would be interesting to see a `Maple` implementation of our approach. A distributed implementation, based on the remote data concept [3] should improve the speedup values of our implementation.

Acknowledgements. We would like to thank Tomas Sauer for supervising the first author's diploma thesis on a similar topic [7]. This paper's theoretical background relies heavily on this thesis and on an excellent book by Gregory and Krishnamurthy [5]. We are grateful to the anonymous referees for their comments.

References

1. Berthold, J., Loogen, R.: Visualizing Parallel Functional Program Executions: Case Studies with the Eden Trace Viewer. In: ParCo 2007. IOS Press, Amsterdam (2007)
2. Borosh, I., Fraenkel, A.S.: Exact solutions of linear equations with rational coefficients by congruence techniques. Math. Comp. 20(93), 107–112 (1966)
3. Dieterle, M., Horstmeyer, T., Loogen, R.: Skeleton composition using remote data. In: Carro, M., Peña, R. (eds.) PADL 2010. LNCS, vol. 5937, pp. 73–87. Springer, Heidelberg (2010)
4. Gregory, R.T.: Error-free computation with rational numbers. BIT Numerical Mathematics 21(2), 194–202 (1981)
5. Gregory, R.T., Krishnamurthy, E.V.: Methods and Applications of Error–Free Computation. Springer, Heidelberg (1984)
6. Kornerup, P., Gregory, R.T.: Mapping integers and Hensel codes onto Farey fractions. BIT Numerical Mathematics 23(1), 9–20 (1983)
7. Lobachev, O.: Multimodulare Arithmetik, Justus-Liebig-Universität Gießen. Diplomarbeit (March 2007) (in German), `http://www.mathematik.uni-marburg.de/~lobachev/diplom.pdf`
8. Loogen, R., Ortega-Mallén, Y., Peña, R., Priebe, S., Rubio, F.: Parallelism Abstractions in Eden. In: Rabhi, F.A., Gorlatch, S. (eds.) Patterns and Skeletons for Parallel and Distributed Computing. Springer, Heidelberg (2003)
9. Loogen, R., Ortega-Mallén, Y., Peña-Marí, R.: Parallel Functional Programming in Eden. Journal of Functional Programming 15(3), 431–475 (2005)
10. Peyton Jones, S. (ed.): Haskell 98 Language and Libraries: The Revised Report. Cambridge University Press, Cambridge (December 2003)

[2] `http://www.mathematik.uni-marburg.de/~lobachev/code/multimod/`

11. Rao, T.M., Gregory, R.T.: Conversion of Hensel codes to rational numbers. Comp. Math. 10(2), 185–189 (1984)
12. Svoboda, A., Valach, M.: Rational system of residue classes. In: Stroje na Zpraccorani Informaci, Sbornik, Nakl, CSZV, Prague, pp. 9–37 (1957)
13. von zur Gathen, J., Gerhard, J.: Modern Computer Algebra, 2nd edn. Cambridge University Press, Cambridge (2003)
14. Wang, P.S.: A p-adic algorithm for univariate partial fractions. In: Proc. ACM Symposium on Symbolic and Algebraic Computation, pp. 212–217. ACM, New York (1981)
15. Wang, P.S., Guy, M.J.T., Davenport, J.H.: p-adic reconstruction of rational numbers. ACM SIGSAM Bulletin 16(2), 3 (1982)

Appendix

Proof (of Theorem 13). We consider again the element-wise operations.

1. The addition works, despite looking somewhat strange. Let $(|a/b|_m, v)$ and $(|\alpha/\beta|_m, \nu)$ be the summands. The trivial case for summation with zero is clear. The case of $v = \nu$ is also not endearing. All left is the complicated case $v \neq \nu$. Without loss of generality, let $|v| < |\nu|$. Now we have three non-trivial sub-cases for different signs of v and ν, all other cases can be seen as one of those with places swapped. Let us consider one of them: the case "$0 < v < \nu$".
$$\frac{am^v}{b} + \frac{\alpha m^\nu}{\beta} = \frac{am^v}{b} + \frac{\alpha m^{\nu-v} m^v}{\beta} = \frac{am^v}{b} + \frac{\alpha m^v}{\beta} m^{\nu-v}.$$
If we extract all we can, namely m^v, the summand with further factor of m turns into zero modulo m, we have exactly $(|a/b|_m, v)$ remaining. All other cases are analogue.
2. The additive inverse is correct. Changing the sign changes not the factors, thus no change at v. The addition of (u, v) and $(|-u|_m, v)$ returns $(0, v)$, which is zero.
3. The multiplication is straight-forward. It follows
$$\frac{a}{b} m^v \cdot \frac{\alpha}{\beta} m^\nu = \frac{a\alpha}{b\beta} m^{v+\nu},$$
which is exactly what we see.
4. The multiplicative inverse is also correct:
$$(u, v) \cdot (|u|_m^{-1}, -v) = (|u \cdot u^{-1}|_m, v - v) = (1, 0).$$

□

Proof (Proof of Theorem 14). We show the field axioms. We refrain to a single element of the list $[(u_1, v_1), \ldots, (u_n, v_n)] \in \mathbb{W}_\beta$. We call it (u, v) without the indices. We think about u as of an element of the residue ring $\mathbb{Z}_m$ with prime m. If needed, we index v with u, like: (u, v_u). We begin with commutativity and save the associativity of the addition at the very end. Let $(a, v_a), (b, v_b), (c, v_c)$ be in the same projection of $\mathbb{W}_\beta$ to the single element modulo m, so-to-say, let $(a, v_a), (b, v_b), (c, v_c)$ be elements of $\mathbb{W}_m$.

– Addition is commutative. Regard $(a, v_a) + (b, v_b)$. If $v_a = v_b$, then the result follows from commutativity of $\mathbb{Z}_m$. Else assume w.l.o.g. $v_a < v_b$. Then the result of the addition is (a, v_a). As the addition table is symmetric, $(b, v_b) + (a, v_a)$ produces the same result.
– The neutral element is $(0, 0)$. The non-unique representation $(a, v)+(-a, v) = (0, v)$ can be normalised with the convention that any $(0, v)$ should be normed to $(0, 0)$.[3] In further we write $(0, 0)$ and say "zero" for all representations of the neutral element. The normed zero is unique.
– The existence of an additive inverse follows from the same fact in $\mathbb{Z}_m$. The uniqueness: assume (b, v_b) and (c, v_c) are two distinct inverses of (a, v_a), then $(b, v_b) = (b, v_b) + \underbrace{(a, v_a) + (c, v_c)}_{=(0,v')} = \underbrace{(b, v_b) + (a, v_a)}_{=(0,v'')} + (c, v_c) = (c, v_c)$. Both v' and v'' can be normed to $0 \in \mathbb{Z}$ per above.
Assuming there exists an inverse (b, v_b) of $(a, v_a) \neq (0, 0)$ with $v_b \neq v_a$. Assume w.l.o.g. that $v_b < v_a$. Then $(a, v_a) + (b, v_b) = (b, v_b)$, hence (a, v_a) is the neutral element. This is a contradiction.
– Multiplication is associative: $((a, v_a)(b, v_b))(c, v_c) = (|ab|_m, v_a + v_b)(c, v_c) = (|abc|_m, v_a + v_b + v_c) = (a, v_a)(|bc|_m, v_b + v_c) = (a, v_a)((b, v_b)(c, v_c))$.
– Multiplication is commutative because the multiplication in $\mathbb{Z}_m$ and addition in $\mathbb{Z}$ are commutative.
– The unity is $(1, 0)$. It is unique.
– The existence of the multiplicative inverse follows from the fact that $\mathbb{Z}_m$ is a field for prime m and that $\mathbb{Z}$ is a ring. By the way $|a/b|_m^{-1} = |b/a|_m$, we have the same limit on m. The uniqueness is a consequence of the fact that inverses in $\mathbb{Z}_m$ and $\mathbb{Z}$ are unique.
– Left distributive law: Show $(a, v_a)((b, v_b) + (c, v_c)) = (a, v_a)(b, v_b) + (a, v_a)(c, v_c)$. If $v_b = v_c$, the result follows from the left distributivity in $\mathbb{Z}_m$: $(a, v_a)((b, v_b) + (c, v_b)) = (a, v_a)(|b + c|_m, v_b) = (|a(b + c)|_m, v_a + v_u) = (|ab + ac|_m, v_a + v_b) = (a, v_a)(b, v_b) + (a, v_a)(c, v_c)$. Now assume w.l.o.g. that $v_b < v_c$. Then $(b, v_b) + (c, v_c) = (b, v_b)$ holds, resulting in $(a, v_a)(b, v_b)$ on the LHS of the law. The RHS has $v_a + v_b < v_a + v_c$, thus resulting in $(a, v_a)(b, v_b)$. The right distributive law follows analogue.
– Now the associativity of addition follows from Proposition 16. □

Proposition 16. *For a projection $\mathbb{W}_m$ of $\mathbb{W}_\beta$ to a single entry holds that its elements are associative w.r.t. addition.*

Proof. We need to show that for $(a, v_a), (b, v_b), (c, v_c) \in \mathbb{W}_m$ holds $(a, v_a) + ((b, v_b) + (c, v_c)) = ((a, v_a) + (b, v_b)) + (c, v_c)$. The case $v_a = v_b = v_c$ is trivial and follows from the associativity of $\mathbb{Z}_m$ and of $\mathbb{Z}$. The case, where only one pair is equal, is a reduced form of the following considerations and of the trivial case. So, the interesting case is that of distinct v_a, v_b, v_c. As we know, for, say $a + b$ and $v_a < v_b$, the result is a for non-zero a and b. So, we need to regard

[3] In fact, the next addition silently does exactly that. However the next multiplication will propagate the non-standard representation further.

```
eeaStep :: (Integral a) ⇒ ((a, a), (a, a)) → ((a, a), (a, a))
eeaStep ((a1, a2), (b1, b2)) = ((a2, a3), (b2, b3))
    where t = a1 ‘div‘ a2
          a3 = a1 ‘mod‘ a2
          b3 = b1 - t∗b2

eea :: (Integral a) ⇒ ((a, a), (a, a)) → ((a, a), (a, a))
eea ((a1, a2), (b1, b2))
  | a2 == 0 = ((a1, a2), (b1, b2))
  | otherwise = eea $ eeaStep ((a1, a2), (b1, b2))
```

```
convertFraction :: (Integral i) ⇒ Fraction i → i → Mod i
convertFraction (F x y) p
  = let ((d,_),(r,_)) = eea ((p,y),(0,x))
    in if d ≠ 1 then error "convertFraction" else makeZ r p
-- Type Mod i and function makeZ are explained on Figure 1.
```

```
eeaSearch :: (Integral a) ⇒ ((a, a), (a, a)) → a → Maybe (a, a)
eeaSearch ((a1, a2), (b1, b2)) n
  | a2==0 = Nothing
  | a2≠0 ∧ ¬(criteria a2 b2 n)
    = flip eeaSearch n $ eeaStep ((a1, a2), (b1, b2))
  | otherwise = Just (a2, b2)
  where criteria x y n = abs x < n ∧ abs y < n

restoreFraction :: (Integral i) ⇒ i → i → Maybe (i, i)
restoreFraction a m = eeaSearch ((a, m), (0, 1)) n
  where n = nFromM m -- converts m to n per (1), see Figure 3.
```

Fig. 7. A generic (Algorithm 1) and two special (Algorithms 2 and 3) implementations of extended euclidean algorithm in Eden

```
instance (Integral a) ⇒ Num (FSingleMod a) where
    (+) x y = addSingle x y
    (−) x y = x + (additiveInverseSingle y)
    -- etc.
```

```
addSingle :: (Integral a) ⇒ FSingleMod a → FSingleMod a → FSingleMod a
addSingle (FM (Z 0 p) _) (FM (Z 0 p′) _) | p==p′ = FM (Z 0 p) 0
addSingle (FM (Z 0 _) _) y = y
addSingle x (FM (Z 0 _) _) = x
addSingle (FM u 0) (FM u′ 0) = FM (u+u′) 0
addSingle (FM u v) (FM u′ 0)  | v >0  = FM u′ 0
                              | v <0  = FM u v
addSingle (FM u 0) (FM u′ v′) | v′>0  = FM u 0
                              | v′<0  = FM u′ v′
addSingle (FM u v) (FM u′ v′) | v<v′  = FM u v
                              | v>v′  = FM u′ v′
                              | v==v′ = FM (u+u′) v
addSingle _ _ = error "Bad case!" -- never happened
```

```
additiveInverseSingle (FM (Z u p) v) = FM (Z (p-u) p) v
```

Fig. 8. Additive operations in a single fractional residue class

different relations between v_a, v_b, v_c, as they lead to different results. However, we show that the result of the addition is the same in both cases, provided the case constellation of v_a, v_b, v_c. However, the trick is that (in some cases) the particular order of the addition provides us with more information, than another.

We assume the commutativity of the addition, see the above proof. We write down the results in a table, were we represent the pairs of v's with the relation between them. We mean here $<$ just as per above. The boxed values show the relation of addition, which can be deduced from the relations in the given case. There are some further cases, like $c < b < a$ and like $c < a, c < b$, but all the

	$(a+b)+c$		$a+(b+c)$
1.	$\begin{array}{cccc} & v_a & v_b & v_c \\ v_a & = & < & < \\ v_b & & = & ? \\ v_c & & & = \end{array}$	$\Leftarrow$	$\begin{array}{cccc} & v_a & v_b & v_c \\ v_a & = & < & \boxed{<} \\ v_b & & = & < \\ v_c & & & = \end{array}$
2.	$\begin{array}{cccc} & v_a & v_b & v_c \\ v_a & = & < & \\ v_b & & = & < \\ v_c & & & = \end{array}$	$\Leftrightarrow$	$\begin{array}{cccc} & v_a & v_b & v_c \\ v_a & = & < & \\ v_b & & = & < \\ v_c & & & = \end{array}$
1'.	$\begin{array}{cccc} & v_a & v_b & v_c \\ v_a & = & < & > \\ v_b & & = & \boxed{<} \\ v_c & & & = \end{array}$	$\Rightarrow$	$\begin{array}{cccc} & v_a & v_b & v_c \\ v_a & = & ? & > \\ v_b & & = & < \\ v_c & & & = \end{array}$
2'.	$\begin{array}{cccc} & v_a & v_b & v_c \\ v_a & = & > & \\ v_b & & = & < \\ v_c & & & = \end{array}$	$\Leftrightarrow$	$\begin{array}{cccc} & v_a & v_b & v_c \\ v_a & = & > & \\ v_b & & = & < \\ v_c & & & = \end{array}$

Relation	Case
$a < b < c$	1.
$b < c < a$	w.l.o.g. 1.
$c < b < a$	w.l.o.g. 1.
$a < c < b$	commutativity 1.
$c < a < b$	1'.
$a < b$, $a < c$	2.
$b < a$, $b < c$	2'. (w.l.o.g. 2.)
$c < a$, $c < b$	w.l.o.g. 2.

cases with swapped symbols hold as we can fix the order of the entries w.l.o.g. We see, that f.e. the cases 1'. and 2'. correspond to cases 1. and 2. The case 1'. results from both w.l.o.g. and commutativity of case 1. So it is sufficient to regard two cases. In the first, the relations are different, but the total order on the one side implies the relation needed on the other side. In the other case, the relations are partial, but equal. □

Fast Generalized Bruhat Decomposition

Gennadi Malaschonok[⋆]

Tambov State University,
Internatsionalnaya 33, 392622 Tambov, Russia
malaschonok@gmail.com

Abstract. The deterministic recursive pivot-free algorithms for computing the generalized Bruhat decomposition of the matrix in the field and for the computation of the inverse matrix are presented. This method has the same complexity as algorithm of matrix multiplication, and it is suitable for the parallel computer systems.

1 Introduction

An LU matrix decomposition without pivoting is a decomposition of the form $A = LU$, a decomposition with partial pivoting has the form $PA = LU$, and decomposition with full pivoting (Trefethen and Bau) has the form $PAQ = LU$, where L and U are lower and upper triangular matrices, P and Q is a permutation matrix.

French mathematician Francois Georges René Bruhat was the first who worked with matrix decomposition in the form $A = VwU$, where V and U are nonsingular upper triangular matrices and w is a matrix of permutation. Bruhat decomposition plays an important role in algebraic group theory. The generalized Bruhat decomposition was introduced and developed by D.Grigoriev[1],[2]. He uses the Bruhat decomposition in the form $A = VwU$, where V and U are upper triangular matrices but they may be singular when the matrix A is singular. In the papers [3] and [4], there was analyzed the sparsity pattern of triangular factors of the Bruhat decomposition of a nonsingular matrix over a field.

Fast matrix multiplication and fast block matrix inversion were discovered by Strassen [5]. The complexity of Strassen's recursive algorithm for block matrix inversion is the same as the complexity of an algorithm for matrix multiplication. But in this algorithm it is assumed that principal minors are invertible and leading elements are nonzero as in the most of direct algorithms for matrix inversion. There are known other recursive methods for adjoint and inverse matrix computation, which have the complexity of matrix multiplications([6]-[8]).

In a general case, it is necessary to find suitable nonzero elements and to perform permutations of matrix columns or rows. Bunch and Hopkroft suggested such algorithm with full pivoting for matrix inversion [9].

The permutation operation is not a very difficult operation in the case of sequential computations by one processor, but it is a difficult operation in the case

[⋆] Supported by the Sci. Program Devel. Sci. Potent. High. School, RNP.2.1.1.1853.

of parallel computations, when different blocks of a matrix are disposed in different processors. A matrix decomposition without permutations is needed for parallel computation for construction of efficient and fast computational schemes.

The problem of obtaining pivot-free algorithm was studied in [10],[11] by S. Watt. He presented the algorithm that is based on the following identity for a nonsingular matrix: $A^{-1} = (A^T A)^{-1} A^T$. Here A^T is the transposed matrix to A, and all principal minors of the matrix $A^T A$ are nonzero. This method is useful for making an efficient parallel program with the help of Strassen's fast decomposition of inverse matrix for dense nonsingular matrix over the field of zero characteristic when field elements are represented by the float numbers. Other parallel matrix algorithms are developed in [12] - [15].

This paper is devoted to the construction of the pivot-free matrix decomposition method in a common case of singular matrices over a field of arbitrary characteristic. The decomposition will be constructed in the form $LAU = E$, where L and U are lower and upper triangular matrices, and E is a truncated permutation matrix, which has the same rank as the matrix A. Then the generalized Bruhat decomposition may easily be obtained using the matrices L, E, and U. This algorithm has the same complexity as matrix multiplication and does not require pivoting. For singular matrices, it allows the obtaining of a nonsingular block of the biggest size, the echelon form, and kernel of matrix. The preliminary variants of this algorithm were developed in [16] and [17].

2 Preliminaries

We introduce some notations that will be used in the following sections.

Let F be a field, $F^{n\times n}$ be an $n\times n$ matrix ring over F, S_n be a permutation group of n elements. Let P_n be a multiplicative semigroup in $F^{n\times n}$ consisting of matrices A having exactly rank(A) nonzero entries, all of them equal to 1. We call P_n the permutation semigroup because it contains the permutation group of n elements S_n and all their truncated matrices.

The semigroup $D_n \subset P_n$ is formed by the diagonal matrices. So $|D_n|=2^n$ and the identity matrix $\mathbf{I}$ is the identity element in D_n, S_n and P_n.

Let $W_{i,j} \in P_n$ be a matrix, which has only one nonzero element in the position (i, j). For an arbitrary matrix E of P_n, which has the rank $n - s$ $(s = 0, ..n)$ we shall denote by $i_{\overline{E}} = \{i_1, .., i_s\}$ the ordered set of zero row numbers and $j_{\overline{E}} = \{j_1, .., j_s\}$ the ordered set of zero column numbers.

Definition 1. *Let $E \in P_n$ be the matrix of the rank $n - s$, let $i_{\overline{E}} = \{i_1, .., i_s\}$ and $j_{\overline{E}} = \{j_1, .., j_s\}$ are the ordered set of zero row numbers and zero columns number of the matrix E. Let us denote by $\overline{E}$ the matrix*

$$\overline{E} = \sum_{k=1,..s} W_{i_k,j_k}$$

and call it the complimentary matrix for E. For the case $s = 0$ we put $\overline{E} = 0$.

It is easy to see that $\forall E \in P_n : \; E + \overline{E} \in S_n$, and $\forall I \in D_n : \; I + \overline{I} = \mathbf{I}$. Therefore, the map $I \mapsto \overline{I} = \mathbf{I} - I$ is the involution, and we have $I\overline{I} = 0$. We can define the partial order on D_n: $I < J \Leftrightarrow J - I \in D_n$. For each matrix $E \in P_n$ we shall denote by

$$I_E = EE^T \text{ and } J_E = E^T E$$

the diagonal matrix: $I_E, J_E \in D_n$. The unit elements of the matrix I_E show nonzero rows of the matrix E and the unit elements of the matrix J_E show nonzero columns of the matrix E. Therefore, we have several zero identities:

$$E^T \overline{I}_E = \overline{I}_E E = E\overline{J}_E = \overline{J}_E E^T = 0. \tag{1}$$

For any pair $I, J \in D_n$ let us denote the subset of matrices $F^{n\times n}$

$$F^{n\times n}_{I,J} = \{B : B \in F^{n\times n}, IBJ = B\}.$$

We call them (I,J)-zero matrix. It is evident that $F^{n\times n} = F^{n\times n}_{\mathbf{I},\mathbf{I}}$, $0 \in \cup_{I,J} F^{n\times n}_{I,J}$, and if $I_2 < I_1$ and $J_2 < J_1$ then $F^{n\times n}_{I_2,J_2} \subset F^{n\times n}_{I_1,J_1}$.

Definition 2. *We shall call the factorization of the matrix $A \in F^{n\times n}_{I,J}$*

$$A = L^{-1} E U^{-1}, \tag{2}$$

LEU-decomposition if $E \in P_n$, L is a nonsingular lower triangular matrix, U is an upper unitriangular matrix, and

$$L - \overline{I}_E \in F^{n\times n}_{I,I_E}, \quad U - \overline{J}_E \in F^{n\times n}_{J_E,J}. \tag{3}$$

If (2) is the LEU-decomposition we shall write

$$(L, E, U) = \mathcal{LU}(A),$$

Sentence 1. *Let $(L,E,U) = \mathcal{LU}(A)$ be the LEU-decomposition of matrix $A \in F^{n\times n}_{I,J}$ then*

$$L = \overline{I}_E + ILI_E, \; U = \overline{J}_E + J_E U J, \; E \in F^{n\times n}_{I,J}, \tag{4}$$

$$L^{-1} = \overline{I}_E + L^{-1} I_E, \; U^{-1} = \overline{J}_E + J_E U^{-1}.$$

Proof. The first and second equalities follow from (3). To prove the property of matrix E we use the commutativity of diagonal semigroup D_n:

$$E = LAU = (\overline{I}_E + ILI_E) IAJ (\overline{J}_E + J_E U J) = I(\overline{I}_E + LI_E I) A (\overline{J}_E + J J_E U) J.$$

To prove the property of matrix L^{-1} let us consider the identity

$$\mathbf{I} = L^{-1} L = L^{-1}(\overline{I}_E + LI_E) = L^{-1}\overline{I}_E + \mathbf{I} I_E$$

Therefore, $L^{-1}\overline{I}_E = \overline{I}_E$ and $L^{-1} = L^{-1}(\overline{I}_E + I_E) = \overline{I}_E + L^{-1} I_E$. The proof of the matrix U^{-1} property may be obtained similarly.

Sentence 1 states the property of matrix E, which may be written in the form $I_E < I$, $J_E < J$. We shall call it *the property of immersion*. On the other hand, each zero row of the matrix E goes over to the unit column of matrix L and each zero column of the matrix E goes over to the unit row of matrix U.

Let us denote by $\mathcal{E}_n$ the permutation matrix $W_{1,n} + W_{2,n-1} + .. + W_{n,1} \in S_n$. It is easy to see that if the matrix $A \in F^{n\times n}$ is lower-(upper-) triangular, then the matrix $\mathcal{E}_n A \mathcal{E}_n$ is upper- (lower-) triangular.

Sentence 2. *Let $(L, E, U) = \mathcal{LU}(A)$ be the LEU-decomposition of matrix $A \in F^{n\times n}$, then the matrix $\mathcal{E}_n A$ has the generalized Bruhat decomposition $V_1 w V_2$ and*

$$V_1 = \mathcal{E}_n(L^{-1} - \overline{I}_E)\mathcal{E}_n,\ w = \mathcal{E}_n(E + \overline{E}),\ V_2 = (U^{-1} - \overline{J}_E).$$

Proof. As far as L^{-1} is a lower triangular matrix, and U^{-1} is an upper triangular matrix we see that V_1 and V_2 are upper triangular matrices. Matrix w is a product of permutation matrices so w is a permutation matrix. One easily checks that $V_1 w V_2 = \mathcal{E}_n L^{-1} E U^{-1} = \mathcal{E}_n A$.

Examples

For any matrix $I \in D_n$, $E \in P_n$, $0 \neq a \in F$ the product $(aI + \overline{I})I\ \mathbf{I}$ is a LEU decompositions of matrix aI and the product $(aI_E + \overline{I}_E)E\ \mathbf{I}$ is a LEU decompositions of matrix aE.

3 Algorithm of *LEU* Decomposition

Theorem 1. *For any matrix $A \in F^{n\times n}$ of size $n = 2^k$, $k \geq 0$ a LEU-decomposition exists. For computing such decomposition it is enough to compute 4 LEU-decompositions, 17 multiplications and several permutations for the matrices of size $n = 2^{k-1}$.*

Proof. For the matrix of size 1×1, when $k = 0$, we can write the following *LEU* decompositions

$$\mathcal{LU}(0) = (1, 0, 1) \text{ and } \mathcal{LU}(a) = (a^{-1}, 1, 1), \text{ if } a \neq 0.$$

Let us assume that for any matrix of size n we can write a *LEU* decomposition, and let the given matrix $A \in F^{2n\times 2n}_{I,J}$ have the size $2n$. We shall construct a *LEU* decomposition of matrix A.

First of all we shall subdivide the matrices A, I, J and a desired matrix E into four equal blocks:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, I = \text{diag}(I_1, I_2), J = \text{diag}(J_1, J_2), E = \begin{bmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{bmatrix},$$

and denote

$$I_{ij} = E_{ij}E_{ij}^T,\ \ J_{ij} = E_{ij}^T E_{ij}\ \ \forall i, j \in \{1, 2\}. \tag{6}$$

Let

$$(L_{11}, E_{11}, U_{11}) = \mathcal{LU}(A_{11}), \tag{7}$$

denote the matrices

$$Q = L_{11}A_{12},\ B = A_{21}U_{11}, \tag{8}$$

$$A^1_{21} = B\overline{J}_{11},\ A^1_{12} = \overline{I}_{11}Q,\ A^1_{22} = A_{22} - BE^T_{11}Q. \tag{9}$$

Let

$$(L_{12}, E_{12}, U_{12}) = \mathcal{LU}(A^1_{12}) \text{ and } (L_{21}, E_{21}, U_{21}) = \mathcal{LU}(A^1_{21}), \tag{10}$$

denote the matrices

$$G = L_{21}A^1_{22}U_{12},\ A^2_{22} = \overline{I}_{21}G\overline{J}_{12}. \tag{11}$$

Let us put

$$(L_{22}, E_{22}, U_{22}) = \mathcal{LU}(A^2_{22}), \tag{12}$$

and denote

$$W = (GE^T_{12}L_{12} + L_{21}BE^T_{11}),\ V = (U_{21}E^T_{21}G\overline{J}_{12} + E^T_{11}QU_{12}), \tag{13}$$

$$L = \begin{pmatrix} L_{12}L_{11} & 0 \\ -L_{22}WL_{11} & L_{22}L_{21} \end{pmatrix}, U = \begin{pmatrix} U_{11}U_{21} & -U_{11}VU_{22} \\ 0 & U_{12}U_{22} \end{pmatrix}. \tag{14}$$

We have to prove that

$$(L, E, U) = \mathcal{LU}(A). \tag{15}$$

As far as $L_{11}, L_{12}, L_{21}, L_{22}$ are lower triangular nonsingular matrices, and U_{11}, U_{12}, U_{21}, U_{22} are upper unitriangular matrices we can see in (10) that the matrix L is a lower triangular nonsingular matrix and the matrix U is upper unitriangular.

Let us show that $E \in P_{2n}$. As far as $E_{11}, E_{12}, E_{21}, E_{22} \in P_n$ and $A_{11} = I_1A_{11}J_1$, $A^1_{21} = B\overline{J}_{11}$, $A^1_{12} = \overline{I}_{11}Q$, $A^2_{22} = \overline{I}_{21}G\overline{J}_{12}$ and due to the Sentence 1 we obtain $E_{11} = I_{11}E_{11}J_{11}$, $E_{21} = E_{21}\overline{J}_{11}$, $E_{12} = \overline{I}_{11}E_{12}$, $E_{22} = \overline{I}_{21}E_{22}\overline{J}_{12}$.

Therefore, the unit elements in each of the four blocks of the matrix E are disposed in different rows and columns of the matrix E. So $E \in P_{2n}$, and next identities hold

$$E_{11}E^T_{21} = E_{11}J_{21} = J_{11}E^T_{21} = J_{11}J_{21} = 0, \tag{16}$$

$$E^T_{12}E_{11} = E^T_{12}I_{11} = I_{12}E_{11} = I_{12}I_{11} = 0, \tag{17}$$

$$E_{12}E^T_{22} = E_{12}J_{22} = J_{12}E^T_{22} = J_{12}J_{22} = 0, \tag{18}$$

$$E^T_{22}E_{21} = E^T_{22}I_{21} = I_{22}E_{21} = I_{22}I_{21} = 0. \tag{19}$$

We have to prove that $E = LAU$. This equation in block form consists of four block equalities:

$$\begin{aligned} &E_{11} = L_{12}L_{11}A_{11}U_{11}U_{21}; \\ &E_{12} = L_{12}L_{11}(A_{12}U_{12} - A_{11}U_{11}V)U_{22}; \\ &E_{21} = L_{22}(L_{21}A_{21} - WL_{11}A_{11})U_{11}U_{21}; \\ &E_{22} = L_{22}((L_{21}A_{22} - WL_{11}A_{12})U_{12} - (L_{21}A_{21} - WL_{11}A_{11})U_{11}V)U_{22}. \end{aligned} \tag{20}$$

Therefore, we have to prove these block equalities.

Let us note that from the identity $A_{11} = I_1 A_{11} J_1$ and Sentence 1 we get

$$L_{11} = \overline{I}_{11} + I_1 L_{11} I_{11},\ U_{11} = \overline{J}_{11} + J_{11} U_{12} J_1. \tag{21}$$

Sentence 1 together with equations $A^1_{12} = \overline{I}_{11} L_{11} A_{12}$, $A^1_{21} = A_{21} U_{11} \overline{J}_{11}$, $A^2_{22} = \overline{I}_{21} L_{21} (A_{22} - A_{21} U_{11} E^T_{11} L_{11} A_{12}) U_{12} \overline{J}_{12}$ give the next properties of L- and U-blocks:

$$\begin{array}{ll} L_{12} = \overline{I}_{12} + \overline{I}_{11} I_1 L_{12} I_{12}, & U_{12} = \overline{J}_{12} + J_{12} U_{12} J_2, \\ L_{21} = \overline{I}_{21} + I_2 L_{21} I_{21}, & U_{21} = \overline{J}_{21} + J_{21} U_{12} J_1 \overline{J}_{11}, \\ L_{22} = \overline{I}_{22} + \overline{I}_{21} I_2 L_{22} I_{22}, & U_{22} = \overline{J}_{22} + J_{22} U_{22} J_2 \overline{J}_{12}. \end{array} \tag{22}$$

The following identities can easily be checked now:

$$L_{12} E_{11} = E_{11},\ L_{12} I_{11} = I_{11}, \tag{23}$$

$$E_{11} U_{21} = E_{11},\ J_{11} U_{21} = J_{11}, \tag{24}$$

$$E_{12} U_{22} = E_{12},\ J_{12} U_{22} = J_{12}, \tag{25}$$

$$L_{22} E_{21} = E_{21},\ L_{22} I_{21} = I_{21}. \tag{26}$$

We shall use the following equalities,

$$L_{11} A_{11} U_{11} = E_{11}, L_{12} A^1_{12} U_{12} = E_{12}, L_{21} A^1_{21} U_{21} = E_{21}, L_{22} A^2_{22} U_{22} = E_{22}, \tag{27}$$

which follow from (7),(10), and (12), the equality

$$E_{11} V = I_{11} Q U_{12}, \tag{28}$$

which follows from the definition of the block V in (13), (24), (16) and (6), the equality

$$W E_{11} = L_{21} B J_{11}, \tag{29}$$

which follows from the definition of the block W in (13), (23), (17) and (6).

1. The first equality of (20) follows from (27), (23) and (24).

2. The right-hand side of the second equality of (20) takes the form $L_{12}(\mathbf{I} - I_{11}) Q U_{12} U_{22}$ due to (8), (27) and (28). To prove the second equality we use the definition of the blocks B and A^1_{12} in (8) and (9), then the second equality in (27) and identity (25): $L_{12}(\mathbf{I} - I_{11}) Q U_{12} U_{22} = L_{12} A^1_{12} U_{12} U_{22} = E_{12} U_{22} = E_{12}$.

3. The right-hand side of the third equality of (20) takes the form $L_{22} L_{21} B (\mathbf{I} - J_{11}) U_{21}$ due to definition of the block B (8), the first equality in (27) and (29). To prove the third equality we use the definition of the blocks A^1_{21} in (9), then the third equality in (27) and identity (26): $L_{22} L_{21} B \overline{J}_{11} U_{21} = L_{22} L_{21} A^1_{21} U_{21} = L_{22} E_{21} = E_{21}$.

4. The identity

$$E^T_{12} L_{12} = E^T_{12} L_{12} (I_{11} + \overline{I}_{11}) = E^T_{12} L_{12} \overline{I}_{11} \tag{30}$$

follows from (23) and (17).

We have to check that $(L_{21}A_{22} - WL_{11}A_{12})U_{12} = (L_{21}A_{22} - (GE_{12}^T L_{12} + L_{21}BE_{11}^T)Q)U_{12} = L_{21}(A_{22} - BE_{11}^T Q)U_{12} - GE_{12}^T L_{12}QU_{12} = L_{21}A_{22}^1 U_{12} - GE_{12}^T L_{12}\overline{I}_{11}QU_{12} = G - GE_{12}^T L_{12}A_{12}^1 U_{12} = G - GE_{12}^T E_{12} = G\overline{J}_{12}$, using the definitions of the blocks W in (13), A_{22}^1 and A_{12}^1 in (9), identity (28), the second equality in (27), and definition (6).

We have to check that

$$\begin{aligned}&-(L_{21}A_{21} - WL_{11}A_{11})U_{11}V = -(L_{21}A_{21}U_{11} - WE_{11})V\\ &= (-L_{21}B + L_{21}BJ_{11})V = -L_{21}B\overline{J}_{11}V\\ &= -L_{21}B\overline{J}_{11}(U_{21}E_{21}^T G\overline{J}_{12} + E_{11}^T QU_{12}) = -L_{21}A_{21}^1 U_{21}E_{21}^T G\overline{J}_{12} = -I_{21}G\overline{J}_{12}\end{aligned}$$

using the first equality in (27), the identity (29), the definitions of the blocks V in (13), (1), then the third equality in (27) and definition (6).

To prove the fourth equality we have to substitute obtained expressions into the right-hand side of the fourth equality:

$$L_{22}(G\overline{J}_{12} - I_{21}G\overline{J}_{12})U_{22} = L_{22}\overline{I}_{21}G\overline{J}_{12}U_{22} = L_{22}A_{22}^2 U_{22} = E_{22}.$$

For the completion of the proof of this theorem we have to demonstrate the special form of the matrices U and L: $L - \overline{I}_E \in F_{I,I_E}$ and $U - \overline{J}_E \in F_{J_E,J}$.

The matrix L is invertible and $I_E < I$, therefore, we have to prove that $L = \overline{I}_E + ILI_E$, where $I_E = \text{diag}(I_{11} + I_{12}, I_{21} + I_{22})$, $\overline{I}_E = \text{diag}(\overline{I}_{11}\overline{I}_{12}, \overline{I}_{21}\overline{I}_{22})$, $I = \text{diag}(I_1, I_2)$.

This matrix equality for matrix L (14) is equivalent to the four block equalities:

$$L_{12}L_{11} = I_1 L_{12}L_{11}(I_{11} + I_{12}) + \overline{I}_{11}\overline{I}_{12},\ 0 = I_1 0(I_{21} + I_{22}),$$
$$-L_{22}WL_{11} = -I_2 L_{22}WL_{11}(I_{11} + I_{12}),\ L_{22}L_{21} = I_2 L_{22}L_{21}(I_{21} + I_{22}) + \overline{I}_{21}\overline{I}_{22}.$$

To prove the first block equalities we have to multiply its left-hand side by the unit matrix in the form $\mathbf{I} = (I_1 + \overline{I}_1)$ from the left side and by the unit matrix in the form $\mathbf{I} = (I_{11} + I_{12}) + \overline{I}_{11}\overline{I}_{12}$ from the left side. Then we use the following identities to obtain in the left-hand side the same expression as in the right-hand side: $L_{11}\overline{I}_{11} = \overline{I}_{11}$, $L_{12}\overline{I}_{12} = \overline{I}_{12}$, $\overline{I}_1 L_{12}L_{11} = \overline{I}_1$, $\overline{I}_1(I_{11} + I_{12}) = 0$. The same idea may be used for proving the last block equality, but we must use other forms of unit matrix: $\mathbf{I} = (I_2 + \overline{I}_2)$, $\mathbf{I} = (I_{21} + I_{22}) + \overline{I}_{21}\overline{I}_{22}$.

The second block equality is evident.

Let us prove the third block equality. We have to multiply the left-hand side of the third block equality by the unit matrix in the form $\mathbf{I} = (I_2 + \overline{I}_2)$ from the left side and by the unit matrix in the form $\mathbf{I} = (I_{11} + I_{12}) + \overline{I}_{11}\overline{I}_{12}$ from the right side.

The block W is equal to the following expression by definitions (13), (11), and (8):

$$W = (L_{21}(A_{22} - A_{21}U_{11}E_{11}^T Q)U_{12}E_{12}^T L_{12} + L_{21}A_{21}U_{11}E_{11}^T).$$

We have to use in the left-hand side the equations $\overline{I}_2 L_{22} = \overline{I}_2$, $\overline{I}_2 L_{21} = \overline{I}_2$, $\overline{I}_2 A_{22} = 0$, $\overline{I}_2 A_{21} = 0$, and $L_{11}\overline{I}_{11} = \overline{I}_{11}$, $L_{12}\overline{I}_{12} = \overline{I}_{12}$, $E_{12}^T\overline{I}_{12} = 0$, $E_{11}^T\overline{I}_{11} = 0$.

The property of the matrix U: $U - \overline{J}_E \in F_{J_E,J}$ may be proved in the same way as the property of the matrix L.

Theorem 2. *For any matrix A of size $s (s \geq 1)$, an algorithm of LEU-decomposition exists, which has the same complexity as matrix multiplication.*

Proof. We have proved an existence of LEU-decomposition for matrices of size $2^k, k > 0$. Let $A \in F_{I,J}^{s\times s}$ be a matrix of size $2^{k-1} < s < 2^k$, A' be a matrix of size 2^k, which has in the left upper corner the submatrix equal to A, and all other elements equal zero. We can construct LEU-decomposition of matrix A': $(L', E', U') = \mathcal{LU}(A')$. According to the Sentence 1 the product $L'A'U' = E'$ has the form

$$\begin{pmatrix} L & 0 \\ 0 & \mathbf{I} \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} U & 0 \\ 0 & \mathbf{I} \end{pmatrix} = \begin{pmatrix} E & 0 \\ 0 & 0 \end{pmatrix}$$

Therefore, $LAU = E$ is a LEU decomposition of matrix A.

The total amount of matrix multiplications in (7)-(15) is equal to 17, and total amount of recursive calls is equal to 4. We do not consider multiplications of the permutation matrices, we can do these multiplications due to permutation of pointers for the blocks which are disposed at the local processors.

We can compute the decomposition of the 2×2 matrix by means of 5 multiplicative operations. Therefore, we obtain the following recurrence equality for complexity

$$t(n) = 4t(n/2) + 17M(n/2), t(2) = 5.$$

Let γ and β be constants, $3 \geq \beta > 2$, and let $M(n) = \gamma n^\beta + o(n^\beta)$ be the number of multiplication operations in one $n \times n$ matrix multiplication.

After summation from $n = 2^k$ to 2^1 we obtain

$$17\gamma(4^0 2^{\beta(k-1)} + \ldots + 4^{k-2}2^{\beta 1}) + 4^{k-2}5 = 17\gamma\frac{n^\beta - 2^{\beta-2}n^2}{2^\beta - 4} + \frac{5}{16}n^2.$$

Therefore, the complexity of the decomposition is

$$\sim \frac{17\gamma n^\beta}{2^\beta - 4}.$$

If A is an invertible matrix, then $A^{-1} = UE^TL$, and a recursive block algorithm of matrix inversion is written in expressions (7)–(15). This algorithm has the complexity of matrix multiplications.

4 Conclusion

The algorithms for finding the generalized Bruhat decomposition and matrix inversion are described. These algorithms have the same complexity as matrix multiplication and do not require pivoting. For singular matrices, they allow to obtain a nonsingular block of the biggest size. These algorithms may be used in any field, including real and complex numbers, finite fields and their extensions.

The proposed algorithms are pivot-free and do not change the matrix block structure. So they are suitable for parallel hardware implementation.

References

1. Grigoriev, D.: Analogy of Bruhat decomposition for the closure of a cone of Chevalley group of a classical serie. Soviet Math. Dokl. 23(2), 393–397 (1981)
2. Grigoriev, D.: Additive complexity in directed computations. Theoretical Computer Science 19, 39–67 (1982)
3. Kolotilina, L.Y.: Sparsity of Bruhat decomposition factors of nonsingular matrices. Notes of Scientific Seminars of LOMI 202, 5–17 (1992)
4. Kolotilina, L.Y., Yemin, A.Y.: Bruhat decomposition and solution of linear algebraic systems with sparse matrices. Sov. J. Numer. Anal. and Math. Model. 2, 421–436 (1987)
5. Strassen, V.: Gaussian Eelimination is not optimal. Numerische Mathematik 13, 354–356 (1969)
6. Malaschonok, G.I.: Effective matrix methods in commutative domains. In: Formal Power Series and Algebraic Combinatorics, pp. 506–517. Springer, Berlin (2000)
7. Malaschonok, G.I.: Matrix Computational Methods in Commutative Rings. Tambov State University, Tambov (2002)
8. Akritas, A., Malaschonok, G.: Computation of adjoint matrix. In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2006. LNCS, vol. 3992, pp. 486–489. Springer, Heidelberg (2006)
9. Bunch, J., Hopkroft, J.: Triangular factorization and inversion by fast matrix multiplication. Mat. Comp. 28, 231–236 (1974)
10. Watt, S.M.: Pivot-free block matrix inversion. In: Maple Conf. 2006, Waterloo, Canada, July 23-26 (2006), http://www.csd.uwo.ca/~watt/pub/reprints/2006-mc-bminv-poster.pdf
11. Watt, S.M.: Pivot-free block matrix inversion. In: Proc. 8th International Symposium on Symbolic and Numeric Algorithms in Symbolic Computation (SYNASC), pp. 151–155. IEEE Computer Society, Los Alamitos (2006)
12. Eberly, W.: Efficient parallel independent subsets and matrix factorization. In: Proc. 3rd IEEE Symposium on Parallel and Distributed Processing, Dallas, USA, pp. 204–211 (1991)
13. Kaltofen, E., Pan, V.: Processor-efficient parallel solution of linear systems over an abstract field. In: Proc. 3rd Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 180–191. ACM Press, New York (1991)
14. Kaltofen, E., Pan, V.: Processor-efficient parallel solution of linear systems II: The general case. In: Proc. 33rd IEEE Symposium on Foundations of Computer Science, Pittsburgh, USA, pp. 714–723 (1992)
15. Kaltofen, E., Pan, V.: Parallel solution of Toeplitz and Toeplitz-like linear systems over fields of small positive characteristic. In: Proc. PASCO 1994: First International Symposium on Parallel Symbolic Computation, pp. 225–233. World Scientific Publishing, Singapore (1994)
16. Malaschonok, G.I.: Parallel Algorithms of Computer Algebra. In: Proc. Conference Dedicated to the 75 Years of the Mathematical and Physical Dept. of Tambov State University, November 22-24, pp. 44–56. Tambov State Univ., Tambov (2005)
17. Malaschonok, G.I., Zuyev, M.S.: Generalized algorithm for computing of inverse matrix. In: 11th Conf. "Derzhavinskie Chteniya", February 2-6, pp. 58–62. Tambov State Univ., Tambov (2006)

Computational Science in Armenia (Invited Talk)

H. Marandjian and Yu. Shoukourian

Institute for Informatics and Automation Problems
National Academy of Science of Republic Armenia

Abstract. This survey is devoted to the development of informatics and computer science in Armenia. The results in theoretical computer science (algebraic models, solutions to systems of general form recursive equations, the methods of coding theory, pattern recognition and image processing), constitute the theoretical basis for developing problem-solving-oriented environments. As examples can be mentioned: a synthesizer of optimized distributed recursive programs, software tools for cluster-oriented implementations of two-dimensional cellular automata, a grid-aware web interface with advanced service trading for linear algebra calculations. In the direction of solving scientific problems that require high-performance computing resources, examples of completed projects include the field of physics (parallel computing of complex quantum systems), astrophysics (Armenian virtual laboratory), biology (molecular dynamics study of human red blood cell membrane), meteorology (implementing and evaluating the Weather Research and Forecast Model for the territory of Armenia). The overview also notes that the Institute for Informatics and Automation Problems of the National Academy of Sciences of Armenia has established a scientific and educational infrastructure, uniting computing clusters of scientific and educational institutions of the country and provides the scientific community with access to local and international computational resources, that is a strong support for computational science in Armenia.

From Petri Nets to Polynomials: Modeling, Algorithms, and Complexity (Abstract) (Invited Talk)

Ernst W. Mayr

Institut für Informatik,
Technische Universität München
Boltzmannstr. 3, 85748 Garching, Germany
mayr@in.tum.de

Abstract. Petri nets are a widely-used model for parallel and distributed systems of concurrent systems using common resources. They admit a precise algebraic formalization as *vector addition* or *vector replacement* systems. If one considers *symmetric* VRS's, it turns out that they are equivalent to finitely presented commutative semigroups or to binomial ideals in a multivariate ring over $\mathbb{Q}$.

We outline and survey the interaction between these domains of computational algebra, system modeling and verification, and, in particular, complexity theory. While many of the fundamental computational problems in these areas turn out to be very complex (*i.e.,* EXPSPACE-complete or even worse, we also present some new results concerning better complexity for restricted subclasses of the problems.

Supporting Global Numerical Optimization of Rational Functions by Generic Symbolic Convexity Tests[⋆]

Winfried Neun[1], Thomas Sturm[2], and Stefan Vigerske[3]

[1] Zuse Institute Berlin, Takustraße 7
14195 Berlin, Germany
neun@zib.de

[2] Dpto. de Matemáticas, Estadística y Computación, Universidad de Cantabria
39071 Santander, Spain
sturmt@unican.es

[3] Department of Mathematics, Humboldt University Berlin, Unter den Linden 6
10099 Berlin, Germany
stefan@math.hu-berlin.de

Abstract. Convexity is an important property in nonlinear optimization since it allows to apply efficient local methods for finding global solutions. We propose to apply symbolic methods to prove or disprove convexity of rational functions over a polyhedral domain. Our algorithms reduce convexity questions to real quantifier elimination problems. Our methods are implemented and publicly available in the open source computer algebra system Reduce. Our long term goal is to integrate Reduce as a "workhorse" for symbolic computations into a numerical solver.

Keywords: Nonlinear Global Optimization, Hybrid Symbolic-Numeric Computation, Convex Functions, Real Quantifier Elimination, Implementation, Reduce.

1 Introduction

Convexity is an important property in nonlinear optimization since it allows to apply efficient local methods for finding global solutions. However, proving convexity of a general nonlinear function is a non-trivial task, for which no general methods are known. In this paper we propose to apply methods originated in computer logic to prove or disprove convexity in the special case of *rational functions.*

A *nonlinear optimization problem* (NLP) is a problem of the following form:

$$\begin{aligned} \text{minimize} \quad & g_0(x), \\ \text{such that} \quad & g_i(x) \le 0, \quad i = 1, \dots, m, \\ & x \in X, \end{aligned} \tag{P}$$

[⋆] This work was supported by the DFG Research Center Matheon *Mathematics for key technologies* in Berlin, http://www.matheon.de where $X \subseteq \mathbb{R}^n$ is polyhedral and $g_i : \mathbb{R}^n \to \mathbb{R}$, $i = 0, \ldots, m$, are functions that are differentiable for all $x \in X$. The function g_0 is the *objective function*, and the $g_i(x) \leq 0$ for $i = 1, \ldots, m$ are *constraints*. The *feasible set* of (P) is the set of *feasible points* in X that satisfy all constraints. In the following we assume that the feasible set is non-empty.

Nonlinear optimization problems arise in various applications, e.g., engineering design, logistics, manufacturing, and the chemical and biological sciences [1,2,3,4,5].

We recall that a set A is called *convex*, if for all x, $y \in A$ and for all $\lambda \in [0,1]$ we have $\lambda x + (1-\lambda)y \in A$. Further, a function $f : \mathbb{R}^n \to \mathbb{R}$ is called *convex* on a convex set $A \subseteq \mathbb{R}^n$ if for all x, $y \in X$ and for all $\lambda \in [0,1]$ we have

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y).$$

A feasible point $\bar{x}$ of (P) is said to be a *global optimum* of (P), if there is no other feasible point that has a lower objective value than $\bar{x}$. Further, a feasible point $\bar{x}$ is said to be a *local optimum* of (P), if there exists no other feasible point in a neighbourhood of $\bar{x}$ which has a lower objective value than $\bar{x}$. Clearly, every global optimum of (P) is also a local optimum. However, in general, not every local optimum is also a global optimum.

Under some regularity conditions, there exist efficient numerical algorithms to find a local optimum of (P) [6]. On the other hand, finding a global optimum solution of (P) is very difficult in general. However, if the set of feasible points is *convex* and the objective function $g_0(x)$ is *convex* on the feasible set, then it can be easily seen that also every local optimum of (P) is a global optimum. Thus, in case that an NLP is known to be convex, methods for finding local optima can be applied for the global optimization of an NLP. A sufficient condition for the feasible set to be convex is that every function $g_i(x)$, $i = 1, \ldots, m$, is convex on X.

Even algorithms for solving nonconvex NLPs [7,5] can profit from information about convexity of a subset of the constraint functions. Such methods usually construct a convex relaxation of (P), the optimal value of which yields a lower bound on the global optimum of (P). The lower bound allows to evaluate the quality of a feasible solution of (P) and to direct the search for a better solution. The convex relaxation is thereby obtained by replacing each function $g_i(x)$, $i = 0, \ldots, m$, by a *convex underestimator* $\breve{g}_i(x)$ that is convex and pointwise less than or equal to $g_i(x)$ on X. The tighter these underestimators are, the better are the lower bounds that can be expected. Unfortunately, there exists no method to construct a tightest convex underestimator for a given function in general. Clearly, if the algorithm knows that a function $g_i(x)$ is already convex, then $\breve{g}_i(x)$ can be chosen to equal $g_i(x)$.

Existing deterministic methods for proving or disproving the convexity of a function (given as composition of elementary expressions) with respect to bounds on its variables are based on

– walking an expression tree and applying convexity rules for function compositions [8], or
– estimating the spectra of the Hessian matrix or its sign in case of a univariate function [9,10], or
– deciding positive semidefiniteness of the interval Hessian [10].

All these approaches may give inconclusive results. For example, the method from [8]—even though very fast—may fail to detect convexity of the function $f(x) = -x/(1+x)$ on the set $X = [0,1]$, since it includes no rules for concluding convexity of a quotient of two non-constant functions. Formulating the function as $f(x) = 1/(1+x) - 1$, however, convexity is proven, since the numerator of $1/(1+x)$ is a positive constant, and the denominator $1+x$ is concave. The second and third method, in contrast, have no problem in proving convexity for $f(x)$, since they only need to prove positivity of the second derivative $f''(x) = 2/(1+x)^3$.

Nevertheless, for the function $f(x) = 2x^7 - 7x^4 + 84x^2 + 42$ a method like [10] may fail to prove convexity of $f(x)$ on the interval $[-1,2]$. In this example the second derivative is $f''(x) = 84(x^5 - x^2 + 2)$. Replacing each occurrence of x by $[-1,2]$ and applying rules for interval arithmetic yields $f''(x) \in 84 \cdot ([-1,32] - [0,4] + [2,2]) = 84 \cdot [-3,34]$, which allows no conclusive result.

Finally, when proving or disproving convexity of a function over a set all these methods can consider only *simple bound constraints* on the variables x_i. These are constraints directly bounding x_i by a number. For instance, the function $f(x) = (x-y)^3$ is obviously convex on the set

$$X = \{\, (x,y) \in \mathbb{R}^2 \mid x,y \in [0,1],\ x \geq y \,\}.$$

However, [8] would fail to prove convexity of $f(x,y)$ since it only considers the simple bounds $x,y \in [0,1]$ and thus does not "see" that $x - y \geq 0$ on X.

In this paper, we present a novel symbolic method to prove or disprove convexity of *rational functions* over polyhedral sets. The key idea is to reduce convexity problems to first-order sentences over the reals and to decide these sentences by quantifier elimination methods. Our original contributions are the following:

– We devise a new complete symbolic method for deciding the convexity of rational functions over polyhedral domains.
– Unlike existing methods, our approach is not restricted to simple bound constraints but can process arbitrary multi-linear constraints.
– We apply *positive quantifier elimination*, which has been successfully used for existential problems in the past [11,12], to universal problems.
– All our methods discussed throughout the paper are efficiently implemented and publicly available in the open-source computer algebra system Reduce.
– We provide and discuss a comprehensive set of benchmark computations to demonstrate the feasibility of our method for established benchmark suites from the NLP community.

The plan of the paper is as follows: In Section 2 we make precise the special case of the convexity problem addressed in this paper. In Section 3 we motivate and develop our various reductions of the problem to suitable first-order sentences. In Section 4 we introduce the concept of positive quantifier elimination and provide an algorithmic reduction of certain convexity problems to make these accessible to this more efficient variant of quantifier elimination. In Section 5 we give asymptotic upper bounds on the time complexity of our method. Section 6 illuminates our work from a software systems point of view and discusses some future plans for our project. In Section 7 we discuss and analyze comprehensive benchmarks carried out for our method. In Section 8 we finally summarize and evaluate our results.

2 Problem Definition

We are now going to precisely state the particular problem addressed in the remainder of our paper. Recall that the exact role of this problem for nonlinear global optimization has been made explicit in the Introduction.

We consider rational functions $f \in \mathbb{Q}(x_1, \dots, x_n)$ as formal objects that establish defining terms for real functions $f : \mathbb{R}^n \to \mathbb{R}$. The *domain* of $f = p/q \in \mathbb{Q}(x_1, \dots, x_n)$, denoted by $\operatorname{dom} f$, is the set of all points $x \in \mathbb{R}^n$ for which the denominator q does not vanish. Note that every real function defined in term of sums, products, and divisions of variables and rational constants can be described by a rational function.

A set $X \subseteq \mathbb{R}^n$ is called *polyhedral*, if it can be written as the intersection of finitely many halfspaces

$$X = \{\, x \in \mathbb{R}^n \mid Ax \le b \,\}, \quad A \in \mathbb{Q}^{n \times m}, \quad b \in \mathbb{Q}^m.$$

A rational function f is called *convex* on a polyhedral set X if for all x, $y \in X$ and for all $\lambda \in [0,1]$ we have

$$f(\lambda x + (1-\lambda)y) \le \lambda f(x) + (1-\lambda) f(y). \tag{1}$$

The question we aim to answer is the following: *Given a rational function $f \in \mathbb{Q}(x_1, \dots, n_n)$ and a polyhedral set $X \subseteq \operatorname{dom} f \subseteq \mathbb{R}^n$, decide whether or not f is convex on X.*

3 Method

We are going to encode various criteria for convexity into first-order sentences over the language $(0, 1, +, -, \cdot)$ of ordered rings, which is also known as the *Tarski algebra* [13]. These sentences are then checked automatically using real quantifier elimination procedures.

Recall our problem statement in the previous section: We are given $f \in \mathbb{Q}(x_1, \dots, x_n)$, which has got integer coefficients. In addition, we are given $X \subseteq \operatorname{dom} f$ polyhedral, which is naturally described by a conjunction γ of linear constraints, i.e. a quantifier-free formula in our language. To start with, our definition (1) above can be directly translated into a first-order formula:

Lemma 1 (Naive Convexity Condition). *Consider a function* $f = \frac{p}{q} \in \mathbb{Q}(x_1, \dots, x_n)$ *and a formula* γ *in* $x_1, \dots, x_n$ *describing a polyhedral set* $X = \{ x \in \mathbb{R}^n \mid \gamma(x) \} \subseteq \mathrm{dom} f$*. Let* $y_1, \dots, y_n$, λ *be new variables, and denote:*

$$\begin{aligned} \delta &= \gamma[y_1/x_1, \dots, y_n/x_n], \\ f_0 &= f(\lambda x_1 + (1-\lambda)y_1, \dots, \lambda x_n + (1-\lambda)y_n), \\ f_1 &= \lambda f + (1-\lambda) f(y_1, \dots, y_n), \end{aligned}$$

where f_0, f_1 *are obtained via evaluation homomorphisms into*

$$\mathbb{Q}(x_1, \dots, x_n, y_1, \dots, y_n)$$

and computation in that field. Denote by N *and* D *the numerator and the denominator of* $f_0 - f_1$*, respectively. Then* f *is convex on* X *if and only if the following first-order sentence holds:*

$$\Phi_1(f, \gamma) = \forall x_1 \dots \forall x_n \forall y_1 \dots \forall y_n \forall \lambda (\gamma \wedge \delta \longrightarrow ND \leq 0). \qquad \square$$

This formulation has got $2n+1$ universal quantifiers. As we will make precise in theory in Section 5 and demonstrate by means of comprehensive example computations in Section 7, the number of quantifiers is the dominant measure of complexity for real quantifier elimination in our case.

As a first optimization we are now going to reduce the number of quantifiers from $2n+1$ to $n+1$. Under some natural conditions, convexity of f is directly related to certain properties of its Hessian $\nabla^2 f$. Recall that a matrix $A \in \mathbb{R}^{n \times n}$ is *positive semidefinite* if $zAz \geq 0$ and *positive definite* if $zAz > 0$ for all $z \in \mathbb{R}^n$. Assume now that f is twice continuously differentiable on X. Then the following are equivalent:

(i) f is convex on X,
(ii) for all $x \in X$ the matrix $(\nabla^2 f)(x) \in \mathbb{R}^{n \times n}$ is positive semidefinite,
(iii) for all $x \in X$ the matrix $(\nabla^2 f)(x) \in \mathbb{R}^{n \times n}$ has got exclusively non-negative eigenvalues.

This gives rise to the following algorithm, which produces an alternative first-order sentence describing convexity:

Subalgorithm 1 (Non-negative Eigenvalues)

Input *a formula* γ *in variables* $x_1, \dots, x_n$ *and a function* $f = \frac{p}{q} \in \mathbb{Q}(x_1, \dots, x_n)$ *that is twice continuously differentiable on* $X = \{ x \in \mathbb{R}^n \mid \gamma(x) \} \subseteq \mathrm{dom} f$*, which must be polyhedral.*

Output *a first-order sentence that is equivalent to* true *if and only if* f *is convex over* X*.*

1. *Compute* $\nabla^2 f \in \mathbb{Q}(x_1, \dots, x_n)^{n \times n}$*. Notice that due to the differentiation rules, the denominators of the entries of* $\nabla^2 f$ *are powers of* q*.*

2. *Compute the characteristic polynomial*
$$\chi = \det(\nabla^2 f - \lambda I) \in \mathbb{Q}(x_1, \dots, x_n)[\lambda]$$
where I denotes the $n \times n$ unit matrix.
3. *Since χ is a linear combination of products of the entries of the matrix $\nabla^2 f - \lambda I \in \mathbb{Q}(x_1, \dots, x_n)[\lambda]^{n\times n}$, it follows that all denominators occurring in the coefficients of χ are once more powers of the denominator q of f and thus do not vanish over X. Let Δ denote the* gcd *of all these coefficient denominators and rewrite $\chi = \frac{\tilde{\chi}}{\Delta}$, where $\tilde{\chi} \in \mathbb{Q}[x_1, \dots, x_n, \lambda]$, $\Delta \in \mathbb{Q}[x_1, \dots, x_n]$. Then for any choice $x_1, \dots, x_n \in X$ we have $\chi(\lambda) = 0$ if and only if $\tilde{\chi}(\lambda) = 0$.*

Altogether we finally obtain:
$$\Phi_{\mathrm{iii}}(f, \gamma) := \forall x_1 \dots \forall x_n \forall \lambda (\gamma \longrightarrow \lambda < 0 \longrightarrow \tilde{\chi} \neq 0). \qquad \square$$

Recall from the problem statement in the previous section our definition (1) of convexity of a function $f : \mathbb{R}^n \to \mathbb{R}$ on a convex set $X \subseteq \mathrm{dom} f$. Similarly to this, f is called *strictly convex* on X if for all x, $y \in X$ with $x \neq y$ and all $\lambda \in [0, 1]$ we have
$$f(\lambda x + (1 - \lambda)y) < \lambda f(x) + (1 - \lambda) f(y). \tag{2}$$
It is easy to see that strict convexity implies convexity. We are now going to exploit this fact for a heuristic test that requires only n quantifiers.

Quite naturally, a similar equivalence as above holds for strict positiveness, but here yet another equivalent enters the stage, which is most interesting from an algorithmic point of view:

(i′) f is strictly convex on X,
(ii′) for all $x \in X$ the matrix $(\nabla^2 f)(x) \in \mathbb{R}^{n\times n}$ is positive definite,
(iii′) for all $x \in X$ the matrix $(\nabla^2 f)(x) \in \mathbb{R}^{n\times n}$ has got exclusively positive eigenvalues,
(iv′) for all $x \in X$ the determinants of all principal subminors of $(\nabla^2 f)(x) \in \mathbb{R}^{n\times n}$ are positive.

This new condition (iv′) can then be effectively expressed as a first-order formula in the Tarski algebra, which gives rise to the following algorithm:

Subalgorithm 2 (Positive Principal Subminors)

Input *a formula γ in variables $x_1, \dots, x_n$ and a function $f = \frac{p}{q} \in \mathbb{Q}(x_1, \dots, x_n)$ such that f is twice continuously differentiable on $X = \{\, x \in \mathbb{R}^n \mid \gamma(x) \,\} \subseteq$* dom$f$*, which must be polyhedral.*
Output *a first-order sentence that is equivalent to* true *if and only if f is strictly convex over X.*

1. *Compute $\nabla^2 f \in \mathbb{Q}(x_1, \dots, x_n)^{n\times n}$. Notice that due to the differentiation rules, the denominators of the entries of $\nabla^2 f$ are powers of q.*

2. *Compute the determinants* $\frac{u_1}{v_1}, \ldots, \frac{u_n}{v_n} \in \mathbb{Q}(x_1, \ldots, x_n)$ *of the principal subminors of* $\nabla^2 f$.
3. *From the Leibniz formula for the determinant it is clear that the denominators* $v_1, \ldots, v_n$ *are again powers of* q. *Since* f *is differentiable on* X *it is in particular continuous. Consequently* q *does not vanish on* X *and neither do the* $v_1, \ldots, v_n$. *Hence for* $x_1, \ldots, x_n \in X$ *a condition* $\frac{u_i}{v_i} > 0$ *can be equivalently rewritten as* $u_i v_i > 0$.

Altogether we finally obtain:

$$\Phi_{\mathrm{iv}'}(f, \gamma) := \forall x_1 \ldots \forall x_n \left(\gamma \longrightarrow \bigwedge_{i=1}^{n} u_i v_i > 0 \right). \qquad \square$$

On the basis that this algorithm yields only a sufficient condition for convexity, the improvement from $n+1$ to n quantifiers might not appear too striking. There is, however, another measure of complexity that turns Subalgorithm 2 considerably superior to Subalgorithm 1: The degrees of the terms in the formula. The degrees in $\Phi_{\mathrm{iv}'}$ only depend on the degrees in the input f and γ. In Φ_{iii}, in contrast, there is additionally the numerator $\tilde{\chi}$ of the characteristic polynomial, the degree of which is bounded from below by the number n of variables.

The degrees are well-known to be a relevant measure for the complexity of real quantifier-elimination [14]. Even more important, low degrees are crucial for the success of the efficient virtual substitution methods [15,16] primarily used by our implementation. If these methods fail, our implementation has to fall back to partial cylindrical decomposition methods [17], which are not only single but double exponential in the number of universal quantifiers.

Our main algorithm now combines the two subalgorithms in the obvious way:

Algorithm 1 (Convexity)

Input *a formula* γ *in variables* $x_1, \ldots, x_n$ *and a function* $f = \frac{p}{q} \in \mathbb{Q}(x_1, \ldots, x_n)$ *such that* f *is twice continuously differentiable on* $X = \{\, x \in \mathbb{R}^n \mid \gamma(x) \,\} \subseteq \mathrm{dom} f$, *which must be polyhedral.*
Output true *if* f *is convex over* X, false *else.*

```
1. $\Phi := \Phi_{\mathrm{iv}'}(f, \gamma)$ by Subalgorithm 2.
2. $\Phi' :=$ realQuantifierElimination($\Phi$)
3. if $\Phi' =$ true then return true.
4. $\Phi := \Phi_{\mathrm{iii}}(f, \gamma)$ by Subalgorithm 1.
5. $\Phi' :=$ realQuantifierElimination($\Phi$)
6. return $\Phi'$
```
$\square$

4 Positive Quantifier Elimination for Universal Formulas

As indicated in the previous section, we employ quantifier elimination procedures contained in the Redlog [18] package of the open source computer algebra system Reduce to finally decide our sentences. The default real quantifier elimination procedure there applies virtual substitution methods [15,16] as long as the degrees of the quantified variables admit this and then falls back to partial cylindrical algebraic decomposition [17]. We are going to refer to this procedure as *QE* in the following.

Besides QE, we take an alternative novel approach: We use a *dual version* of *positive quantifier elimination.* Positive quantifier elimination had been originally developed by the second author for existential sentences. This original version has been successfully applied for discovering oscillations in gene regulatory networks in the area of algebraic biology, where it clearly outperformed QE by all means [11,12].

The essential idea of positive quantifier elimination is to consider problems, where *all* contained variables are known to be positive. Within virtual substitution methods this knowledge can be heuristically exploited in numerous ways; see [11,12] for details. For the partial cylindrical algebraic decomposition [17] we exploit positivity to some extent as well though much less systematically so far. In the following we are going to refer to positive quantifier elimination as *PQE.*

In the situation of this paper, we can systematically arrive at a positive situation in many cases:

Lemma 2 (Shift to the 1. Hyper-Quadrant). *Consider a function* $f \in \mathbb{Q}(x_1, \dots, x_n)$ *and a formula* γ *in* $x_1, \dots, x_n$ *describing a polyhedral set* $X = \{\, x \in \mathbb{R}^n \mid \gamma(x) \,\} \subseteq \operatorname{dom} f$*. Assume that* γ *bounds* x_i *from below by some* $a \in \mathbb{Q}$*. We set* $\hat{\gamma} = \gamma[x_i - a/x_i] \land x_i > 0$ *silently dropping a (positive) denominator after substitution and obtain* $\hat{f} \in \mathbb{Q}(x_1, \dots, x_n)$ *by plugging* $x_i - a$ *for* x_i *into* f*. Then* x_i *is positive on the polyhedral set* $\hat{X}$ *described by* $\hat{\gamma}$*, and* $\hat{f}$ *is convex on* $\hat{X}$ *if and only if* f *is convex on* X*.*

Proof. If x_i is strictly bounded from below by a, then we have just moved our problem along the x_i-axis via a simple linear transformation. If, in contrast, γ guarantees only $a \le x_i$ then we have turned this in addition into a strict order relation. Since, however, f is smooth on $\operatorname{dom} f$, convexity—in contrast to strict convexity—remains invariant. □

Of course the argument about the smoothness of f in the proof could be avoided by shifting by $a + 1$ instead of a. We have observed, however, that this leads to slightly more complicated terms, which can be disadvantageous for the quantifier elimination procedures.

By iterative application of the lemma, we finally arrive at a completely positive situation provided that all variables are explicitly bounded from below. As our examples in Section 7 will demonstrate this is frequently the case for NLP.

In the positive case, we in addition have to take care of the variables λ in Φ_1 from Lemma 1 and in Φ_{iii} from Subalgorithm 1. To Φ_1 we conjunctively add within the scope of the quantifiers the case $\lambda = 0$ via substitution. In Φ_{iii} we substitute within the scope of the quantifiers $-\lambda$ for λ.

5 Complexity

The complexity of our method is dominated by the quantifier elimination step. This is asymptotically bounded by an exponential function in the number of quantifiers, i.e. essentially the dimension n of the domain $\mathbb{R}^n$ of f. As there are no quantifier alternations in our case, that bound is only single exponential [19,15]. Notice that when we have to use Subalgorithm 1 in contrast to Subalgorithm 2, the additional quantifier $\forall\lambda$ contributes exponentially to the complexity. Either do the quantifiers $\forall y_1, \ldots, \forall y_n$ with the naive approach according to Lemma 1.

6 System Architecture

The computations of the entities needed here, such as derivatives, matrices, and Hessians are done using the computer algebra system Reduce. The system is the well-known host of the Redlog[1] software system which is essential for the algorithms presented above. Reduce is free software since January 2009 which allows us to manage the communication between several independent tasks, e.g., by modifications and technical add-ons to the base system. The system is hosted at SourceForge[2]. Information on the Reduce system in general can be found at its website[3]. It is considered to make the PSL-based Reduce system available also as a linkable library in the near future, which could be easily used by other (e.g. numerical) software systems.

For the experiments that are discussed in the next section, we have let a numerical optimization software write the function f and the linear inequalities that state the set X into a file, which is then read in by Reduce. The interpretation of the generated outputs is done automatically by using standard Linux tools. However, to allow a seamless integration of Reduce as a service for symbolic computations in an optimization software, a more efficient mechanism for communication is currently developed. We have chosen as basis to send binary objects via shared memory, which avoids the overhead of coding and decoding character strings. In our case the data send to Reduce is relatively large (f and X) compared to the results (convex, strictly convex, not convex, or unknown).

It is an interesting option to run this software on a parallel system, since the evaluations of convexity issues for multiple formulas are independent from each other. In fact, we have run our examples on a 16 processors (2.8 GHz each) x86_64 machine with 256 GB of memory under Linux.

7 Examples

In order to evaluate our different convexity test methods, we assembled a test set of NLPs and MINLPs from various sources; an MINLP is an NLP where some variables are additionally restricted to take only integer variables.

[1] `http://www.redlog.eu`
[2] `http://reduce-algebra.sourceforge.net`
[3] `http://www.reduce-algebra.com`

Firstly, the COPS testset [20] is a collection of difficult NLP models which have their origin in various applications. It is frequently used to benchmark NLP solvers. From COPS, we selected instantiations of the models `bearing`, `catmix`, `gasoil`, `glider`, `robot`, and `rocket`.

Secondly, we picked some models from the "CMU-IBM Cyber-Infrastructure for MINLP" webpage[4], which collects MINLP models from real-world applications. We selected the models "Periodic Scheduling of Continuous Multiproduct Plants" (#34), "Stabilizing controller design and the Belgian chocolate problem" (#57), "The Delay Constrained Routing Problem" (#63), and "Simultaneous Cyclic Scheduling and Control of a Multiproduct CSTR" (#71).

Thirdly, we took two instances from a recent paper on solvers for convex MINLPs [21] and one instance from the MINLPLib [22]. These are a constrained layout problem (`clay`), a stochastic service system design problem (`sssd`), and the instance `du-opt5`. Further, we selected a formula for so-called "second-order isotherms" as they appear in the modeling of chromatographic separation processes [1].

Finally, we added the three convex functions that were mentioned in the Introduction as counterexamples to the completeness of existing approaches.

For each NLP, we selected those constraints that are nonlinear and rational functions. For equational constraints $g_i(x) = 0$ we considered also $-g_i(x)$ in order to check for concavity of $g_i(x)$ too. Sets of functions that differ only in the naming of the variables were replaced by one representative.

For each example obtained this way, we have proceeded as follows:

1. (a) Compute Φ_1 according to Lemma 1, and apply QE.
   (b) Compute $\Phi_{\mathrm{iv}'}$ according to Subalgorithm 2, and apply QE.
   (c) If $\Phi_{\mathrm{iv}'}$ did not yield true, i.e. strict convexity, then compute Φ_{iii} according to Subalgorithm 1, and apply QE.
2. If the variables in the example are bounded from below, then move the problem to the first hyper-quadrant, and proceed as in 1. (a)–1. (c) but with PQE instead of QE.

For every single computation 1. (a), ..., 2. (c) we imposed a timelimit of 10 minutes after which non-finished computations were automatically interrupted. The steps (b) and (c) reflect our proposed Algorithm 1, while the steps (a) are supposed to demonstrate that our algorithmic ideas formulated in Subalgorithm 1 and Subalgorithm 2 outperform the naive approach from Lemma 1. Finally, by considering 1. vs. 2., we are able to judge the efficiency of PQE compared to regular QE.

In Table 1 there are results given for all examples, where PQE is used whenever possible and regular QE else. The columns *example* and *function* show the names of the NLP and the function $f = p/q$ in the NLP that are considered in this line, respectively. The columns n, $\deg p$, and $\deg q$ show the number of variables in f, the total degree of the polynomial p, and the total degree of the polynomial q,

[4] `http://www.minlp.org`

respectively. The column *curvature* shows whether f was proven to be strictly convex, convex, not convex. It states "unknown", if no method was able to give a result within the time limit. Column PQE indicates whether there was positive quantifier elimination applied. The columns Φ_1, $\Phi_{iv'}$, Φ_{iii} present the corresponding running times in milliseconds, while "$\perp$" is printed if the method hit the time limit of 10 minutes. Recall that Φ_{iii} is not considered if $\Phi_{iv'}$ already yields that the function is strictly convex. In that case we have "–" instead of a running time.

Note that our method was able to prove convexity for all examples mentioned in the introduction including the example $f = (x - y)^3$, where convexity is only given on the region defined by the linear condition $x - y \geq 0$. Furthermore, we were able to decide convexity for the formulas in the instances `catmix100` and `robot50`, whereas [8] reported inconclusive results.

For those examples, where PQE could be used and Table 1 thus gives PQE timings instead of QE timings, Table 2 explicitly compares the computation times of PQE and regular QE.

Table 1. Results and timings of tests for convexity. All times are given in milliseconds, $\perp$ indicates computations that did not finish within 10 minutes of CPU time.

example	function	n	deg p	deg q	curvature	PQE	Φ_1	$\Phi_{iv'}$	Φ_{iii}
Introduction									
`intro`	$-x/(1+x)$	1	1	1	strictly convex	✓	20	10	–
	$2x^7 - 7x^4 + 84x^2 + 42$	1	7	0	convex	✓	$\perp$	< 10	10
	$(x_1 - x_2)^3$	2	3	0	convex	✓	100	10	< 10
COPS test set [20]									
`bearing`	`e10_0`	3	5	0	not convex	✓	3610	< 10	$\perp$
	`e10_1`	1	2	0	strictly convex	✓	10	< 10	–
	`e13`	2	2	0	not convex	✓	10	< 10	< 10
	`e2`	2	2	0	not convex	✓	20	10	10
	`e3_1`	2	2	0	not convex	✓	10	< 10	< 10
	`e4`	2	2	0	not convex	✓	< 10	< 10	10
	`e5_0`	3	3	0	not convex	✓	210	10	590
	`e6`	4	2	0	not convex	✓	290	< 10	< 10
	`-e13`	2	2	0	not convex	✓	20	< 10	< 10
	`-e2`	2	2	0	not convex	✓	20	< 10	10
	`-e3_1`	2	2	0	not convex	✓	10	< 10	10
	`-e4`	2	2	0	not convex	✓	< 10	< 10	< 10
	`-e5_0`	3	3	0	not convex	✓	2060	< 10	180
	`-e6`	4	2	0	not convex	✓	120	10	10
`catmix100`	`e103`	6	2	0	not convex	–	10	< 10	< 10
	`e3`	6	2	0	not convex	–	20	10	< 10
	`-e103`	6	2	0	not convex	–	10	10	10
	`-e3`	6	2	0	not convex	–	20	< 10	10
`gasoil50`	`e1100_0`	2	3	0	not convex	–	20	< 10	< 10
	`e1100_1`	2	2	0	not convex	–	10	< 10	10
	`e899`	194	2	0	unknown	–	$\perp$	$\perp$	$\perp$
	`e900`	3	3	0	not convex	–	40	< 10	< 10
	`-e1100_0`	2	3	0	not convex	–	20	< 10	< 10
	`-e1100_1`	2	2	0	not convex	–	10	10	< 10
	`-e900`	3	3	0	not convex	–	30	< 10	10

continued on next page

continued from previous page

example	function	n	deg p	deg q	curvature	PQE	Φ_1	$\Phi_{\mathrm{iv}'}$	Φ_{iii}
`glider50`	`e207`	2	4	0	not convex	✓	3650	< 10	30
	`e307`	2	3	0	not convex	✓	30	< 10	< 10
	`e309`	5	2	1	not convex	–	60	10	10
	`e3`	1	2	0	not convex	✓	10	10	< 10
	`e561`	4	2	0	not convex	–	10	10	10
	`e610`	3	2	0	not convex	–	< 10	10	10
	`-e207`	2	4	0	not convex	✓	149570	< 10	30
	`-e307`	2	3	0	not convex	✓	40	< 10	< 10
	`-e309`	5	2	1	not convex	–	40	10	10
	`-e3`	1	2	0	strictly convex	✓	< 10	< 10	–
	`-e561`	4	2	0	not convex	–	20	< 10	< 10
	`-e610`	3	2	0	not convex	–	10	10	< 10
`robot50`	`e203`	5	3	2	not convex	–	4790	< 10	⊥
	`e3`	3	2	0	not convex	–	10	< 10	10
	`e400`	1	2	0	not convex	✓	10	< 10	< 10
	`-e203`	5	3	2	not convex	–	60840	< 10	⊥
	`-e3`	3	2	0	not convex	–	< 10	< 10	< 10
	`-e400`	1	2	0	strictly convex	✓	< 10	< 10	–
`rocket50`	`e105`	3	2	0	not convex	✓	10	< 10	< 10
	`e154`	6	3	1	not convex	✓	760	10	480
	`e253`	3	2	0	not convex	✓	20	< 10	< 10
	`e53`	1	0	2	not convex	✓	30	< 10	< 10
	`-e105`	3	2	0	not convex	✓	10	< 10	< 10
	`-e154`	6	3	1	not convex	✓	760	< 10	480
	`-e253`	3	2	0	not convex	✓	10	< 10	< 10
	`-e53`	1	0	2	strictly convex	✓	30	< 10	–
minlp.org									
`minlp_org_34a`	`balrecs_i1_k2_t1_`	3	2	1	not convex	✓	20	< 10	50
	`balrecs_i1_k2_t4_`	4	2	1	not convex	✓	50	< 10	900
	`-balrecs_i1_k2_t1_`	3	2	1	not convex	✓	60	< 10	80
	`-balrecs_i1_k2_t4_`	1	1	0	unknown	✓	⊥	⊥	⊥
	`-balrecs_i1_k2_t5_`	4	2	1	not convex	✓	30	10	640
	`-object`	45	2	0	unknown	–	⊥	3010	⊥
	`object`	45	2	0	unknown	–	⊥	2810	⊥
`minlp_org_34b`	`defrate_i1_k2_`	3	2	0	not convex	✓	10	< 10	< 10
	`-defrate_i1_k2_`	3	2	0	not convex	✓	10	< 10	10
	`-object`	11	3	0	unknown	–	⊥	50	⊥
	`object`	11	3	0	unknown	–	⊥	50	⊥
`minlp_org_57`	`defrx_3_1_`	3	2	1	not convex	✓	50	< 10	720
	`-defrx_3_1_`	3	2	1	not convex	✓	30	< 10	360
	`-polyy2`	6	3	0	not convex	✓	810	< 10	⊥
	`-polyy3`	7	4	0	unknown	✓	⊥	20	⊥
`minlp_org_71`	`fecolc_1_1_1_`	4	2	0	not convex	–	20	40	100
	`-fecolc_1_1_1_`	4	2	0	not convex	–	20	40	110
	`-obj`	611	3	1	unknown	✓	⊥	⊥	⊥
	`-odec_20_3_5_`	2	3	0	not convex	✓	20	< 10	10
	`odec_20_3_5_`	2	3	0	not convex	✓	140	< 10	10
miscellaneous [1,21,22]									
`clay0203h`	`e107`	3	3	1	strictly convex	✓	⊥	10	–
	`e110`	3	3	1	strictly convex	✓	⊥	170	–
`sssd-8-4-3`	`e30`	1	1	1	strictly convex	✓	10	10	–
`du-opt5`	`e1`	18	2	0	strictly convex	✓	⊥	180	–
`isotherms`	`isotherm`	2	2	2	not convex	✓	⊥	670	2320
	`-isotherm`	2	2	2	not convex	✓	⊥	660	2290

Table 2. Times of regular vs. positive quantifier elimination. All times are given in milliseconds, $\perp$ indicates computations that did not finish within 10 minutes of CPU time.

example	function	n	deg p	deg q	curvature	Φ_1		$\Phi_{\mathrm{iv}'}$		Φ_{iii}	
						QE	PQE	QE	PQE	QE	PQE
Introduction											
`intro`	$-x/(1+x)$	1	1	1	str. conv.	10	20	10	10	–	–
	$2x^7-7x^4+84x^2+42$	1	7	0	conv.	$\perp$	$\perp$	10	< 10	10	10
	$(x_1-x_2)^3$	2	3	0	conv.	100	100	< 10	10	< 10	< 10
COPS test set [20]											
`bearing`	`e10_0`	3	5	0	not conv.	3640	3610	< 10	< 10	$\perp$	$\perp$
	`e10_1`	1	2	0	str. conv.	< 10	10	< 10	< 10	–	–
	`e13`	2	2	0	not conv.	10	10	10	< 10	< 10	< 10
	`e2`	2	2	0	not conv.	10	20	10	10	10	10
	`e3_1`	2	2	0	not conv.	< 10	10	< 10	< 10	< 10	< 10
	`e4`	2	2	0	not conv.	10	< 10	< 10	< 10	< 10	10
	`e5_0`	3	3	0	not conv.	230	210	< 10	10	600	590
	`e6`	4	2	0	not conv.	390	290	10	< 10	10	< 10
	`-e13`	2	2	0	not conv.	10	20	10	< 10	< 10	< 10
	`-e2`	2	2	0	not conv.	20	20	< 10	< 10	10	10
	`-e3_1`	2	2	0	not conv.	< 10	10	< 10	< 10	< 10	10
	`-e4`	2	2	0	not conv.	< 10	< 10	10	< 10	< 10	< 10
	`-e5_0`	3	3	0	not conv.	1970	2060	< 10	< 10	200	180
	`-e6`	4	2	0	not conv.	150	120	10	10	10	10
`glider50`	`e207`	2	4	0	not conv.	3760	3650	< 10	< 10	30	30
	`e307`	2	3	0	not conv.	30	30	< 10	< 10	10	< 10
	`e3`	1	2	0	not conv.	< 10	10	< 10	10	< 10	< 10
	`-e207`	2	4	0	not conv.	150280	149570	< 10	< 10	20	30
	`-e307`	2	3	0	not conv.	50	40	< 10	< 10	10	< 10
	`-e3`	1	2	0	str. conv.	< 10	< 10	< 10	< 10	–	–
`robot50`	`e400`	1	2	0	not conv.	< 10	10	< 10	< 10	< 10	< 10
	`-e400`	1	2	0	str. conv.	< 10	< 10	10	< 10	–	–
`rocket50`	`e105`	3	2	0	not conv.	10	10	< 10	< 10	< 10	< 10
	`e154`	6	3	1	not conv.	750	760	10	10	500	480
	`e253`	3	2	0	not conv.	10	20	< 10	< 10	10	< 10
	`e53`	1	0	2	not conv.	30	30	< 10	< 10	< 10	< 10
	`-e105`	3	2	0	not conv.	10	10	< 10	< 10	10	< 10
	`-e154`	6	3	1	not conv.	720	760	10	< 10	460	480
	`-e253`	3	2	0	not conv.	20	10	< 10	< 10	< 10	< 10
	`-e53`	1	0	2	str. conv.	40	30	< 10	< 10	–	–
minlp.org											
`minlp_org_34a`	`balrecs_i1_k2_t1_`	3	2	1	not conv.	20	20	10	< 10	50	50
	`balrecs_i1_k2_t4_`	4	2	1	not conv.	50	50	< 10	< 10	1160	900
	`-balrecs_i1_k2_t1_`	3	2	1	not conv.	50	60	< 10	< 10	80	80
	`-balrecs_i1_k2_t4_`	1	1	0	unknown	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
	`-balrecs_i1_k2_t5_`	4	2	1	not conv.	40	30	< 10	10	790	640
`minlp_org_34b`	`defrate_i1_k2_`	3	2	0	not conv.	< 10	10	10	< 10	< 10	< 10
	`-defrate_i1_k2_`	3	2	0	not conv.	10	10	10	< 10	10	10
`minlp_org_57`	`defrx_3_1_`	3	2	1	not conv.	40	50	10	< 10	700	720
	`-defrx_3_1_`	3	2	1	not conv.	40	30	< 10	< 10	360	360
	`-polyy2`	6	3	0	not conv.	860	810	10	< 10	$\perp$	$\perp$
	`-polyy3`	7	4	0	unknown	$\perp$	$\perp$	30	20	$\perp$	$\perp$
`minlp_org_71`	`-obj`	611	3	1	unknown	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
	`-odec_20_3_5_`	2	3	0	not conv.	20	20	< 10	< 10	10	10
	`odec_20_3_5_`	2	3	0	not conv.	150	140	< 10	< 10	10	10
miscellaneous [1,21,22]											

continued on next page

continued from previous page

example	function	n	deg p	deg q	curvature	Φ_1 QE	Φ_1 PQE	$\Phi_{\mathrm{iv}'}$ QE	$\Phi_{\mathrm{iv}'}$ PQE	Φ_{iii} QE	Φ_{iii} PQE
`clay0203h`	`e107`	3	3	1	str. conv.	$\perp$	$\perp$	< 10	10	–	–
	`e110`	3	3	1	str. conv.	$\perp$	$\perp$	70	170	–	–
`sssd-8-4-3`	`e30`	1	1	1	str. conv.	10	10	< 10	10	–	–
`du-opt5`	`e1`	18	2	0	str. conv.	$\perp$	$\perp$	1480	180	–	–
`isotherms`	`isotherm`	2	2	2	not conv.	$\perp$	$\perp$	620	670	2180	2320
	`-isotherm`	2	2	2	not conv.	$\perp$	$\perp$	610	660	2210	2290

8 Conclusions

Our benchmarking has confirmed that our proposed Algorithm 1 is the most suitable combination of the methods introduced throughout this paper. Furthermore, PQE should probably be used rather than regular QE, although the difference in performance is not at all as striking as with the examples previously reported in the literature [11,12]. Of course, we do not consider our symbolic method a stand-alone solution which should replace more efficient though incomplete approaches. We think that it would be a reasonable scheme to first try the very fast convexity rules from [8], then to try to disprove convexity numerically, and finally—in the case of rational functions—apply our Algorithm 1. A conclusive result on nonconvexity can be obtained numerically by computing the Hessian H in some points of X, and using a robust numerical algorithm to find a vector z such that $z^T Hz < 0$ [8, Sec. 5]. Altogether we consider our work described here an encouraging milestone in our research on integrating Reduce as a symbolic library with state-of-the-art numerical NLP solvers.

References

1. Ballerstein, M., Michaels, D., Seidel-Morgenstern, A., Weismantel, R.: A theoretical study of continuous counter-current chromatography for adsorption isotherms with inflection points. Computers & Chemical Engineering 34(4), 447–459 (2010)
2. Grossmann, I.E. (ed.): Global Optimization in Engineering Design. Kluwer Academic Publishers, Dordrecht (1996)
3. Grossmann, I.E., Kravanja, Z.: Mixed-integer nonlinear programming: A survey of algorithms and applications. In: Conn, A., Biegler, L., Coleman, T., Santosa, F. (eds.) Large-Scale Optimization with Applications, Part II: Optimal Design and Control. Springer, Heidelberg (1997)
4. Jüdes, M., Tsatsaronis, G., Vigerske, S.: Optimization of the design and partial-load operation of power plants using mixed-integer nonlinear programming. In: Kallrath, J., Pardalos, P., Rebennack, S., Scheidt, M. (eds.) Optimization in the Energy Industry. Springer, Heidelberg (2009)
5. Tawarmalani, M., Sahinidis, N.V.: Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications. Kluwer Academic Publishers, Dordrecht (2002)
6. Nocedal, J., Wright, S.: Numerical Optimization. Springer, Heidelberg (2000)
7. Adjiman, C.S., Floudas, C.A.: Rigorous convex underestimators for general twice-differentiable problems. Journal of Global Optimization 9, 23–40 (1997)

8. Fourer, R., Maheshwari, C., Neumaier, A., Orban, D., Schichl, H.: Convexity and concavity detection in computational graphs: Tree walks for convexity assessment. INFORMS Journal on Computing 22(1), 26–43 (2009)
9. Mönnigmann, M.: Efficient calculation of bounds on spectra of Hessian matrices. SIAM Journal on Scientific Computing 30(5), 2340–2357 (2008)
10. Nenov, I.P., Fylstra, D.H., Kolev, L.V.: Convexity determination in the Microsoft Excel solver using automatic differentiation techniques. Technical report, Frontline Systems Inc. (2004)
11. Sturm, T., Weber, A.: Investigating generic methods to solve Hopf bifurcation problems in algebraic biology. In: Horimoto, K., Regensburger, G., Rosenkranz, M., Yoshida, H. (eds.) AB 2008. LNCS, vol. 5147, pp. 200–215. Springer, Heidelberg (2008)
12. Sturm, T., Weber, A., Abdel-Rahman, E.O., El Kahoui, M.: Investigating algebraic and logical algorithms to solve Hopf bifurcation problems in algebraic biology. Mathematics in Computer Science 2(3), 493–515 (2009)
13. Tarski, A.: A decision method for elementary algebra and geometry. Prepared for publication by J.C.C. McKinsey. RAND Report R109, August 1 (1948) (revised May 1951); Second Edition, RAND, Santa Monica, CA (1957)
14. Basu, S., Pollack, R., Roy, M.F.: On the combinatorial and algebraic complexity of quantifier elimination. Journal of the ACM 43(6), 1002–1045 (1996)
15. Weispfenning, V.: The complexity of linear problems in fields. Journal of Symbolic Computation 5(1&2), 3–27 (1988)
16. Weispfenning, V.: Quantifier elimination for real algebra—the quadratic case and beyond. Applicable Algebra in Engineering Communication and Computing 8(2), 85–101 (1997)
17. Collins, G.E., Hong, H.: Partial cylindrical algebraic decomposition for quantifier elimination. Journal of Symbolic Computation 12(3), 299–328 (1991)
18. Dolzmann, A., Sturm, T.: Redlog: Computer algebra meets computer logic. ACM SIGSAM Bulletin 31(2), 2–9 (1997)
19. Davenport, J.H., Heintz, J.: Real quantifier elimination is doubly exponential. Journal of Symbolic Computation 5(1-2), 29–35 (1988)
20. Dolan, E.D., Moré, J.J., Munson, T.S.: Benchmarking optimization software with COPS 3.0. Technical Report ANL/MCS-273, Mathematics and Computer Science Division, Argonne National Laboratory (2004), http://www.mcs.anl.gov/~more/cops
21. Bonami, P., Kilinç, M., Linderoth, J.: Algorithms and software for convex mixed integer nonlinear programs (2009), Optimization Online, http://www.optimization-online.org/DB_HTML/2009/10/2429.html
22. Bussieck, M.R., Drud, A.S., Meeraus, A.: MINLPLib—A Collection of Test Models for Mixed-Integer Nonlinear Programming. INFORMS Journal on Computing 15(1), 114–119 (2003), http://www.gamsworld.org/minlp/minlplib.htm

Term Cancellations in Computing Floating-Point Gröbner Bases

Tateaki Sasaki[1] and Fujio Kako[2]

[1] Professor emeritus, University of Tsukuba,
Tsukuba-city, Ibaraki 305-8571, Japan
sasaki@math.tsukuba.ac.jp

[2] Department of Info. and Comp. Sci., Nara Women's University,
Nara-city, Nara 630-8506, Japan
kako@ics.nara-wu.ac.jp

Abstract. We discuss the term cancellation which makes the floating-point Gröbner basis computation unstable, and show that error accumulation is never negligible in our previous method. Then, we present a new method, which removes accumulated errors as far as possible by reducing matrices constructed from coefficient vectors by the Gaussian elimination. The method manifests amounts of term cancellations caused by the existence of approximate linearly dependent relations among input polynomials.

1 Introduction

Although floating-point Gröbner bases are indispensable in scientific computation, they have been scarcely used so far. The reason is that the computation is so unstable that it is very difficult to obtain desired results.

There are two kinds of floating-point Gröbner bases. The first kind is that the coefficients of the input polynomials are exact (say algebraic numbers or transcendental numbers) but we utilize the floating-point numbers for some reasons. The second kind is that the coefficients of input polynomials are inexact hence we inevitably express the coefficients by floating-point numbers. If the numerical errors increase during the computation, we can replay the computation with higher precision for the first kind, however, for the second kind, we must devise a method to preserve the initial accuracies of input polynomials as far as possible. In this paper, we deal with the second kind. We should emphasize that, since the errors in the input polynomials are unknown, the algorithm should return the bases which do not critically depend on the input errors.

The first kind of floating-point Gröbner bases were studied by Shirayanagi and Sweedler [16,17,18]. The second kind of floating-point Gröbner bases were studied by Stetter [19], Kalkbrener [6], Fortuna, Gianni and Trager [3], Traverso and Zanoni [24], Traverso [23], Weispfenning [25], Kondratyev, Stetter and Winkler [8], Gonzalez-Vega, Traverso and Zanoni [4], Stetter [21], Bodrato and Zanoni [1], and so on. Recently, Suzuki [22] and Nagasaka [11] proposed to compute Gröbner bases by reducing large numerical matrices by Gaussian elimination. In spite of these studies, computation of floating-point Gröbner bases of the second kind has been a serious problem; the computation was so unstable in most cases if performed naively. This seriousness forced European researchers to study the so-called "border bases" [9,10,5], see also [2].

The key point of stabilizing the floating-point Gröbner basis computation is how to control the errors caused by term cancellation; see Sect. 2 for details. We classify the term cancellation into two classes, systematic and accidental. These four years, the present authors have studied this theme and presented the symbolic coefficient method [14] and the high-precision method [15]. The high-precision method is quite stable and useful so long as the accidental cancellations do not occur. However, the method has a weakness: although the coefficients are computed pretty accurately, the estimation of intrinsic errors occurred on the coefficients is very bad. Here, by "intrinsic errors" we mean errors caused by ill-conditionedness of the input basis, which cannot be reduced by the computational techniques. For example, suppose there exists an approximate linearly dependent relation among input polynomials, then, when the relation is computed, the errors of its coefficients will inevitably be increased. Knowing the amounts of intrinsic errors is very important for the Gröbner bases with inexact coefficients. Therefore, we want to develop a method which informs us the amounts of intrinsic errors. In this paper, we present such a method. The idea is to improve the estimation of errors of polynomials which are computed by our previous method, by reducing coefficient matrices by Gaussian elimination.

Since many readers will be unfamiliar with the phenomenon of term cancellation, we will explain our method as elementally as possible using examples.

2 Term Cancellation and Its Monitoring

By F, G, etc., we denote polynomials in $\mathbb{F}[x, y, \ldots, z]$, where $\mathbb{F}$ denotes the floating-point numbers of a fixed precision. By $\|F\|$ we denote the *norm* of F; we employ the infinity norm in this paper. The *power product* is the product of powers of variables. By $\mathrm{supp}(F)$ we denote the *support* of F, i.e., the set of all the power products appearing in F. By $\mathrm{lt}(F)$ and $\mathrm{rt}(F)$, we denote the *leading term* and the *rest terms*, of F, respectively, with respect to a given order $\succ$; $F = \mathrm{lt}(F) + \mathrm{rt}(F)$. By $\mathrm{Spol}(F, G)$ and $\mathrm{Lred}(F, G)$, we denote the *S-polynomial* of F and G and the *leading-term reduction* of F by G, respectively. We express $\mathrm{Lred}(F, G)$ also as $F \xrightarrow{G}$. By $F \overset{G}{\twoheadrightarrow} \tilde{F}$, we denote successive leading-term reductions of F by G so that $\mathrm{lt}(\tilde{F})$ is no longer reducible by G.

If the error of a floating-point number f starts at the (l+1)-st bit then we say that the accuracy of f is $1/2^l$, and we call the leading l bits the significant ones. Let c_1T and c_2T be monomials, where $c_1, c_2 \in \mathbb{F}$. If leading l bits of c_1 and c_2 are the same but the (l+1)-st ones are not then the l bits are lost in the subtraction $c_1T - c_2T$. We call this *term cancellation* of amount 2^l, and the relative error of $c_1 - c_2$ increases by 2^l compared with c_1 and c_2. If the resulting $c_1 - c_2$ has no significant bit then we call the cancellation *exact*, otherwise *inexact*. If the term cancellation is exact, we call the resulting term $(c_1 - c_2)T$ a *fully-erroneous term*.

We may classify the term cancellation as follows.

$$\begin{cases} \textit{systematic} \begin{cases} \textit{exact cancellation} \\ \textit{inexact cancellation} \end{cases} \\[2ex] \textit{accidental} \begin{cases} \textit{exact cancellation} \\ \textit{inexact cancellation} \end{cases} \end{cases}$$

We explain the difference between systematic and accidental cancellations. Let a polynomial Q be contained in both P_1 and P_2 such as $P_i = P_i'+c_iQ$ $(i = 1, 2)$. If $c_1 \simeq c_2$ then c_1Q and c_2Q cancel in the subtraction P_1-P_2. This is a typical systematic cancellation. The cancellation is exact if c_1 and c_2 cancel exactly, otherwise the cancellation is inexact. We also call the case $\|P_1-P_2\| \ll \|P_1\|$ systematic cancellation. This case occurs, for example, if there exists an approximate linear dependence among polynomials in the initial or intermediate bases. Let P_1 and P_2 contain c_1T and c_2T, respectively, where these terms originate from different initial terms. If $c_1 \simeq c_2$ then the term cancellation occurs in the subtraction P_1-P_2. This is the accidental cancellation. The actual term cancellation is complicated because other term $c'T$ may be mixed before the subtraction of c_1T and c_2T. We call $(c_1+c')T-c_2T$ and c_1T-c_2T the term cancellation *with* and *without mixing*, respectively.

The systematic exact cancellation occurs frequently in polynomial linear algebra, such as the computation of determinants with polynomial entries and the polynomial remainder sequences. Similarly, it occurs frequently in the computation of Gröbner bases, as shown in [14,15,13]. If the systematic exact cancellation is caused by polynomials with small leading terms, we usually encounter large cancellation errors. On the other hand, the accidental cancellation occurs rarely, especially if the input polynomials are generated randomly or determined by experiments. In the rest of this section, we survey our previous work briefly.

If a fully-erroneous term appears as the leading term then the subsequent computation becomes meaningless. Therefore, we must remove the fully-erroneous terms completely. So far, two ideas have been proposed to do so. Shirayanagi [16,17,18] proposed to represent the input coefficients by intervals and remove the fully-erroneous terms by replacing the interval by 0 if it contains 0. The present authors devised the so-called *effective floating-point numbers*, or *efloats* in short [7], so as to detect the cancellation errors automatically but approximately. We represent the efloat number as #E(f, e), where f is a floating-point number representing the value of this number, and e is a short floating-point number representing the error of f. We call f and e as *value-part* and *error-part*, respectively. For the arithmetic of efloats, see [7].

Let $\varepsilon_{\rm cal}$ be the precision of floating-point numbers employed; we have $\varepsilon_{\rm cal} \simeq 2\times10^{-16}$ in the double-precision arithmetic. In our algebra system, the error-part of efloat is set slightly larger than $\varepsilon_{\rm cal}|f|$: we set e to $10^{-15}|f|$ in the double-precision arithmetic, and to $100\,\varepsilon_{\rm cal}|f|$ in the high-precision arithmetic. If the input coefficient f contains relative error $\varepsilon_{\rm init}$, then we may set the error-part to $5\,\varepsilon_{\rm init}|f|$, say. Our algebra system sets the efloat #E(f, e) to 0 if $|f| < e$.

The exact term cancellation with mixing is not so simple. Suppose we have $|c_1| \gg c'$ in $((c_1+c')-c_2)T$, then the cancellation of amount $|c_1/c'|$ occurs in the resulting $c_1+c'-c_2$. Fortunately, if the exact cancellation is systematic, we can preserve the initial accuracy in $c_1+c'-c_2$. The idea is to convert all the input coefficients to high precision floating-point numbers [15]. Then, since c_1 and c_2 originate from a single coefficient c of an input polynomial, their errors are the same initially and subsequent computation contaminates only some lower bits of them. Suppose ℓ lower bits of c_1 and c_2 are contaminated at most. Then, the significant part of c' is safe so long as the precision has been increased initially by more than 2^ℓ. This is the essence of the high-precision method.

3 Accidental Cancellations and Current Problems

In this section, we consider accidental cancellations. For simplicity, we assume that the input coefficients are accurate to $\varepsilon_{\rm init}$ at most; if the coefficients are of different accuracies, we set $\varepsilon_{\rm init}$ to the maximum of the accuracies. It is natural to assume that $\varepsilon_{\rm init} \gg \varepsilon_{\rm cal}$.

If accidental exact cancellation without mixing occurs then all the bits above $\varepsilon_{\rm init}$ are lost. The resulting term is also fully-erroneous, and we must remove such terms, too. This removal can be done easily if we represent coefficients by efloats. Suppose the accidental exact cancellation occurs in c_1T-c_2T, hence c_1-c_2 is fully-erroneous. Since c_1 and c_2 originate from different coefficients, c_1-c_2 will be a number of relative error $\sim \varepsilon_{\rm init}$, and we assumed that $\varepsilon_{\rm init} \gg \varepsilon_{\rm cal}$. We can detect this fully-erroneous term by monitoring the corresponding efloat. Thus, we can remove such a fully-erroneous term by replacing an efloat #E$[f,e]$ by 0 if we have $|f| < n_{\rm g}\varepsilon_{\rm init}\, e/(100\,\varepsilon_{\rm cal})$, where $n_{\rm g}$ is a guard number, say 10.

Example 1 (Fully-erroneous term by accidental exact cancellation). We consider the following system with the lexicographic order.

$$\begin{cases} F_1 = x^2\,(2yz+1)/2.0, \\ F_2 = (x\,(3xz-2) - (2yz+1))/3.0, \\ F_3 = (yz\,(3xz-2))/2.0. \end{cases}$$

We first convert the coefficients into double-precision floats. Note that F_3 is input by dividing by 2.0, not by 3.0. This artificial trick introduces different errors into yz terms of F_2 and F_3. Computing the floating-point Gröbner basis by the high-precision method with 30 decimal precisions, we obtain $\{1\}$. Investigating the computation process, we found the following intermediate polynomial.

$$\begin{array}{rl} & \#\mathrm{E}(8.43749999999999945356\cdots\mathrm{e}{-1},\ 4.2\mathrm{e}{-28})\,x^2z \\ - & \#\mathrm{E}(3.70074341541718836536\cdots\mathrm{e}{-17},\ 6.7\mathrm{e}{-28})\,xy^2 \\ + & \#\mathrm{E}(1.12499999999999994795\cdots\mathrm{e}{-0},\ 3.0\mathrm{e}{-28})\,xyz \\ + & \quad\cdots\qquad\cdots \end{array}$$

The above second term is fully erroneous, caused by the accidental cancellation. This term is very small but the result depends on it critically. □

We show a weakness of our previous method by an example.

Example 2 (Large errors in the result). We computed an unreduced Gröbner basis w.r.t. the total-degree order, of the above system, obtaining

$$\begin{cases} G_1 = \#\mathrm{E}(9.9999999999999999999\cdots\mathrm{e}{-1},\ 1.9\mathrm{e}{-25})\, x \\ \quad\ + \#\mathrm{E}(1.3333333333333348136\cdots\mathrm{e}{-0},\ 2.5\mathrm{e}{-25})\, y, \\ G_2 = \#\mathrm{E}(9.9999999999999999999\cdots\mathrm{e}{-1},\ 1.0\mathrm{e}{-28})\, yz \\ \quad\ + \#\mathrm{E}(5.8139534883720897891\cdots\mathrm{e}{-2},\ 3.7\mathrm{e}{-27})\, x \\ \quad\ + \#\mathrm{E}(7.7519379844961242801\cdots\mathrm{e}{-2},\ 4.9\mathrm{e}{-27})\, y \\ \quad\ + \#\mathrm{E}(5.0000000000000001952\cdots\mathrm{e}{-1},\ 6.7\mathrm{e}{-27}). \end{cases}$$

G_2 can be reduced by G_1 and we obtain the same Gröbner basis as that in Example 1. We, however, find that the errors of the above result are large compared with those in Example 1; error-parts of G_1 are about 10^3 larger than its initial values (the value-parts are accurate to $O(10^{-16})$ relatively.) The origin of these large errors can be understood by considering syzygies (a_{i1}, a_{i2}, a_{i3}) for G_i $(i=1,2)$: $G_i = a_{i1}F_1 + a_{i2}F_2 + a_{i3}F_3$. Normalizing the leading coefficients of G_1 to 1, we find that a_{11}, a_{12}, a_{13} contain 49, 54 and 40 terms, of norms 5109/5, 560 and 4439/10, respectively, and we have $\max\{\|a_{11}F_1\|, \|a_{12}F_2\|, \|a_{13}F_3\|\} = 10218/5$. The largeness of $\|a_{ij}\|$ $(1 \le i \le 2;\ 1 \le j \le 3)$ implies that, during Buchberger's procedure, corresponding polynomials are multiplied by large monomials but they will cancel later because the final polynomials are of norm $O(1)$. Therefore, big errors were induced in the final basis. □

Summarizing the above discussions, we have the following problems to solve.

1) Remove the errors caused by accidental exact cancellation with mixing.
2) Remove the errors caused by accidental inexact cancellation.

Furthermore, in our high-precision method, small cancellation errors accumulate gradually but steadily to error-parts of efloats, as Example 2 shows. Therefore, we must solve the following problem, too.

3) Reduce the gradually accumulating errors as far as possible.

4 Our Idea: Reduce the Errors by a Matrix Method

In this paper, *we restrict the reduction of polynomials only to the leading-term reduction*: we compute the Gröbner bases by constructing S-polynomials and performing only the leading-term reductions successively. If we need the reduced Gröbner basis then we perform the reduction of non-leading terms after computing an unreduced Gröbner basis.

The problems given in Sect. 3 are quite difficult to solve algebraically. For example, the problem 3) is inevitable so long as Buchberger's procedure is employed. We note that problems 1) and 2) are common in numerical computation, and numerical analysts developed excellent techniques to suppress the increase of errors. In the case of matrix computation, the QR-decomposition is one of such techniques. On the other hand, our high-precision method can compute floating-point Gröbner bases stably. Therefore, we will reduce numerically the errors of the results computed by the high-precision method.

Let $\Phi_k = \{F_1^{(k)}, \dots, F_r^{(k)}\}$ and $\Phi_l = \{F_1^{(l)}, \dots, F_s^{(l)}\}$ be intermediate bases appearing in the computation of Gröbner basis by Buchberger's procedure, where Φ_k is a basis computed before Φ_l (Φ_k may be the initial basis). Then, each polynomial $F_i^{(l)}$ in Φ_l can be expressed in terms of the elements of Φ_k, as follows:

$$F_i^{(l)} = a_{i1}F_1^{(k)} + \cdots + a_{ir}F_r^{(k)}. \tag{1}$$

We call the tuple $(a_{i1}, \dots, a_{ir})$ a syzygy for $F_i^{(l)}$. In order to reduce the errors occurred during the computation from Φ_k to Φ_l, we utilize the syzygies.

Let $\mathcal{R}_i$ $(1 \le i \le s)$ and $\mathcal{C}_{ij}$ $(1 \le j \le r)$ be sets of power-products defined as follows.

$$\begin{aligned} &\mathcal{R}_i = \cup_{j=1}^r \mathrm{supp}(a_{ij}F_j^{(k)}) \overset{\text{def}}{=} \{T_1, T_2, \dots, T_{\bar{n}}\}, \quad T_1 \succ T_2 \succ \cdots \succ T_{\bar{n}}, \\ &\mathcal{C}_{ij} = \mathrm{supp}(a_{ij}) \overset{\text{def}}{=} \{S_{ij,1}, S_{ij,2}, \dots, S_{ij,m_j}\}, \quad S_{ij,1} \succ S_{ij,2} \succ \cdots \succ S_{ij,m_j}. \end{aligned} \tag{2}$$

We express $F_i^{(l)}$ and $F_j^{(k)}$ as follows.

$$\begin{aligned} F_i^{(l)} &= f_{i1}^{(l)}T_{i1} + f_{i2}^{(l)}T_{i2} + \cdots, \quad T_{i1} \succ T_{i2} \succ \cdots, \\ F_j^{(k)} &= f_{j1}^{(k)}T_{j1} + f_{j2}^{(k)}T_{j2} + \cdots . \, T_{j1} \succ T_{j2} \succ \cdots . \end{aligned} \tag{3}$$

Since $\{S_{ij,m}T_{j1}, \dots, S_{ij,m}T_{jn_j}\} \subset \{T_1, T_2, \dots, T_{\bar{n}}\}$ for any $j \in \{1, \dots, r\}$ and $m \in \{1, \dots, m_j\}$, (1) assures that we can express the coefficient vector of $F_i^{(l)}$ in terms of a sum of coefficient vectors of $S_{ij,m}F_j^{(k)}$ $(1 \le j \le r;\ 1 \le m \le m_j)$, expanded in power-products $T_1, T_2, \dots, T_{\bar{n}}$.

We put $\bar{m} = m_1 + \cdots + m_j$. We define an $\bar{m} \times \bar{n}$ matrix $\mathcal{M}_i$, which we call *reduction matrix*, as follows.

$$\mathcal{M}_i = \begin{pmatrix} \text{coefficient vector of } S_{i1,1}F_1^{(k)} \\ \quad \text{coefficient vector of } S_{i1,2}F_1^{(k)} \\ \ddots \quad \ddots \quad \ddots \quad \ddots \\ \text{coefficient vector of } S_{i2,1}F_2^{(k)} \\ \quad \text{coefficient vector of } S_{i2,2}F_2^{(k)} \\ \ddots \quad \ddots \quad \ddots \quad \ddots \\ \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \end{pmatrix} \tag{4}$$

Let $\mathcal{M}_i^{(\mathrm{L})}$ be the submatrix of $\mathcal{M}_i$, composed of the columns located in the left side of the power product T_{i1}. We eliminate the columns of $\mathcal{M}_i^{(\mathrm{L})}$, by transforming the rows of $\mathcal{M}_i$ simultaneously. This elimination gives us a row of $\mathcal{M}_i$, of the form $(0, \dots, 0, \star, *, \dots, *)$, where $\star$ is not zero and it stands at T_{i1}, while $*$ may be zero. Then, $(\star, *, \dots, *)$ is a coefficient vector of $F_i^{(l)}$.

Example 3 (Reduction matrix $\mathcal{M}_2$ for G_2 in example 1). The G_2 was computed as $F_1 \xrightarrow{F_2} F_1'$, $F_2 \xrightarrow{F_1'} F_2'$, $F_3 \xrightarrow{F_2'} F_3'$, $F_2' \xrightarrow{F_3'} G_2$. The corresponding syzygy is

($a_{2,1} = -3yz^2 + 3/2z,\ a_{2,2} = 2y^2z^2 - 1/2,\ a_{2,3} = 4/3y$). By this, we can easily obtain the following minimal power-product set for G_2:

$$\mathcal{R}_2 = (x^2y^2z^3,\, x^2yz^2,\, x^2z,\, xy^2z^2,\, x,\, y^3z^3,\, y^2z^2,\, y^2z,\, yz,\, 1)$$

Then, the reduction matrix $\mathcal{M}_2$ for G_2, expressed by F_1, F_2, F_3, is as follows.

$$\left(\begin{array}{r|cccc|cccccc}
 & x^2y^2z^3 & x^2yz^2 & x^2z & xy^2z^2 & x & y^3z^3 & y^2z^2 & y^2z & yz & 1 \\
\hline
yz^2F_1 & 1.0 & 0.5 & & & & & & & & \\
zF_1 & & 1.0 & 0.5 & & & & & & & \\
y^2z^2F_2 & 1.0 & & & -2/3. & & -2/3. & -1/3. & & & \\
1F_2 & & & 1.0 & & -2/3. & & & & -2/3. & -1/3. \\
yF_3 & & & & 3/2. & & & & -1.0 & &
\end{array}\right)$$

Here, $\mathcal{M}_2^{(\mathrm{L})}$ is the matrix composed of the left four columns. □

Remark 1. It is obvious that the rows in $\mathcal{M}_i^{(\mathrm{L})}$ are linearly dependent. However, the dimension of null space may be greater than 1. Even if the dimension is 1, the corresponding vector in $\mathcal{M}_i$ may be different from the coefficient vector of $F_i^{(l)}$, because the matrix reduction is rarely the same as the corresponding reductions in Buchberger's method. □

Our idea solves problem 3). The systematic exact cancellation is analogous to that occurs in naive Gaussian elimination of numerical matrices: if a row of small pivot is used to eliminate columns then there occur large cancellations in subsequent eliminations. We can avoid such cancellations almost completely by pivoting. We show this by Example 4, below. Furthermore, the matrix method mentioned above decreases the risk of accidental cancellations largely.

Example 4 (Reducing the errors in G_1, G_2 in Example 2). Eliminating reduction matrices for G_1, G_2, we obtained the following improved G_1, G_2.

$$\begin{cases} G_1 = \ x + \#\mathrm{E}(1.333333333333333407348201641677\mathrm{e}{-0},\ 2.7\mathrm{e}{-28})\, y \\ G_2 = yz + \#\mathrm{E}(4.999999999999999722444243843710\mathrm{e}{-1},\ 1.0\mathrm{e}{-28})\, x \\ \qquad\quad + \#\mathrm{E}(6.666666666666666666666666666666\mathrm{e}{-1},\ 1.3\mathrm{e}{-28})\, y \\ \qquad\quad + \#\mathrm{E}(5.000000000000000000000000000000\mathrm{e}{-1},\ 1,5\mathrm{e}{-29})\,. \end{cases}$$

Almost no cancellation occurred in the computation of the above G_1, G_2. □

Remark 2. The above G_2 is different from that in Example 2. The matrix $\mathcal{M}_2$ for G_2 is of size 112×88, and 84 columns are eliminated. When the elimination finished, we have 21 nonzero rows (the left 84 elements are zero). These 21 rows are linear combinations of the following two vectors

$$\begin{array}{llllllll}
(0 & \cdots & 0 & 3.30\cdots & 0.00 & 0.00 & 1.65\cdots &), \\
(0 & \cdots & 0 & 0.00 & 3.60\cdots & 4.80\cdots & 0.00 &).
\end{array}$$

These vectors correspond to $yx + 1/2$ and $x + 4/3y$, respectively. Hence, the ideal itself is the same. □

Example 5 (Case of large systematic exact cancellation). We compute an unreduced Gröbner basis w.r.t. the total-degree order, of the following system.

$$\begin{cases} F_1 = x^3/30.0 + x^2y + y^2/3.0, \\ F_2 = x^2y^2/3.0 - xy^2 - xy/3.0, \\ F_3 = y^3/20.0 + x^2. \end{cases}$$

The high-precision method gives us the following unreduced Gröbner basis.

$$\begin{cases} G_2 = \#\mathrm{E}(9.9999999999999999999\cdots\mathrm{e}{-1},\ 1.7\mathrm{e}{-18})\, y^2, \\ G_4 = \#\mathrm{E}(9.9999999999999999999\cdots\mathrm{e}{-1},\ 2.2\mathrm{e}{-19})\, xy \\ \quad + \#\mathrm{E}(8.4402255045219586762 8\cdots\mathrm{e}{-2},\ 1.8\mathrm{e}{-20})\, y^2, \\ G_5 = \#\mathrm{E}(9.9999999999999999999\cdots\mathrm{e}{-1},\ 1.0\mathrm{e}{-28})\, x^2 \\ \quad + \#\mathrm{E}(7.1484968974627070063 9\cdots\mathrm{e}{-0},\ 1.7\mathrm{e}{-18})\, xy \\ \quad + \#\mathrm{E}(5.7371613952464572257 6\cdots\mathrm{e}{-1},\ 1.3\mathrm{e}{-19})\, y^2. \end{cases}$$

We see that there occurred large cancellations of amounts $O(10^{10})$. Applying the matrix method, we obtained the following result (we omit G_2).

$$\begin{cases} G_4 = xy + \#\mathrm{E}(8.4402255045219586762 8\cdots\mathrm{e}{-2},\ 3.7\mathrm{e}{-29})\, y^2, \\ G_5 = x^2 + \#\mathrm{E}(7.1484968974627070063 9\cdots\mathrm{e}{-0},\ 3.6\mathrm{e}{-26})\, xy \\ \qquad + \#\mathrm{E}(5.7371613952464572257 6\cdots\mathrm{e}{-1},\ 2.7\mathrm{e}{-27})\, y^2. \end{cases}$$

We see that the error-parts of the efloats are improved drastically. (The value-parts are the same up to the double-precision, showing that the high-precision method is trusty). □

We next consider intrinsic errors. From the viewpoint of matrix method, the intrinsic errors are regarded as errors occurring in reducing ill-conditioned matrices, hence they cannot be reduced by the matrix method; we need some preconditioning operation to reduce the errors.

Example 6 (Case of intrinsic errors). We compute an unreduced Gröbner basis w.r.t. the total-degree order, of the following system. Note that we have the relation $\|56/57\, yzF_1 - 57/56\, xzF_3 - 2F_2\| = 0.000041$.

$$\begin{cases} F_1 = 57/56\, x^2y + 68/67\, xz^2 - 79/78\, xy + 89/88\, x, \\ F_2 = \qquad\qquad xyz^3 - \qquad xy^2z + \qquad xyz, \\ F_3 = 56/57\, xy^2 - 67/68\, yz^2 + 78/79\, y^2 - 88/89\, y. \end{cases}$$

Using the high-precision method of 30 decimal precision, we found that the following polynomials were generated in the computation.

$$\begin{aligned} G_6 = &\ \#\mathrm{E}(9.9999999999999999999\cdots\mathrm{e}{-1},\ 1.23\mathrm{e}{-23})\, x^2y^2 \\ &- \#\mathrm{E}(2.9954369477325526445 3\cdots\mathrm{e}{-0},\ 3.68\mathrm{e}{-23})\, xy^2 \\ &- \#\mathrm{E}(1.0020782165123748257 6\cdots\mathrm{e}{-0},\ 1.23\mathrm{e}{-23})\, y^3 \\ &+ \#\mathrm{E}(1.9983254691737245140 1\cdots\mathrm{e}{-0},\ 2.45\mathrm{e}{-23})\, xy \\ &+ \#\mathrm{E}(1.0035217172564144753 6\cdots\mathrm{e}{-0},\ 1.23\mathrm{e}{-23})\, y^2, \\ F' = &\ \#\mathrm{E}(9.9999999999999999999\cdots\mathrm{e}{-1},\ 6.33\mathrm{e}{-13})\, xy^2 \\ &- \#\mathrm{E}(568.42904607161639553 8\cdots\mathrm{e}{-0},\ 3.60\mathrm{e}{-10})\, xz^2 \\ &+ \#\mathrm{E}(565.42958527123132600 3\cdots\mathrm{e}{-0},\ 3.58\mathrm{e}{-10})\, xy \\ &- \#\mathrm{E}(566.43422488720734641 9\cdots\mathrm{e}{-0},\ 3.59\mathrm{e}{-10})\, x. \end{aligned}$$

G_6 is changed only a little by the matrix method: the error-parts of the coefficients of G_6 are changed only in the third digits.

The F' does not appear in the final basis, but if we discard F' just when it is generated, the resulting basis becomes very different from the basis over $\mathbb{Q}$. The F' suggests that, once the intrinsic errors of considerable amounts occur, the computation will become unstable. □

We mentioned in **1** that we want to know the amounts of intrinsic errors. If the errors due to the systematic exact cancellations are eliminated completely, then the resulting errors are intrinsic. We can expect that the matrix method will avoid the systematic exact cancellations almost completely. Then, we may say that the matrix method informs us the amounts of intrinsic errors.

5 Actual Implementation and Discussions

In this section, by "quality improvement of a polynomial" we mean reducing the error-parts of the polynomial coefficients by the matrix method.

In examples in **4**, qualities of final polynomials were improved directly from the initial ones. In this approach, we must often handle matrices of very large sizes. For example, in Example 4, we handled matrices of sizes 143×109 and 112×88, and the cost of matrix reduction is about twice of that of the Gröbner basis computation itself. Furthermore, in this approach, we know the amounts of intrinsic errors only at the final stage, where initial accuracies of some polynomials may be lost at all. Therefore, we improve the qualities of polynomials not only in the final but also in intermediate bases.

We divide the whole computation into many stages, the initial stage, the 1-st stage, the 2-nd stage, and so on. In the beginning of the k-th stage, we have a set of starting polynomials $\{F_1^{(k)}, \ldots, F_{r_k}^{(k)}\}$. At the end of the k-th stage, we improve the qualities of all the existing polynomials, and the improved polynomials are used as the starting polynomials of the $(k+1)$-st stage:

$$\Phi_0 = \{F_1^{(0)}, \ldots, F_{r_0}^{(0)}\} \longrightarrow \cdots \longrightarrow \Phi_k = \{F_1^{(k)}, \ldots, F_{r_k}^{(k)}\} \longrightarrow \cdots . \tag{5}$$

Each stage is closed and the next stage begins when a systematic cancellation of amount C_{small} or more is detected or when the cancellations accumulate to C_{med} or more ($C_{\text{small}} = 100$ and $C_{\text{med}} = 1000$ in the current program). The quality improvement is performed by the following two procedures.

- `improvePols`(Φ): this procedure is called at the end of each stage, and it improves the quality of every polynomial in the basis Φ, intermediate or final. Each polynomial is then replaced by the improved one.
- `improvePol`(P): this procedure improves a single polynomial P and checks whether P is actually reduced to 0. This procedure is called when the cancellation of C_{big} or more is detected in $\mathrm{Spol}(P_1, P_2)$ or $P' \xrightarrow{Q}\!\!\!\!\rightarrow P$ ($C_{\text{big}} = 10^6$ in the current program).

One important notice in this approach is that the systematic exact cancellation may not be removed inside a single stage. A typical mechanism of systematic exact cancellation is as follows. Suppose polynomials P_1 and P_2 are reduced by Q the leading term of which is small: $P_i \xrightarrow{Q} \tilde{P}_i$ $(i=1,2)$ (P_1 and P_2 may be reduced by $Q_1, \dots, Q_k$). Then, $P_i \approx -(\mathrm{lt}(P_i)/\mathrm{lt}(Q))\,\mathrm{rt}(\cdots \mathrm{rt}(Q)\cdots)$, and the multiples of $\mathrm{rt}(\cdots \mathrm{rt}(Q)\cdots)$ cancel exactly in subsequent $\mathrm{Spol}(\tilde{P}_1, \tilde{P}_2)$ or $\mathrm{Lred}(\tilde{P}_1, \tilde{P}_2)$. In [14,15], we called $\tilde{P}_1$ and $\tilde{P}_2$ *clones* of Q, and the subsequent cancellation *self-reduction*. Suppose the clones are generated in the k-th stage and the self-reduction of the clones occurs in the l-th stage. If $k = l$ then the quality improvement at the end of the l-th stage will remove the systematic exact cancellation. If, however, $k < l$ then we must back to the k-th stage so as to remove the cancellation occurred in the l-th stage. On the other hand, for the systematic inexact cancellation, we need not back to previous stages.

Another important notice is on how to express polynomials in the l-th stage by the starting polynomials in the k-th stage when $l > k+1$. One may think that we can do this by connecting syzygies in the k-th to l-th stages. This is true if no quality improvement is made in the stages. However, in many cases, the quality improvement changes the polynomial structure; some terms may be missing and some terms may be added by the improvement. Therefore, we compute the polynomials in the l-th stage as follows: first, compute the starting polynomials of the $(k+1)$-st stage by using the syzygies, then compute the polynomial in the l-th stage by applying Buchberger's procedure to the starting polynomials in the $(k+1)$-st stage.

So far, we have tested only a preliminary version which does not back stages. We show some timing data on Lenovo IdeaPad U450p (1.2GHz, 1MB), where we used efloats of 30 decimal precision. Each Gröbner basis computed is the same as that over $\mathbb{Q}$ up to double precision.

(unit: milliseconds)
(datum 1180 for Katsura-4 is by the computation over $\mathbb{Q}$)

sample	high-prec. method	with quality-improve
Example-4	3.6	10.4
Example-5	1.8	7.6
Example-6	2.4	28.0
Katsura-4	[1180 (+320GC)]	824 (+112GC)

Katsura-4 contains 4 polynomials of total-degree 2 and 1 polynomial of total-degree 1, in 5 variables. The Gröbner basis w.r.t. the total-degree order is composed of 16 polynomials of total-degrees 1 to 5.

We found the following features of our method.

1. Gaussian elimination with partial pivoting which selects a row is often fail to reduce the errors well, but the full pivoting which selects the maximum magnitude element works well. We also tested matrix reduction by the Givens rotation, and found that this reduction method is much more expensive than the Gaussian elimination.

2. Procedure `improvePol`(P) is quite effective for detecting whether P should be regarded as 0.
3. If a considerable amounts of intrinsic errors occur on a polynomial P, then the errors propagate to other polynomials via $\mathrm{Spol}(P, P')$ and $F \xrightarrow{P}\!\!\!\!\rightarrow \tilde{F}$, and the computation becomes unstable soon. (In Katsura-4, we encountered no intrinsic cancellation.)

We must comment on the above point 3. One may think that the point 3. implies a severe limitation of our method. Our opinion is completely different. We note that we have tested polynomials of limited accuracies: we have converted the coefficients of given polynomials into double-precision floating-point numbers, which introduces $O(10^{-16})$ relative errors, then converted them into efloats of 30 decimal precision. When handling polynomials of limited accuracies, the bases computed have no meaning if the initial accuracies are lost during the computation. Therefore, in order to obtain some meaningful results, we must discard polynomials the accuracies of which were lost largely. That is, we must define "approximate Gröbner bases" so that the polynomials of very low accuracies are discarded by some criteria derived theoretically.

In [14], we have tried to define approximate Gröbner bases by using syzygies. However, the definition in [14] is too immature. We explain this by considering G_2 in Example 2. The syzygy computed by Buchberger's procedure contains a term of magnitude $O(10^4)$, which does not mean that the intrinsic cancellations of magnitude $O(10^4)$ occur in the computation of G_2. In fact, Example 4 shows that no intrinsic cancellation occurred on G_2. Therefore, we must define approximate Gröbner bases without using syzygies. We are now trying to define approximate Gröbner bases more appropriately.

References

1. Bodrato, M., Zanoni, A.: Intervals, syzygies, numerical Gröbner bases: a mixed study. In: Ganzha, V.G., Mayr, E.W., Vorozhtsov, E.V. (eds.) CASC 2006. LNCS, vol. 4194, pp. 64–76. Springer, Heidelberg (2006)
2. Chen, Y., Meng, X.: Border bases of positive dimensional polynomial ideals. In: Proceedings of SNC 2007, Symbolic Numeric Computation, London, Canada, pp. 65–71 (2007)
3. Fortuna, E., Gianni, P., Trager, B.: Degree reduction under specialization. J. Pure Appl. Algebra 164, 153–164 (2001)
4. Gonzalez-Vega, L., Traverso, C., Zanoni, A.: Hilbert stratification and parametric Gröbner bases. In: Ganzha, V.G., Mayr, E.W., Vorozhtsov, E.V. (eds.) CASC 2005. LNCS, vol. 3718, pp. 220–235. Springer, Heidelberg (2005)
5. Kreuzer, M., Kehrein, A.: Computing border bases. J. Pure Appl. Alg. 200, 279–295 (2006)
6. Kalkbrener, M.: On the stability of Gröbner bases under specialization. J. Symb. Comput. 24, 51–58 (1997)
7. Kako, F., Sasaki, T.: Proposal of "effective" floating-point number. Preprint of Univ. Tsukuba (May 1997) (unpublished)

8. Kondratyev, A., Stetter, H.J., Winkler, S.: Numerical computation of Gröbner bases. In: Proceedings of CASC 2004, Computer Algebra in Scientific Computing, St. Petersburg, Russia, pp. 295–306 (2004)
9. Mourrain, B.: A new criterion for normal form algorithms. LNCS, vol. 179, pp. 430–443. Springer, Heidelberg (1999)
10. Mourrain, B.: Pythagore's dilemma, symbolic-numeric computation, and the border basis method. In: Symbolic-Numeric Computations, Trends in Mathematics, pp. 223–243. Birkhäuser Verlag, Basel (2007)
11. Nagasaka, K.: A study on gröbner basis with inexact input. In: Gerdt, V.P., Mayr, E.W., Vorozhtsov, E.V. (eds.) CASC 2009. LNCS, vol. 5743, pp. 248–258. Springer, Heidelberg (2009)
12. Sasaki, T.: A practical method for floating-point Gröbner basis computation. In: Proceedings of ASCM 2009, Asian Symposium on Computer Mathematics, Fukuoka, Japan, Math-for-industry series, vol. 22, pp. 167–176. Kyushu Univ. (2009)
13. Sasaki, T.: A subresultant-like theory for Buchberger's procedure, 17 p. Preprint of Univ. Tsukuba (March 2010)
14. Sasaki, T., Kako, F.: Computing floating-point Gröbner base stably. In: Proceedings of SNC 2007, Symbolic Numeric Computation, London, Canada, pp. 180–189 (2007)
15. Sasaki, T., Kako, F.: Floating-point Gröbner basis computation with ill-conditionedness estimation. In: Kapur, D. (ed.) ASCM 2007. LNCS (LNAI), vol. 5081, pp. 278–292. Springer, Heidelberg (2008)
16. Shirayanagi, K.: An algorithm to compute floating-point Gröbner bases. In: Mathematical Computation with Maple V. Ideas and Applications, pp. 95–106. Birkhäuser, Basel (1993)
17. Shirayanagi, K.: Floating point Gröbner bases. Mathematics and Computers in Simulation 42, 509–528 (1996)
18. Shirayanagi, K., Sweedler, M.: Remarks on automatic algorithm stabilization. J. Symb. Comput. 26, 761–765 (1998)
19. Stetter, H.J.: Stabilization of polynomial systems solving with Gröbner bases. In: Proceedings of ISSAC 1997, Intern'l Symposium on Symbolic and Algebraic Computation, pp. 117–124. ACM Press, New York (1997)
20. Stetter, H.J.: Numerical Polynomial Algebra. SIAM Publ., Philadelphia (2004)
21. Stetter, H.J.: Approximate Gröbner bases – an impossible concept? In: Proceedings of SNC 2005, Symbolic-Numeric Computation, Xi'an, China, pp. 235–236 (2005)
22. Suzuki, A.: Computing Gröbner bases within linear algebra. In: Gerdt, V.P., Mayr, E.W., Vorozhtsov, E.V. (eds.) CASC 2009. LNCS, vol. 5743, pp. 310–321. Springer, Heidelberg (2009)
23. Traverso, C.: Syzygies, and the stabilization of numerical Buchberger algorithm. In: Proceedings of LMCS 2002, Logic, Mathematics and Computer Science, RISC-Linz, Austria, pp. 244–255 (2002)
24. Traverso, C., Zanoni, A.: Numerical stability and stabilization of Gröbner basis computation. In: Proceedings of ISSAC 2002, Intern'l Symposium on Symbolic and Algebraic Computation, pp. 262–269. ACM Press, New York (2002)
25. Weispfenning, V.: Gröbner bases for inexact input data. In: Proceedings of CASC 2003, Computer Algebra in Scientific Computing, Passau, Germany, pp. 403–411 (2003)

One Class of Third-Order Linear ODE's

S.Yu. Slavyanov

St. Petersburg State University, Dept. of Comput.Phys.
slav@SS2034.spb.edu

Abstract. A classification of equations originated by Fuchsian third-order equation with three regular points is proposed. Links to generalized hypergeometric equation are discussed.

Keywords: Fuchsian equation, third-order ODE's, polynomial coefficients.

Introduction

The aim of this presentation is to give a classification of linear homogeneous third-order equations arising as a result of confluence or reduction from the Fuchsian third-order equation with three regular singularities. The merging of two singularities accompanied by limiting transform in the space of parameters is meant by confluence. Specification of parameters resulted in change of solution characteristics is meant by reduction. The set of the equations arising from the above-mentioned Fuchsian equation comprises the class M_3^3 according to [1]. Different types of equations belonging to M_3^3 are distinguished. Relation to generalized hypergeometric functions ${}_pF_q$ is discussed.

Our study is based on the generalization of the notion of the s-rank of a singularity and of the s-multisymbol of an equation proposed in [2] and [3] for the second-order equations.

The Fuchsian third-order equation with two regular singularities at $z_1 = 0$, $z_2 = 1$ and a regular singularity at $z_3 = \infty$ can be exposed with the help of its symbol – a polynomial T^C in two variables: z and D

$$T^C(z, D) := (\zeta)^3 D^3 + \zeta^2 P_1(z) D^2 + \zeta P_2(z) D + P_3(z), \tag{1}$$

where D is the differentiation operator, and z is the operator of multiplication by independent variable, $\zeta = z(1 - z)$, P_k are polynomials of degrees k.

The following well-known lemma holds.

Lemma. *There exists a transform of the dependent variable converting the canonical form T^C of a Fuchsian equation into normal form with the symbol*

$$T^N(z, D) := S_U(T^C) = \zeta^3 D^3 + \zeta Q_2(z) D + Q_3(z), \tag{2}$$

where ζ^3 has triple roots in two regular singularities $0, 1$.

Practical classification of singularities of a third-order equation with polynomial coefficients can be constructed on the basis of the notion of the s-rank of the singularity. At analytical study of roots of symbolic characteristic equation

$$T^N(z, D) = 0. \tag{3}$$

its solutions can be presented in the vicinity of a singularity as Puiseux series of the form

$$D_m(z_j) = C_{mj}(z - z_j)^{-\mu_j} \sum_{k=0}^{\infty} h_k (z - z_j)^{k\sigma},$$
$$C_{mj} \neq 0, m = 1, 2, 3, \quad j = 1, 2, 3$$
$$D_m(\infty) = C_{m\infty} z^{\mu_\infty - 2} \sum_{k=0}^{\infty} h_k z^{-k\sigma}, \qquad C_{m\infty} \neq 0.$$

with integer or fractional μ_j satisfying

$$\frac{1}{3} \leq \mu_j \leq 3 \tag{4}$$

and σ_j taking the values $1, 1/2$, and $1/3$. The first term of these equations determines the behavior of the logarithmic derivative of solutions of differential equation related to the symbol (1) in the vicinity of the corresponding singularity.

$$|\ln w(z)| \leq K|z - z_j|^{-\mu_j - \epsilon} \qquad \text{for} \quad \mu_j \geq 1 \quad \epsilon \geq 0; \tag{5}$$

with corresponding positive constant K determined by coefficients of polynomials in (1) (see also [3] page 9).

It is important that $\mu_j = 1$ for generating regular singularities.

Definition. The set of numbers $\{\mu_1, \mu_2, \mu_\infty\}$ is termed the *s-multisymbol* of the studied equation. Two equations belong to one type if and only if they have the same s-multisymbol.

A Fuchsian equation with n regular singularities is characterized by s-multisymbol consisting of n unities. We keep unity as the s-rank for regular singularity below. However, at confluence of two singularities the s-multisymbol changes decreasing by one element while two corresponding μ_j generate a new element numerically equal or less to the sum of original elements. The possible numerical value can be either integer or half-integer or be multiple of $1/3$.

Starting from a Fuchsian equation with n regular singularities, the following question arises: how many different types of equations in the frame of proposed classification (that is having different s-multisymbols) can be obtained by means of confluence and specialization of coefficients.

The following theorem holds for second-order equation [5].

Theorem. *The number of different types of equations, which can be obtained by confluence and reduction of Fuchsian second order equation with n regular singularities at $n \geq 3$, according chosen classification in terms of s-multisymbol is equal to nth coefficient of Taylor series at zero for the function*

$$\exp \sum_{j>0} \frac{2z^j}{j(1 - z^j)} =$$
$$1 + 2z + 5z^2 + 10z^3 + 20z^4 + 36z^5 + 65z^6 + 110z^7 + 185z^8 + O(z^9).$$

Our object in this publication is to count the number of types of equations and to expose the equations themselves and their multisymbols for the class M_3^3.

Classification

The initial equation is the Fuchsian equation with three regular singularities the solutions of which depend on 7 parameters – six singular parameters and one accessory parameter.

$$\{1,1,1\} \quad [z^3(1-z)^3D^3 + z(1-z)(p_2z^2 + p_1z + p_0)D + (q_3z^3 + q_2z^2 + q_1z + q_0)]y = 0. \tag{6}$$

By *singular parameters* are meant those parameters, which characterize local solutions of the equation in the vicinity of a regular singularity namely data of lateral connection problem. Because of the Fuchs theorem the number of parameters corresponding to infinity $z_3 = \infty$ is equal to two as in the case of singularities $z_1 = 0$, $z_2 = 1$. The relation between characteristic exponents $\rho_m(z_j)$, $m = 1,2,3$ and the coefficients of equation can be obtained from the third-order algebraic characteristic equations

$$\begin{gathered}\rho(z_1)(\rho(z_1) - 1)(\rho(z_1 - 2) + p_0\rho(z_1) + q_0 = 0,\\ \rho(z_3)(\rho(z_3) + 1)(\rho(z_3) + 2) + p_2\rho(z_3) + q_3 = 0,\\ \rho(z_2)(\rho(z_2) - 1)(\rho(z_2) - 2) + \rho(z_2)\sum_{i=0}^{2} p_i + \sum_{i=0}^{3} q_i = 0.\end{gathered} \tag{7}$$

The *accessory parameter* determines the solution of global central connection problem which is part of general monodromy data. Since the sum $q_1 + q_2$ is fixed in (7) the following combination can be taken

$$\theta = q_1 - q_2 \tag{8}$$

as the accessory parameter. Thus, $p_0, p_1, p_2, q_0.q_3, q_0 + q_1 + q_2 + q_3$ are singular parameters of (6). The Fuchs relation for (6) reads

$$\sum_{j=1}^{3}\sum_{m=1}^{3} \rho_m(z_j) = 3. \tag{9}$$

The confluent equation is originated by (6) at limiting confluence process when the regular singularity z_2 tends to infinity in (6) accompanied by appropriate tending to infinity of the characteristic exponents at this singularity. Solutions depend upon 6 parameters - five singular parameters and one accessory parameter.

$$\{1,2\} \quad [z^3D^3 + z(z^2 + p_1z + p_0)D + (q_3z^3 + q_2z^2 + q_1z + q_0)]y = 0. \tag{10}$$

The characteristic equation at $z_1 = 0$ is the same as in (7)

$$\rho(z_1)(\rho(z_1) - 1)(\rho(z_1 - 2) + p_0\rho(z_1) + q_0 = 0.$$

However, at infinity the behavior of solutions would be

$$y_m(z) \sim \exp(\kappa_m z) z^{-\alpha_m} \sum_{n=0}^{\infty} c_{mn} z^{-n}. \tag{11}$$

That leads to characteristic equations for κ_m and α_m

$$\begin{gathered} \kappa^3 + \kappa + q_3 = 0, \\ -\alpha(3\kappa^2 + 1) + p_1\kappa + q_2 = 0. \end{gathered} \tag{12}$$

As the result of equalities

$$\kappa_1 + \kappa_2 + \kappa_3 = 0, \quad \kappa_1\kappa_2 + \kappa_1\kappa_3 + \kappa_2\kappa_3 = 1, \tag{13}$$

which in their turn follow from (12), it holds

$$\alpha_1 + \alpha_2 + \alpha_3 = 0 \tag{14}$$

and

$$\sum_{m=1}^{3} \rho_m(0) + \sum_{m=1}^{3} \alpha_m = 3. \tag{15}$$

This is an extension of the Fuchs theorem formulated in terms of s-rank. Namely, the sum of appropriate characteristic exponents over all singularities is equal to sum of all s-ranks. This statement can be regarded as a possible conjecture.

Under the process of confluence when the regular singularity $z_1 = 0$ in (10) is driven to infinity with corresponding changes in characteristic exponents we arrive at the double confluent equation. Solutions of it depend upon five parameters – four singular parameters and one accessory parameter.

$$\{3\} \quad [D^3 + (-z^2 + p_0)D + q_3 z^3 + q_2 z^2 + q_1 z + q_0]y = 0. \tag{16}$$

The behavior of solutions at irregular point $z = \infty$ with the s-rank equal to 3 is determined by asymptotics

$$y(z) \sim \exp(\kappa z^2/2 + \gamma z) z^{-\alpha}. \tag{17}$$

The characteristic exponents ρ_m, γ_m and α_m satisfy the following characteristic equations

$$\begin{aligned} &\kappa^3 - \kappa + q_3 = 0, \\ &\gamma(3\kappa^2 - 1) + q_2 = 0, \\ &\alpha(3\kappa^2 - 1) = 3\gamma^2\kappa + \kappa p_0 + q_1. \end{aligned} \tag{18}$$

From the first of these equations it follows that

$$\sum_{i=1}^{m} \kappa_i = 0, \quad \sum_{i=1}^{3} k_i^2 = 1, \quad \sum_{i=1, l=1, i \neq l}^{3} \kappa_i \kappa_l = -1. \tag{19}$$

Several other equalities can be derived from (18) and (19). Manipulations with MAPLE lead to one more equality

$$\sum_{i=1}^{3} \gamma_i = 0. \tag{20}$$

However, we were unable to prove that

$$\sum_{i=1}^{3} \alpha_i = 3 \tag{21}$$

what corresponds to our conjecture.

Several reduced confluent equations (with non-integer s-rank) can also be studied. These are

$$\begin{aligned}
&\{1, \frac{5}{3}\} \quad [z^3 D^3 + p_0 z D + (z^2 + q_1 z + q_0)]y = 0 \\
&\{1, \frac{3}{2}\} \quad [z^3 D^3 + z(z + p_0) D + (q_1 z + q_0)]y = 0 \\
&\{1, \frac{4}{3}\} \quad [z^3 D^3 + p_0 z D + (z + q_0)]y = 0 \\
&\{\frac{8}{3}\} \quad [D^3 + (p_1 z + p_0) D + z^2 + q_1 z]y = 0 \\
&\{\frac{5}{2}\} \quad [D^3 + (-z + p_0) D + q_1 z]y = 0 \\
&\{\frac{7}{3}\} \quad [D^3 + p_0 D + z]y = 0.
\end{aligned}$$

Their detailed investigation is beyond the frame of this presentation.

Application to Generalized Hypergeometric Equation

The generalized hypergeometric equation in particular cases is also a third-order equation either it is a Fuchsian equation or confluent equation or double confluent equation. The main difference lays in absence of accessory parameters. The generalized hypergeometric function reads as

$${}_pF_q\left(\begin{matrix}\alpha_1, \ldots, \alpha_p; z \\ \rho_1, \ldots, \rho_q\end{matrix}\right) =_p F_q(\alpha_r; \rho_t; z) = \sum_{n=0}^{\infty} \frac{(\alpha_1)_n \ldots (\alpha_p)_n}{(\rho_1)_n \ldots (\rho_q)_n} \frac{z^n}{n!} \tag{22}$$

with

$$(\alpha)_0 = 1, \quad (\alpha)_n = \frac{\Gamma(\alpha + n)}{\Gamma(\alpha)}.$$

Let $\delta = z\frac{d}{dz}$, then $u =_p F_q$ satisfies ODE:

$$[\delta(\delta + \rho_1 - 1) \ldots (\delta + \rho_q - 1) - z(\delta + \alpha_1) \ldots (\delta + \alpha_p)]u = 0. \tag{23}$$

If $p = 3$, $q = 2$ then equation (23) belongs to M_3^3.

The equivalent equation reads:

$$z^2(z-1)D^3v + \sum_{n=1}^{2} z^{n-1}(a_n z - b_n)D^n v + a_0 v = 0, \tag{24}$$

where $D = \frac{d}{dz}$, and a_n, b_n are constants.

The substitution

$$v = e^{(-\int \frac{a}{3}dz)}u$$

transforms the equation $\left[D^3 + a(z)D^2 + b(z)D + c(z)\right]v = 0$ to the following

$$\Big[D^3 + (-\frac{a(z)^2}{3} - a'(z) + b(z))D +$$

$$(-\frac{a''(z)}{3} + \frac{2}{27}a(z)^3 - \frac{a(z)b(z)}{3} + c(z))\Big]u = 0,$$

where

$$a(z) = \frac{a_2 z - b_2}{z(z-1)}, \quad b(z) = \frac{a_1 z - b_1}{z^2(z-1)}, \quad c(z) = \frac{a_0}{z^2(z-1)},$$

which is the particular case of (6). In the same way the cases with other p and q namely $p = 3$, $q + 1$ and $p = 3$, $q = 0$ can be examined.

Conclusion

The question which arises in relation with proposed classification is the following. Are some of the listed equations related to one another by integral transforms, especially for particular values of parameters? This complicated problem can be solved by tools of Computer Algebra.

The possible particular solutions in terms of second-order equation can be found by methods developed in [6]. For those equations which have only one accessory parameter an open problem is also the existence of the corresponding Painlevé equation.

References

1. Akopyan, A.M., Pirozhnikov, A.V., Slavyanov, S.Y., Zolotarev, V.I.: Elements of data base on special functions. In: Conference: Theoretical, Applied and Computational Celestial Mechanics, ITA RAN, St.-Petersburg (1993)
2. Seeger, A., Lay, W., Slavyanov, S.Y.: Confluence of Fuchsian second-order differential equations. Theor. and Math. Phys. 104(2), 233–247 (1995)
3. Slavyanov, S.Y., Lay, W.: Special Functions: a Unified Theory Based on Singularities. Oxford University Press, Oxford (2000)
4. Slavyanov, S.Y., Lay, W., Seeger, A.: Classification. In: Ronveaux, A. (ed.) Heun's Differential Equation. Oxford University Press, Oxford (1995)
5. Salvy, B., Slavyanov, S.Y.: A combinatorial problem in the classification of second-order linear ODE's, INRIA, Report RR-2600 (1995)
6. Hoeij, M.: Solving third order linear differential equations in terms of second order equations. In: ISSAC 2007 Proc., pp. 355–360 (2007)

GPGCD, an Iterative Method for Calculating Approximate GCD, for Multiple Univariate Polynomials

Akira Terui

Graduate School of Pure and Applied Sciences
University of Tsukuba
Tsukuba, 305-8571, Japan
terui@math.tsukuba.ac.jp

Abstract. We present an extension of our GPGCD method, an iterative method for calculating approximate greatest common divisor (GCD) of univariate polynomials, to multiple polynomial inputs. For a given pair of polynomials and a degree, our algorithm finds a pair of polynomials which has a GCD of the given degree and whose coefficients are perturbed from those in the original inputs, making the perturbations as small as possible, along with the GCD. In our GPGCD method, the problem of approximate GCD is transferred to a constrained minimization problem, then solved with the so-called modified Newton method, which is a generalization of the gradient-projection method, by searching the solution iteratively. In this paper, we extend our method to accept more than two polynomials with the real coefficients as an input.

1 Introduction

For algebraic computations on polynomials and matrices, approximate algebraic algorithms are attracting broad range of attentions recently. These algorithms take inputs with some "noise" such as polynomials with floating-point number coefficients with rounding errors, or more practical errors such as measurement errors, then, with minimal changes on the inputs, seek a meaningful answer that reflect desired property of the input, such as a common factor of a given degree. By this characteristic, approximate algebraic algorithms are expected to be applicable to more wide range of problems, especially those to which exact algebraic algorithms were not applicable.

As an approximate algebraic algorithm, we consider calculating the approximate greatest common divisor (GCD) of univariate polynomials, such that, for a given pair of polynomials and a degree d, finding a pair of polynomials which has a GCD of degree d and whose coefficients are perturbations from those in the original inputs, with making the perturbations as small as possible, along with the GCD. This problem has been extensively studied with various approaches including the Euclidean method on the polynomial remainder sequence (PRS) ([1], [2], [3]), the singular value decomposition (SVD) of the Sylvester matrix ([4],

[5]), the QR factorization of the Sylvester matrix or its displacements ([6], [7], [8]), Padé approximation [9], optimization strategies ([10], [11], [12], [13], [14]). Furthermore, stable methods for ill-conditioned problems have been discussed ([6], [15], [16]).

Among methods in the above, we focus our attention on optimization strategies. Already proposed algorithms utilize iterative methods including the Levenberg-Marquardt method [10], the Gauss-Newton method [14] and the structured total least norm (STLN) method ([11], [12]). Among them, STLN-based methods have shown good performance calculating approximate GCD with sufficiently small perturbations efficiently.

In this paper, we discuss an extension of the GPGCD method, proposed by the present author ([17], [21]), an iterative method with transferring the original approximate GCD problem into a constrained optimization problem, then solving it by the so-called modified Newton method [18], which is a generalization of the gradient-projection method [19]. In the previous papers ([17], [21]), we have shown that our method calculates approximate GCD with perturbations as small as those calculated by the STLN-based methods and with significantly better efficiency than theirs. While our previous methods accept two polynomials with the real or the complex coefficients as inputs and outputs, respectively, we extend it to handle more than two polynomial inputs with the real coefficients in this paper.

The rest part of the paper is organized as follows. In Section 2, we transform the approximate GCD problem into a constrained minimization problem for the case with the complex coefficients. In Section 3, we show details for calculating the approximate GCD, with discussing issues in minimizations. In Section 4, we demonstrate performance of our algorithm with experiments.

2 Formulation of the Approximate GCD Problem

Let $P_1(x), \dots, P_n(x)$ be real univariate polynomials of degree $d_1, \dots, d_n$, respectively, given as

$$P_i(x) = p_{d_i}^{(i)} x^{d_i} + \cdots p_1^{(i)} x + p_0^{(i)},$$

for $i = 1, \dots, n$, with $\min\{d_1, \dots, d_n\} > 0$. We permit P_i and P_j be relatively prime for any $i \neq j$ in general. For a given integer d satisfying $\min\{d_1, \dots, d_n\} > d > 0$, let us calculate a deformation of $P_1(x), \dots, P_n(x)$ in the form of

$$\tilde{P}_i(x) = P_i(x) + \Delta P_i(x) = H(x) \cdot \bar{P}_i(x),$$

where $\Delta P_i(x)$ is a real polynomial whose degrees do not exceed d_i, respectively, $H(x)$ is a polynomial of degree d, and $\bar{P}_i(x)$ and $\bar{P}_j(x)$ are pairwise relatively prime for any $i \neq j$. In this situation, $H(x)$ is an approximate GCD of $P_1(x), \dots, P_n(x)$. For a given d, we try to minimize $\|\Delta P_1(x)\|_2^2 + \cdots + \|\Delta P_n(x)\|_2^2$, the norm of the deformations.

For a real univariate polynomial $P(x)$ represented as $P(x) = p_n x^n + \cdots + p_0 x^0$, let $C_k(P)$ be a real $(n+k, k+1)$ matrix defined as

$$C_k(P) = \underbrace{\begin{pmatrix} p_n & & \\ \vdots & \ddots & \\ p_0 & & p_n \\ & \ddots & \vdots \\ & & p_0 \end{pmatrix}}_{k+1},$$

and let $\boldsymbol{p}$ be the coefficient vector of $P(x)$ defined as

$$\boldsymbol{p} = (p_n, \ldots, p_0). \tag{1}$$

In this paper, for a generalized Sylvester matrix, we use a formulation by Rupprecht [20, Sect. 3]. Then, a generalized Sylvester matrix for $P_1, \ldots, P_n$ becomes as

$$N(P_1, \ldots, P_n) = \begin{pmatrix} C_{d_1-1}(P_2) & C_{d_2-1}(P_1) & \mathbf{0} & \cdots & \mathbf{0} \\ C_{d_1-1}(P_3) & \mathbf{0} & C_{d_3-1}(P_1) & \cdots & \mathbf{0} \\ \vdots & \vdots & & \ddots & \vdots \\ C_{d_1-1}(P_n) & \mathbf{0} & \cdots & \mathbf{0} & C_{d_n-1}(P_1) \end{pmatrix}, \tag{2}$$

and the k-th subresultant matrix (with $\min\{d_1, \ldots, d_n\} > k \geq 0$) is also defined similarly as

$$N_k(P_1, \ldots, P_n) = \begin{pmatrix} C_{d_1-1-k}(P_2) & C_{d_2-1-k}(P_1) & \mathbf{0} & \cdots & \mathbf{0} \\ C_{d_1-1-k}(P_3) & \mathbf{0} & C_{d_3-1-k}(P_1) & \cdots & \mathbf{0} \\ \vdots & \vdots & & \ddots & \vdots \\ C_{d_1-1-k}(P_n) & \mathbf{0} & \cdots & \mathbf{0} & C_{d_n-1-k}(P_1) \end{pmatrix}, \tag{3}$$

with

$$r_k = d_1 + d_2 + \cdots + d_n - (n-1)k + (n-2)d_1 \tag{4}$$

rows and

$$c_k = d_1 + d_2 + \cdots + d_n - n \cdot k \tag{5}$$

columns.

Calculation of GCD is based on the following fact.

Proposition 1 (Rupprecht [20, Proposition 3.1]). *$N_k(P_1, \ldots, P_n)$ has full rank if and only if* $\deg(\gcd(P_1, \ldots, P_n)) \leq k$.

Thus, for a given degree d, if $N_{d-1}(\tilde{P}_1, \ldots, \tilde{P}_n)$ is rank-deficient, then there exist real univariate polynomials $U_1(x), \ldots, U_n(x)$ of degree at most $d_1 - d, \ldots, d_n - d$, respectively, satisfying

$$U_1 \tilde{P}_i + U_i \tilde{P}_1 = 0, \tag{6}$$

for $i = 2, \dots, n$. In such a case, if U_i and U_j are pairwise relatively prime for any $i \neq j$, then $H = \frac{\tilde{P}_1}{U_1} = -\frac{\tilde{P}_2}{U_2} = \cdots = -\frac{\tilde{P}_n}{U_n}$ becomes the expected GCD. Therefore, for given polynomials $P_1, \dots, P_n$ and a degree d, our problem is to find perturbations $\Delta P_1, \dots, \Delta P_n$ along with cofactors $U_1, \dots, U_n$ satisfying (6) with making $\|\Delta P_1(x)\|_2^2 + \cdots + \|\Delta P_n(x)\|_2^2$ as small as possible.

By representing $\tilde{P}_i(x)$ and $U_i(x)$ as

$$\begin{aligned} \tilde{P}_i(x) &= \tilde{p}_{d_i}^{(i)} x^{d_i} + \cdots + \tilde{p}_1^{(i)} x + \tilde{p}_0^{(i)}, \\ U_i(x) &= u_{d_i-d}^{(i)} x^{d_i-d} + \cdots + u_1^{(i)} x + u_0^{(i)}, \end{aligned} \tag{7}$$

we express the objective function and the constraint as follows. For the objective function, $\|\Delta P_1(x)\|_2^2 + \cdots + \|\Delta P_n(x)\|_2^2$ becomes as

$$\|\Delta P_1(x)\|_2^2 + \cdots + \|\Delta P_n(x)\|_2^2 = \sum_{i=1}^{n} \left\{ \sum_{j=0}^{d_i} \left(\tilde{p}_j^{(i)} - p_j^{(i)} \right)^2 \right\}. \tag{8}$$

For the constraint, (6) becomes as

$$N_{d-1}(\tilde{P}_1, \dots, \tilde{P}_n) \cdot {}^t(\boldsymbol{u}_1, \dots, \boldsymbol{u}_n) = \boldsymbol{0}, \tag{9}$$

where $\boldsymbol{u}_i$ is the coefficient vector of $U_i(x)$ defined as in (1). Furthermore, we add another constraint for the coefficient of $U_i(x)$ as

$$\|U_1\|_2^2 + \cdots + \|U_n\|_2^2 = 1, \tag{10}$$

which can be represented together with (9) as

$$\begin{pmatrix} \boldsymbol{u}_1 & \cdots & \boldsymbol{u}_n & -1 \\ \multicolumn{3}{c}{N_{d-1}(\tilde{P}_1, \dots, \tilde{P}_n)} & \boldsymbol{0} \end{pmatrix} \cdot {}^t(\boldsymbol{u}_1, \dots, \boldsymbol{u}_n, 1) = \boldsymbol{0}, \tag{11}$$

where (10) has been put on the top of (9). Note that, in (11), we have total of

$$\bar{d} = d_1 + \cdots + d_n - (n-1)(d-1) + (n-2)d_1 + 1 \tag{12}$$

equations in the coefficients of polynomials in (7) as a constraint, with the j-th row of which is expressed as $g_j = 0$.

Now, we substitute the variables

$$(\tilde{p}_{d_1}^{(1)}, \dots, \tilde{p}_0^{(1)}, \dots, \tilde{p}_{d_n}^{(n)}, \dots, \tilde{p}_0^{(n)}, u_{d_1-d}^{(1)}, \dots, u_0^{(1)}, \dots, u_{d_n-d}^{(n)}, \dots, u_0^{(n)}), \tag{13}$$

as $\boldsymbol{x} = (x_1, \dots, x_{2(d_1+\cdots+d_n)+(2-d)n})$, then (8) and (11) become as

$$\begin{aligned} f(\boldsymbol{x}) = {} & (x_1 - p_{d_1}^{(1)})^2 + \cdots + (x_{d_1} - p_0^{(1)})^2 + \cdots \\ & \cdots + (x_{d_1+\cdots+d_{n-1}+n} - p_{d_n}^{(n)})^2 + \cdots + (x_{d_1+\cdots+d_{n-1}+d_n+n} - p_0^{(n)})^2, \end{aligned} \tag{14}$$

$$\boldsymbol{g}(\boldsymbol{x}) = {}^t(g_1(\boldsymbol{x}), \dots, g_{\bar{d}}(\boldsymbol{x})) = \boldsymbol{0}, \tag{15}$$

respectively, where $\bar{d}$ in (15) is defined as in (12). Therefore, the problem of finding an approximate GCD can be formulated as a constrained minimization problem of finding a minimizer of the objective function $f(\boldsymbol{x})$ in (14), subject to $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}$ in (15).

3 The Algorithm for Approximate GCD

We calculate an approximate GCD by solving the constrained minimization problem (14), (15) with the gradient projection method by Rosen [19] (whose initials become the name of our GPGCD method) or the modified Newton method by Tanabe [18] (for review, see the author's previous paper [17]). Our preceding experiments ([17, Sect. 5.1], [21, Sect. 4]) have shown that the modified Newton method was more efficient than the original gradient projection method while the both methods have shown almost the same convergence property, thus we adopt the modified Newton method in this paper.

In applying the modified Newton method to the approximate GCD problem, we discuss issues in the construction of the algorithm in detail, such as

- Representation of the Jacobian matrix $J_{\boldsymbol{g}}(\boldsymbol{x})$ and certifying that $J_{\boldsymbol{g}}(\boldsymbol{x})$ has full rank (Sect. 3.1),
- Setting the initial values (Sect. 3.2),
- Regarding the minimization problem as the minimum distance problem (Sect. 3.3),
- Calculating the actual GCD and correcting the coefficients of $\tilde{P}_i$ (Sect. 3.4),

as follows.

3.1 Representation and the Rank of the Jacobian Matrix

By the definition of the constraint (15), we have the Jacobian matrix $J_{\boldsymbol{g}}(\boldsymbol{x})$ (with the original notation of variables (13) for $\boldsymbol{x}$) as

$$
\begin{aligned}
J_{\boldsymbol{g}}(\boldsymbol{x}) = & \left(\begin{matrix}
\mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\
C_{d_1}(U_2) & C_{d_2}(U_1) & \mathbf{0} & \cdots & \mathbf{0} \\
C_{d_1}(U_3) & \mathbf{0} & C_{d_3}(U_1) & & \mathbf{0} \\
\vdots & \vdots & & \ddots & \vdots \\
C_{d_1}(U_n) & \mathbf{0} & \cdots & \mathbf{0} & C_{d_n}(U_1)
\end{matrix} \right. \\
& \qquad \left. \begin{matrix}
2 \cdot {}^t\boldsymbol{u}_1 & 2 \cdot {}^t\boldsymbol{u}_2 & 2 \cdot {}^t\boldsymbol{u}_3 & \cdots & 2 \cdot {}^t\boldsymbol{u}_n \\
C_{d_1-d}(P_2) & C_{d_2-d}(P_1) & \mathbf{0} & \cdots & \mathbf{0} \\
C_{d_1-d}(P_3) & \mathbf{0} & C_{d_3-d}(P_1) & & \mathbf{0} \\
\vdots & \vdots & & \ddots & \vdots \\
C_{d_1-d}(P_n) & \mathbf{0} & \cdots & \mathbf{0} & C_{d_n-d}(P_1)
\end{matrix} \right),
\end{aligned}
$$

which can easily be constructed in every iteration. Note that the number of rows in $J_{\boldsymbol{g}}(\boldsymbol{x})$ is equal to $\bar{d}$ in (12), which is equal to the number of constraints, while the number of columns is equal to $2(d_1 + \cdots + d_n) + (2 - d)n$, which is equal to the number of variables (see (13)).

In executing iterations, we need to keep that $J_{\boldsymbol{g}}(\boldsymbol{x})$ has full rank: otherwise, we are unable to decide proper search direction. For this requirement, we have the following observations.

Proposition 2. *Assume that we have* $\deg d < \min\{d_1, \ldots, d_n\} - 1$ *and* $\deg U_i \geq 1$ *for* $i = 1, \ldots, n$. *Let* $\boldsymbol{x}^* \in V_{\boldsymbol{g}}$ *be any feasible point satisfying* (15). *Then, if the corresponding polynomials do not have a GCD whose degree exceeds d, then* $J_{\boldsymbol{g}}(\boldsymbol{x}^*)$ *has full rank.*

Proof. Let

$$\boldsymbol{x}^* = (\tilde{p}^{(1)}_{d_1}, \ldots, \tilde{p}^{(1)}_0, \ldots, \tilde{p}^{(n)}_{d_n}, \ldots, \tilde{p}^{(n)}_0, u^{(1)}_{d_1-d}, \ldots, u^{(1)}_0, \ldots, u^{(n)}_{d_n-d}, \ldots, u^{(n)}_0)$$

as in (13), with its polynomial representation expressed as in (7) (note that this assumption permits the polynomials $\tilde{P}_i(x)$ to be relatively prime in general). To verify our claim, we show that we have $\operatorname{rank}(J_{\boldsymbol{g}}(\boldsymbol{x}^*)) = \bar{d} = d_1 + \cdots + d_n - (n-1)(d-1) + (n-2)d_1 + 1$ (see (12)). Let us divide $J_{\boldsymbol{g}}(\boldsymbol{x}^*)$ into two column blocks such that $J_{\boldsymbol{g}}(\boldsymbol{x}^*) = \big(J_{\mathrm{L}} \mid J_{\mathrm{R}}\big)$, where J_{L} and J_{R} are expressed as

$$J_{\mathrm{L}} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ C_{d_1}(U_2) & C_{d_2}(U_1) & \mathbf{0} & \cdots & \mathbf{0} \\ C_{d_1}(U_3) & \mathbf{0} & C_{d_3}(U_1) & & \mathbf{0} \\ \vdots & \vdots & & \ddots & \vdots \\ C_{d_1}(U_n) & \mathbf{0} & \cdots & \mathbf{0} & C_{d_n}(U_1) \end{pmatrix},$$

$$J_{\mathrm{R}} = \begin{pmatrix} 2 \cdot {}^t\boldsymbol{u}_1 & 2 \cdot {}^t\boldsymbol{u}_2 & 2 \cdot {}^t\boldsymbol{u}_3 & \cdots & 2 \cdot {}^t\boldsymbol{u}_n \\ C_{d_1-d}(P_2) & C_{d_2-d}(P_1) & \mathbf{0} & \cdots & \mathbf{0} \\ C_{d_1-d}(P_3) & \mathbf{0} & C_{d_3-d}(P_1) & & \mathbf{0} \\ \vdots & \vdots & & \ddots & \vdots \\ C_{d_1-d}(P_n) & \mathbf{0} & \cdots & \mathbf{0} & C_{d_n-d}(P_1) \end{pmatrix},$$

respectively. Then, we have the following lemma.

Lemma 1. *We have* $\operatorname{rank}(J_{\mathrm{L}}) = \bar{d} = d_1 + \cdots + d_n - (n-1)(d-1) + (n-2)d_1$.

Proof. Let $\bar{J}$ be a submatrix of J_{L} by eliminating the top row. Since the number of rows in J_{L} is equal to $\bar{d} = d_1 + \cdots + d_n - (n-1)(d-1) + (n-2)d_1$, we show that $\bar{J}$ has full rank.

For $i = 2, \ldots, n$, let us divide column blocks $C_{d_1}(U_i)$ and $C_{d_i}(U_1)$ as

$$\begin{aligned} C_{d_1}(U_i) &= \big(\overbrace{C_{d_1}(U_i)_{\mathrm{L}}}^{d+1}\ \overbrace{C_{d_1}(U_i)_{\mathrm{R}}}^{d_1-d}\big), \\ C_{d_1}(U_i)_{\mathrm{L}} &= \begin{pmatrix} C_d(U_i) \\ \mathbf{0} \end{pmatrix} \}_{d_1-d}, \quad C_{d_1}(U_i)_{\mathrm{R}} = \begin{pmatrix} \mathbf{0} \\ C_{d_1-d-1}(U_i) \end{pmatrix} \begin{matrix} \}_{d+1} \\ \}_{d_1+d_i-2d} \end{matrix}, \quad (16) \\ C_{d_i}(U_1) &= \big(\overbrace{C_{d_i}(U_1)_{\mathrm{L}}}^{d+1}\ \overbrace{C_{d_i}(U_1)_{\mathrm{R}}}^{d_1-d}\big), \\ C_{d_i}(U_1)_{\mathrm{L}} &= \begin{pmatrix} C_d(U_1) \\ \mathbf{0} \end{pmatrix} \}_{d_1-d}, \quad C_{d_i}(U_1)_{\mathrm{R}} = \begin{pmatrix} \mathbf{0} \\ C_{d_i-d-1}(U_1) \end{pmatrix} \begin{matrix} \}_{d+1} \\ \}_{d_1+d_i-2d} \end{matrix}, \quad (17) \end{aligned}$$

respectively, thus $\bar{J}$ is expressed as

$$\bar{J} = \left(\begin{array}{cc|cc|ccc} C_{d_1}(U_2)_{\mathrm{L}} & C_{d_1}(U_2)_{\mathrm{R}} & C_{d_2}(U_1)_{\mathrm{L}} & C_{d_2}(U_1)_{\mathrm{R}} & \mathbf{0} & \mathbf{0} & \cdots \\ C_{d_1}(U_3)_{\mathrm{L}} & C_{d_1}(U_3)_{\mathrm{R}} & \mathbf{0} & \mathbf{0} & C_{d_3}(U_1)_{\mathrm{L}} & C_{d_3}(U_1)_{\mathrm{L}} & \\ \vdots & \vdots & \vdots & \vdots & & \ddots & \ddots \\ C_{d_1}(U_n)_{\mathrm{L}} & C_{d_1}(U_n)_{\mathrm{R}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots \end{array} \right.$$
$$\left. \begin{array}{ccc|cc} \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \ddots & \ddots & & \vdots & \vdots \\ & C_{d_{n-1}}(U_1)_{\mathrm{L}} & C_{d_{n-1}}(U_1)_{\mathrm{R}} & \mathbf{0} & \mathbf{0} \\ \cdots & \mathbf{0} & \mathbf{0} & C_{d_n}(U_1)_{\mathrm{R}} & C_{d_n}(U_1)_{\mathrm{R}} \end{array} \right).$$

Then, by exchanges of columns, we can transform $\bar{J}$ to $\hat{J} = \begin{pmatrix} \hat{J}_{\mathrm{L}} & \hat{J}_{\mathrm{R}} \end{pmatrix}$, where

$$\hat{J}_{\mathrm{L}} = \begin{pmatrix} C_{d_1}(U_2)_{\mathrm{L}} & C_{d_2}(U_1)_{\mathrm{L}} & \mathbf{0} & \cdots & \mathbf{0} \\ C_{d_1}(U_3)_{\mathrm{L}} & \mathbf{0} & C_{d_3}(U_1)_{\mathrm{L}} & & \mathbf{0} \\ \vdots & \vdots & & \ddots & \vdots \\ C_{d_1}(U_n)_{\mathrm{L}} & \mathbf{0} & \cdots & \mathbf{0} & C_{d_n}(U_1)_{\mathrm{L}} \end{pmatrix},$$
$$\hat{J}_{\mathrm{R}} = \begin{pmatrix} C_{d_1}(U_2)_{\mathrm{R}} & C_{d_2}(U_1)_{\mathrm{R}} & \mathbf{0} & \cdots & \mathbf{0} \\ C_{d_1}(U_3)_{\mathrm{R}} & \mathbf{0} & C_{d_3}(U_1)_{\mathrm{R}} & & \mathbf{0} \\ \vdots & \vdots & & \ddots & \vdots \\ C_{d_1}(U_n)_{\mathrm{R}} & \mathbf{0} & \cdots & \mathbf{0} & C_{d_n}(U_1)_{\mathrm{R}} \end{pmatrix}.$$

We see that nonempty rows in $\hat{J}_{\mathrm{R}}$ consist of $N(U_1,\dots,U_n)$, a generalized Sylvester matrix for $U_1,\dots,U_n$ (see (2)). By the assumption, $U_1,\dots,U_n$ are pairwise relatively prime, thus, by Prop. 1, $\mathrm{rank}(\hat{J}_{\mathrm{R}})$ is equal to the number of nonempty rows in $\hat{J}_{\mathrm{R}}$, which is equal to $d_2+\cdots+d_n+(n-1)(d_1-2d)$ (see (16) and (17)).

On the other hand, in $\hat{J}_L$, column blocks $C_{d_2}(U_1)_{\mathrm{L}}, C_{d_3}(U_1)_{\mathrm{L}},\dots,C_{d_n}(U_1)_{\mathrm{L}}$ are lower triangular matrices with $d+1$ diagonal elements, which shows that $\mathrm{rank}(\hat{J}_L)$ is equal to the sum of the number of columns in $C_{d_2}(U_1)_{\mathrm{L}}, C_{d_3}(U_1)_{\mathrm{L}},\dots$, $C_{d_n}(U_1)_{\mathrm{L}}$, which is equal to $(n-1)(d+1)$.

Furthermore, we see that the row position of diagonal elements in $C_{d_2}(U_1)_{\mathrm{L}}$, $C_{d_3}(U_1)_{\mathrm{L}},\dots,C_{d_n}(U_1)_{\mathrm{L}}$ correspond to the position of the empty rows in $\hat{J}_{\mathrm{R}}$, thus the columns in $C_{d_2}(U_1)_{\mathrm{L}}, C_{d_3}(U_1)_{\mathrm{L}},\dots,C_{d_n}(U_1)_{\mathrm{L}}$ are linearly independent along with the columns in $\hat{J}_R$. Therefore, we have

$$\mathrm{rank}(\bar{J}) = \mathrm{rank}(\hat{J}_L) + \mathrm{rank}(\hat{J}_R) = d_1+\cdots+d_n-(n-1)(d-1)+(n-2)d_1,$$

which proves the lemma.

Proof of Proposition 2 (continued). By the assumptions, we have at least one nonzero coordinate in the top row in J_{R}, while we have no nonzero coordinate in the top row in J_{L}, thus we have $\mathrm{rank}(J_{\boldsymbol{g}}(\boldsymbol{x})) = d_1+\cdots+d_n-(n-1)(d-1)+(n-2)d_1+1$, which proves the proposition. □

Proposition 2 says that, under certain conditions, so long as the search direction in the minimization problem satisfies that corresponding polynomials have a

GCD of degree not exceeding d, then $J_{\boldsymbol{g}}(\boldsymbol{x})$ has full rank, thus we can safely calculate the next search direction for approximate GCD.

3.2 Setting the Initial Values

At the beginning of iterations, we give the initial value $\boldsymbol{x}_0$ by using the singular value decomposition (SVD) [22] of $N_{d-1}(P_1,\ldots,P_n)$ (see (3)) as $N_{d-1} = U\,\Sigma\,{}^tV, U = (\boldsymbol{w}_1,\ldots,\boldsymbol{w}_{c_{d-1}}),\ \Sigma = \mathrm{diag}(\sigma_1,\ldots,\sigma_{c_{d-1}}), V = (\boldsymbol{v}_1,\ldots,\boldsymbol{v}_{c_{d-1}})$, where $\boldsymbol{w}_j \in \mathbf{R}^{r_{d-1}}$, $\boldsymbol{v}_j \in \mathbf{R}^{c_{d-1}}$ with r_k and c_k as in (4) and (5), respectively, and $\Sigma = \mathrm{diag}(\sigma_1,\ldots,\ \sigma_{c_{d-1}})$ denotes the diagonal matrix with the j-th diagonal element of which is σ_j. Note that U and V are orthogonal matrices. Then, by a property of the SVD [22, Theorem 3.3], the smallest singular value $\sigma_{c_{d-1}}$ gives the minimum distance of the image of the unit sphere $\mathrm{S}^{c_{d-1}-1}$, given as $\mathrm{S}^{c_{d-1}-1} = \{\boldsymbol{x} \in \mathbf{R}^{c_{d-1}} \mid \|\boldsymbol{x}\|_2 = 1\}$, by $N_{d-1}(P_1,\ldots,P_n)$, represented as $N_{d-1}\cdot \mathrm{S}^{c_{d-1}-1} = \{N_{d-1}\boldsymbol{x} \mid \boldsymbol{x} \in \mathbf{R}^{c_{d-1}}, \|\boldsymbol{x}\|_2 = 1\}$, from the origin, along with $\sigma_{c_{d-1}}\boldsymbol{w}_{c_{d-1}}$ as its coordinates. Thus, we have $N_{d-1}\cdot \boldsymbol{v}_{c_{d-1}} = \sigma_{c_{d-1}}\boldsymbol{w}_{c_{d-1}}$. For $\boldsymbol{v}_{c_{d-1}} = {}^t(\bar{u}^{(1)}_{d_1-d},\ldots,\bar{u}^{(1)}_0,\ldots,\bar{u}^{(n)}_{d_n-d},\ldots,\bar{u}^{(n)}_0)$, let $\bar{U}_i(x) = \bar{u}^{(i)}_{d_i-d}x^{d_i-d} + \cdots + \bar{u}^{(i)}_0 x^0$ for $i = 1,\ldots,n$. Then, $\bar{U}_1(x),\ldots,\bar{U}_n(x)$ give the least norm of $U_1P_i + U_iP_1$ satisfying $\|U_1\|_2^2 + \cdots + \|U_n\|_2^2 = 1$ by putting $U_i(x) = \bar{U}_i(x)$ in (7).

Therefore, we admit the coefficients of $P_1,\ldots,P_n$, $\bar{U}_1,\ldots,\bar{U}_n$ as the initial values of the iterations as

$$\boldsymbol{x}_0 = (p^{(1)}_{d_1},\ldots,p^{(1)}_0,\ldots,p^{(n)}_{d_n},\ldots,p^{(n)}_0,\bar{u}^{(1)}_{d_1-d},\ldots,\bar{u}^{(1)}_0,\ldots,\bar{u}^{(n)}_{d_n-d},\ldots,\bar{u}^{(n)}_0).$$

3.3 Regarding the Minimization Problem as the Minimum Distance (Least Squares) Problem

Since we have the object function f as in (14), we have

$$\begin{aligned}\nabla f(\boldsymbol{x}) = 2\cdot {}^t(&x_1 - p^{(1)}_{d_1},\ldots,x_{d_1} - p^{(1)}_0,\ldots,\\ &x_{d_1+\cdots+d_{n-1}+n} - p^{(n)}_{d_n},\ldots,x_{d_1+\cdots+d_{n-1}+d_n+n} - p^{(n)}_0,0,\ldots,0).\end{aligned}$$

However, we can regard our problem as finding a point $\boldsymbol{x} \in V_{\boldsymbol{g}}$ which has the minimum distance to the initial point $\boldsymbol{x}_0$ with respect to the $(x_1,\ldots,x_{d_1+\cdots+d_{n-1}+d_n+n})$-coordinates which correspond to the coefficients in $P_i(x)$. Therefore, as in the case for two polynomials (see the author's previous papers ([17], [21])), we change the objective function as $\bar{f}(\boldsymbol{x}) = \frac{1}{2}f(\boldsymbol{x})$, then solve the minimization problem of $\bar{f}(\boldsymbol{x})$, subject to $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}$.

3.4 Calculating the Actual GCD and Correcting the Deformed Polynomials

After successful end of the iterations, we obtain the coefficients of $\tilde{P}_i(x)$ and $U_i(x)$ satisfying (6) with $U_i(x)$ are relatively prime. Then, we need to compute the actual GCD $H(x)$ of $\tilde{P}_i(x)$. Although H can be calculated as the quotient of $\tilde{P}_i$ divided by U_i, naive polynomial division may cause numerical errors in the coefficient. Thus, we calculate the coefficients of H by the so-called least squares division [14], followed by correcting the coefficients in $\tilde{P}_i$ by using the calculated H, as follows.

For polynomials $\tilde{P}_i$, and U_i represented as in (7) and H represented as

$$H(x) = h_d x^d + \cdots + h_0 x^0,$$

solve the equations $HU_i = \tilde{P}_i$ with respect to H as solving the least squares problems of a linear system

$$C_d(U_i)\,{}^t(h_d \ldots, h_0) = {}^t(p_{d_i}^{(i)}, \cdots, p_0^{(i)}). \tag{18}$$

Let $H_i(x) \in \mathbf{R}[x]$ be a candidate for the GCD whose coefficients are calculated as the least squares solutions of (18). Then, for $i = 2, \ldots, n$, calculate the norms of the residues as

$$r_i = \sum_{j=1}^{n} \|P_j - H_i U_j\|_2^2,$$

and set the GCD $H(x)$ be $H_i(x)$ giving the minimum value of r_i so that the perturbed polynomials make the minimum amount of perturbations in total.

Finally, for the chosen $H(x)$, correct the coefficients of $\tilde{P}_i(x)$ as $\tilde{P}_i(x) = H(x) \cdot U_i(x)$ for $i = 1, \ldots, n$.

4 Experiments

We have implemented our GPGCD method on the computer algebra system Maple and compared its performance with a method based on the structured total least norm (STLN) method [11] for randomly generated polynomials with approximate GCD. The tests have been carried out on Intel Core2 Duo Mobile Processor T7400 (in Apple MacBook "Mid-2007" model) at 2.16 GHz with RAM 2GB, under MacOS X 10.5.

In the tests, we have generated random polynomials with GCD then added noise, as follows. First, we have generated a monic polynomial $P_0(x)$ of degree m with the GCD of degree d. The GCD and the prime parts of degree $m - d$ are generated as monic polynomials and with random coefficients $c \in [-10, 10]$ of floating-point numbers. For noise, we have generated a polynomial $P_{\mathrm{N}}(x)$ of degree $m - 1$ with random coefficients as the same as for $P_0(x)$. Then, we have defined a test polynomial $P(x)$ as $P(x) = P_0(x) + \frac{e_P}{\|P_{\mathrm{N}}(x)\|_2} P_{\mathrm{N}}(x)$, scaling the noise such that the 2-norm of the noise for P is equal to e_P. In the present test, we set $e_P = 0.1$.

In this test, we have compared our implementation against a method based on the structured total least norm (STLN) method [11], using their implementation (see Acknowledgments). In their STLN-based method, we have used the procedure `R_con_mulpoly` which calculates the approximate GCD of several polynomials in $\mathbf{R}[x]$. The tests have been carried out on Maple 13 with `Digits=15`

Table 1. Test results for (m, d, n): n input polynomials of degree m with the degree of approximate GCD d. See Section 4 for details.

Ex.	(m, d, n)	#Fail		Error		#Iterations		Time (sec.)	
		STLN	GPGCD	STLN	GPGCD	STLN	GPGCD	STLN	GPGCD
1	$(10, 5, 3)$	0	0	$2.31e{-}3$	$2.38e{-}3$	5.50	11.2	1.17	0.45
2	$(10, 5, 5)$	0	0	$5.27e{-}3$	$5.22e{-}3$	4.70	13.5	3.10	1.53
3	$(10, 5, 10)$	0	0	$5.48e{-}3$	$5.62e{-}3$	4.40	17.9	12.49	8.59
4	$(20, 10, 3)$	0	0	$5.17e{-}3$	$5.40e{-}3$	4.50	12.0	3.35	1.52
5	$(20, 10, 5)$	0	0	$5.89e{-}3$	$5.85e{-}3$	4.40	12.7	10.37	4.97
6	$(20, 10, 10)$	0	1	$6.31e{-}3$	$6.20e{-}3$	4.00	25.6	44.62	43.16
7	$(40, 20, 3)$	0	0	$5.32e{-}3$	$5.39e{-}3$	4.90	12.8	13.60	5.83
8	$(40, 20, 5)$	0	0	$6.01e{-}3$	$5.97e{-}3$	4.30	12.1	41.46	17.92
9	$(40, 20, 10)$	0	0	$6.41e{-}3$	$6.25e{-}3$	4.10	8.90	200.88	60.21

executing hardware floating-point arithmetic. For every example, we have generated 10 random test cases as in the above. In executing the GPGCD method, we set $u = 100$ and a threshold of the 2-norm of the "update" vector in each iteration $\varepsilon = 1.0 \times 10^{-8}$; in `R_con_mulpoly`, we set the tolerance $e = 1.0 \times 10^{-8}$.

Table 1 shows the results of the test. In each test, we have given several polynomials of the same degree as the input. The second column with (m, d, n) denotes the degree of input polynomials, degree of GCD, and the number of input polynomials, respectively. The columns with "STLN" are the data for the STLN-based method, while those with "GPGCD" are the data for the GPGCD method. "#Fail" is the number of "failed" cases such as: in the STLN-based method, the number of iterations exceeds 50 times (which is the built-in threshold in the program), while, in the GPGCD method, the number of iterations exceeds 100 times. All the other data are the average over results for the "not failed" cases: "Error" is the sum of perturbation $\sum_{i=1}^{n} \|\tilde{P}_i - P_i\|_2^2$, where "$ae - b$" denotes $a \times 10^{-b}$; "#Iterations" is the number of iterations; "Time" is computing time in seconds.

We see that, in the most of tests, both methods calculate approximate GCD with almost the same amount of perturbations. In the most of tests, the GPGCD method runs faster than STLN-based method. However, running time of the GPGCD method increases as much as that of the STLN-based method in some cases with relatively large number of iterations (such as Ex. 6). There is a case in which the GPGCD method does not converge (Ex. 6). Factors leading to such phenomena is under investigation.

5 Concluding Remarks

Based on our previous research ([17], [21]), we have extended our GPGCD method for more than two input polynomials with the real coefficients. We have shown that, at least theoretically, our algorithm properly calculates an approximate GCD under certain conditions for multiple polynomial inputs.

Our experiments have shown that, in the case that the number of iterations is relatively small, the GPGCD method calculates an approximate GCD efficiently with almost the same amount of perturbations as the STLN-based method. However, computing time of the GPGCD method increases as the number of iterations becomes larger; it suggests that we need to reduce the computing time of each iteration in the GPGCD method for the cases with relatively large number of iterations. It is desirable to have more detailed experiments for analyzing stability, performance for input polynomials of larger degree, etc.

For the future research, generalizing this result to polynomials with the complex coefficients will be among our next problems. It is also an interesting problem how the choice of P_1 affects the performance of the algorithm. Furthermore, one can also use arbitrary linear combination to transform $\gcd(P_1, P_2, \ldots, P_n)$ to $\gcd(P_1, a_2P_2 + \cdots + a_nP_n)$. This will reduce the size of the generalized Sylvester matrix and will be another approach for calculating approximate GCD.

Acknowledgments

We thank Professor Erich Kaltofen for making their implementation for computing approximate GCD available on the Internet. We also thank the anonymous reviewers for their valuable suggestions.

References

1. Beckermann, B., Labahn, G.: A fast and numerically stable Euclidean-like algorithm for detecting relatively prime numerical polynomials. J. Symbolic Comput. 26(6), 691–714 (1998), Symbolic numeric algebra for polynomials
2. Sasaki, T., Noda, M.T.: Approximate square-free decomposition and root-finding of ill-conditioned algebraic equations. J. Inform. Process. 12(2), 159–168 (1989)
3. Schönhage, A.: Quasi-gcd computations. J. Complexity 1(1), 118–137 (1985)
4. Corless, R.M., Gianni, P.M., Trager, B.M., Watt, S.M.: The singular value decomposition for polynomial systems. In: Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation, pp. 195–207. ACM, New York (1995)
5. Emiris, I.Z., Galligo, A., Lombardi, H.: Certified approximate univariate GCDs. J. Pure Appl. Algebra 117/118, 229–251 (1997), Algorithms for algebra (Eindhoven, 1996)
6. Corless, R.M., Watt, S.M., Zhi, L.: QR factoring to compute the GCD of univariate approximate polynomials. IEEE Trans. Signal Process. 52(12), 3394–3402 (2004)
7. Zarowski, C.J., Ma, X., Fairman, F.W.: QR-factorization method for computing the greatest common divisor of polynomials with inexact coefficients. IEEE Trans. Signal Process. 48(11), 3042–3051 (2000)
8. Zhi, L.: Displacement structure in computing approximate GCD of univariate polynomials. In: Computer mathematics: Proc. Six Asian Symposium on Computer Mathematics (ASCM 2003), River Edge, NJ. Lecture Notes Ser. Comput., vol. 10, pp. 288–298. World Sci. Publ. (2003)
9. Pan, V.Y.: Computation of approximate polynomial GCDs and an extension. Inform. and Comput. 167(2), 71–85 (2001)

10. Chin, P., Corless, R.M., Corliss, G.F.: Optimization strategies for the approximate GCD problem. In: Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation, pp. 228–235. ACM, New York (1998) (electronic)
11. Kaltofen, E., Yang, Z., Zhi, L.: Approximate greatest common divisors of several polynomials with linearly constrained coefficients and singular polynomials. In: Proceedings of the 2006 International Symposium on Symbolic and Algebraic Computation, pp. 169–176. ACM, New York (2006)
12. Kaltofen, E., Yang, Z., Zhi, L.: Structured low rank approximation of a Sylvester matrix. In: Wang, D., Zhi, L. (eds.) Symbolic-Numeric Computation. Trends in Mathematics, pp. 69–83. Birkhäuser, Basel (2007)
13. Karmarkar, N.K., Lakshman, Y.N.: On approximate GCDs of univariate polynomials. J. Symbolic Comput. 26(6), 653–666 (1998), Symbolic numeric algebra for polynomials
14. Zeng, Z.: The approximate GCD of inexact polynomials, Part I: a univariate algorithm (extended abstract), 8 p. (2004) (preprint)
15. Ohsako, N., Sugiura, H., Torii, T.: A stable extended algorithm for generating polynomial remainder sequence. Trans. Japan Soc. Indus. Appl. Math. 7(3), 227–255 (1997) (in Japanese)
16. Sanuki, M., Sasaki, T.: Computing approximate GCDs in ill-conditioned cases. In: SNC 2007: Proceedings of the 2007 International Workshop on Symbolic-Numeric Computation, pp. 170–179. ACM, New York (2007)
17. Terui, A.: An iterative method for calculating approximate GCD of univariate polynomials. In: Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation, pp. 351–358. ACM Press, New York (2009)
18. Tanabe, K.: A geometric method in nonlinear programming. J. Optim. Theory Appl. 30(2), 181–210 (1980)
19. Rosen, J.B.: The gradient projection method for nonlinear programming. II. Nonlinear constraints. J. Soc. Indust. Appl. Math. 9, 514–532 (1961)
20. Rupprecht, D.: An algorithm for computing certified approximate GCD of n univariate polynomials. J. Pure Appl. Algebra 139, 255–284 (1999)
21. Terui, A.: GPGCD, an iterative method for calculating approximate GCD of univariate polynomials, with the complex coefficients. In: Proceedings of the Joint Conference of ASCM 2009 and MACIS 2009. COE Lecture Note., Faculty of Mathematics, vol. 22, pp. 212–221. Kyushu University (December 2009)
22. Demmel, J.W.: Applied numerical linear algebra. Society for Industrial and Applied Mathematics (SIAM), Philadelphia (1997)

Derivation of Explicit Difference Schemes for Ordinary Differential Equations with the Aid of Lagrange–Burmann Expansions

Evgenii V. Vorozhtsov

Khristianovich Institute of Theoretical and Applied Mechanics, Russian Academy of Sciences, Novosibirsk 630090, Russia
vorozh@itam.nsc.ru

Abstract. We propose to derive the explicit multistage methods of the Runge–Kutta type for ordinary differential equations (ODEs) with the aid of the expansion of grid functions into the Lagrange–Burmann series. New explicit first- and second-order methods are derived, which are applied to the numerical integration of the Cauchy problem for a moderately stiff ODE system. It turns out that the L_2 norm of the error of the solution obtained by the new numerical second-order method is 50 times smaller than in the case of the classical second-order Runge–Kutta method.

1 Introduction

Ordinary differential equations are of great importance at the modeling of many applied problems. To find the solution of ODEs one has, as a rule, to use the approximate methods. The finite-difference methods have gained here a widespread acceptance. But at the use of these methods one has to account for the accuracy and stability problems. Stability problems become especially complicated in the case of the stiff ODE systems, which are typical of the chemical kinetics problems. It was proved in [6] that the explicit linear multistage method cannot be A-stable. One can then derive the implicit multistep methods as well as the implicit Runge–Kutta methods, which possess the A-stability property, but these methods require iterations [7]. In this connection, there are in the literature the investigations, whose purpose is the obtaining of explicit methods of the Runge–Kutta type with an extended stability region [8,1,2,9,12]. In particular, it was proposed for the first time in [8] to derive the explicit second-order Runge–Kutta methods with an extended stability region on the negative real axis by using the shifted Chebyshev polynomials of the first kind; these explicit methods were termed the Runge–Kutta–Chebyshev (RKC) methods. Abdulle [1,2] has constructed also the families of the second- and fourth-order RKC methods. A review of the RKC methods may be found in [9,12]. A shortcoming of the RKC methods is that they need large computer time expenses for their implementation. This is related to the fact that the RKC methods require the execution of several dozens of stages at one step; for example, the RKC method proposed in

[12] requires up to 320 stages. Therefore, there is a need in a search for other possibilities of deriving the explicit methods, which possess an extended stability region and, at the same time, a higher local accuracy.

We will consider in the following the Cauchy problem of the form

$$\frac{du}{dx} = f(x,u), \quad x \in [x_0, X], \tag{1}$$

$$u(x_0) = u_0, \tag{2}$$

where x is the independent variable, $f(x,u)$ is a given scalar function, $x_0 < X$, x_0, X, and u_0 are the given quantities. The expansion of the solution of the Cauchy problem under consideration into the truncated Taylor series is the most widespread approach to the derivation of difference methods of various orders for solving equation (1). The Taylor series is a series in the powers of function $x - x_0$. Thus, the truncated Taylor series is a polynomial. It is, however, known from the theory of the approximation of functions by polynomials (see, for example, [14]) that such approximations result in the oscillations of the interpolating function in the neighborhood of the discontinuities of the original function as well as in the regions of large solution gradients.

On the other hand, it is known that one can expand the function also in the power series of general form, that is in the powers of some function $\varphi(x - x_0)$. One can then choose the function $\varphi(x)$ in such a way that the amplitude of numerical solution oscillations reduces. The task of the expansion of a function into the series in powers of other function is solved efficiently with the aid of the Lagrange–Burmann formula [3,11]. It was proposed for the first time in [18] to apply the Lagrange–Burmann expansion for constructing the difference schemes for the numerical integration of partial differential equations of the hyperbolic type, in particular, the Euler equations governing the flows of an inviscid compressible non-heat-conducting gas. It turned out that the difference schemes derived in this way suit well for the numerical computation of gas flows with shock waves and other strong discontinuities despite the fact that neither the artificial viscosity nor TVD limiters were introduced in the derived schemes.

It was proposed in [18] to use the explicit Runge–Kutta schemes to ensure the required approximation order of the constructed schemes in time. Thus, the difference schemes derived in [18] combined the Lagrange–Burmann formula for the approximation of partial derivatives with respect to spatial variables with the Taylor expansion formula for the approximation of derivatives with respect to time. One can, however, use the Lagrange–Burmann expansion formula also for the approximation of the derivatives with respect to time. In the present work, the difference schemes are derived for the numerical solution of the Cauchy problem (1),(2), which have the local orders of accuracy $p = 1$ and 2. In Section 2, we discuss the Lagrange–Burmann expansion formula. In Section 3, we present the general procedure for deriving the explicit methods of the Runge–Kutta type of a finite order with the aid of the expansions of grid functions into the Lagrange–Burmann series. In Section 4, an analog of the Euler explicit method is derived with the aid of the Lagrange–Burmann expansion. In Section 5, an explicit second-order method of the Runge–Kutta type is derived, and the examples of the numerical solution of a moderately stiff ODE system with the aid of new explicit methods are presented.

2 The Lagrange–Burmann Expansion Formula

Let $u(x)$ be an infinitely differentiable function and let $y = g(x)$, $y_0 = g(x_0)$, $g'(x_0) \neq 0$. Then the Lagrange–Burmann expansion formula may be written in the form [3]

$$u(x) = u(x_0) + \sum_{k=1}^{\infty} \frac{(y-y_0)^k}{k!} \left\{ \frac{d^{k-1}}{dx^{k-1}} \left[u'(x) \left(\frac{x-x_0}{g(x)-y_0} \right)^k \right] \right\}_{x=x_0} . \tag{3}$$

It is obvious that in the particular case $g(x) = x$, $y_0 = x_0$ the Lagrange–Burmann formula (3) goes over to the Taylor formula

$$u(x) = u(x_0) + \sum_{k=1}^{\infty} \left\{ \frac{d^k u(x)}{dx^k} \right\}_{x_0} \frac{(x-x_0)^k}{k!}. \tag{4}$$

It will be convenient for the following to rewrite formula (3) in the form, which is maximally similar to the Taylor formula (4). Let us introduce the function φ by formula $\varphi(x-x_0) = g(x) - y_0$. Then $\varphi(x-x_0) = \varphi(0)$ at $x = x_0$, and we must require that $\varphi(0) = 0$, $\varphi'(0) \neq 0$, that is the point $x = 0$ is the zero of the first order of the function $\varphi(x)$. Upon introducing the function $\varphi(x)$ we can rewrite the Lagrange–Burmann formula (3) in the form

$$u(x) = u(x_0) + \sum_{k=1}^{\infty} \frac{[\varphi(x-x_0)]^k}{k!} \left\{ \frac{d^{k-1}}{dx^{k-1}} \left[u'(x) \left(\frac{x-x_0}{\varphi(x-x_0)} \right)^k \right] \right\}_{x=x_0} . \tag{5}$$

Thus, formula (5) enables the expansion of the function $u(x)$ into the series in powers of the function $\varphi(x-x_0)$.

Let us denote by b_k the coefficient affecting $[\varphi(x-x_0)]^k$ in (5), that is

$$b_k = \frac{1}{k!} \left\{ \frac{d^{k-1}}{dx^{k-1}} \left[u'(x) \left(\frac{x-x_0}{\varphi(x-x_0)} \right)^k \right] \right\}_{x=x_0} , \quad k = 1, 2, \ldots . \tag{6}$$

In particular,

$$b_1 = \left[u'(x) \frac{x-x_0}{\varphi(x-x_0)} \right]_{x=x_0} = u'(x_0) \frac{0}{0} .$$

It is seen that there is an uncertainty of the form $0/0$ in the expression for b_1. To resolve this uncertainty we make use of the following representation of the coefficients b_k [11]:

$$b_k = \lim_{x \to x_0} \frac{1}{k!} \left\{ \frac{d^{k-1}}{dx^{k-1}} \left[u'(x) \left(\frac{x-x_0}{\varphi(x-x_0)} \right)^k \right] \right\}, \quad k = 1, 2, \ldots . \tag{7}$$

In particular, we have at $k = 1$:

$$b_1 = \lim_{x\to x_0}\left[u'(x)\frac{x-x_0}{\varphi(x-x_0)}\right] = \left[\lim_{x\to x_0} u'(x)\right]\cdot\left[\lim_{x\to x_0}\frac{x-x_0}{\varphi(x-x_0)}\right] = \frac{u'(x_0)}{\varphi'(0)}. \quad (8)$$

We have used the L'Hospital's rule at the computation of the second limit. The complexity of the computation of limits in expressions (7) for b_k grows nonlinearly with increasing k. The application of the CAS *Mathematica* has proved to be very useful here. Note that the *Mathematica* function `Limit[...]` was unable to calculate even the simple limit (8). Therefore, at the realization of the computation of coefficients b_k in the *Mathematica* system we have at first expanded the function $\varphi(x)$ into the truncated Maclaurin series. The limit (7) was then found simply by substituting $x = 0$ into the obtained rational expressions. Here is the corresponding program in the language of the *Mathematica* system:

```
m = 4; Phi[x_] := x*Sum[(D[fi[x], {x, k}] /. x -> 0)*x^(k - 1)/k!,
{k, m}]; Do[b = (1/k!)*D[u'[x + x0]*(x/Phi[x])^k, {x, k - 1}];
 b = b /. x -> 0; b = Expand[b];
 Print["b(", k, ") = ", TraditionalForm[b]], {k, m}];
```

As a result of computations by the above *Mathematica* program we have obtained formula (8) for b_1, and we have obtained the following expressions for b_2, b_3, b_4:

$$b_2 = \frac{u''(x_0)}{2\varphi'(0)^2} - \frac{u'(x_0)\varphi''(0)}{2\varphi'(0)^3}, \quad (9)$$

$$b_3 = \frac{u'(x_0)\varphi''(0)^2}{2\varphi'(0)^5} - \frac{u''(x_0)\varphi''(0)}{2\varphi'(0)^4} + \frac{u^{(3)}(x_0)}{6\varphi'(0)^3} - \frac{u'(x_0)\varphi^{(3)}(0)}{6\varphi'(0)^4}, \quad (10)$$

$$b_4 = -\frac{5u'(x_0)\varphi''(0)^3}{8\varphi'(0)^7} + \frac{5u''(x_0)\varphi''(0)^2}{8\varphi'(0)^6} - \frac{u^{(3)}(x_0)\varphi''(0)}{4\varphi'(0)^5} + \frac{5u'(x_0)\varphi^{(3)}(0)\varphi''(0)}{12\varphi'(0)^6}$$
$$- \frac{u''(x_0)\varphi^{(3)}(0)}{6\varphi'(0)^5} + \frac{u^{(4)}(x_0)}{24\varphi'(0)^4} - \frac{u'(x_0)\varphi^{(4)}(0)}{24\varphi'(0)^5}. \quad (11)$$

If in the particular case one sets in formulas (8)–(11) $\varphi(x) = x$, then it is easy to see that the obtained expressions for coefficients b_k go over to the coefficients affecting $(x - x_0)^k$ in the Taylor formula (4).

3 General Procedure of the Explicit Methods of the Runge–Kutta Type

This procedure is intended for an approximate numerical solution of the Cauchy problem (1),(2). We at first introduce the grid as a finite set $G_h = \{x_0, x_1, \ldots, x_N\}$ of the nodes $x = x_j$, $j = 0, 1, \ldots, N$ in the interval $[x_0, X]$, and $x_0 < x_1 < \cdots < x_N = X$. The approximate values of the solution $u(x_j)$ are found within the framework of the Runge–Kutta methods sequentially for the increasing numbers j starting from $j = 0$. Let the value $u(x_j)$ be known. The solution value in the next node x_{j+1} is then computed by the formula

$$u(x_{j+1}) = u(x_j) + \Delta u_{h,j}. \quad (12)$$

The formula for computing $\Delta u_{h,j}$ depends on the number of stages of the employed Runge–Kutta method. In the general case of the $(q+1)$-stage method, one computes at first sequentially the quantities

$$\begin{aligned}
g_0 &= \varphi_j(h_j)f(x,u),\\
g_1 &= \varphi_j(h_j)f(x+\alpha_1 h_j, u+\beta_{10}g_0),\\
g_2 &= \varphi_j(h_j)f(x+\alpha_2 h_j, u+\beta_{20}g_0+\beta_{21}g_1),\\
&\dots\\
g_q &= \varphi_j(h_j)f(x+\alpha_q h_j, u+\beta_{q0}g_0+\beta_{q1}g_1+\dots+\beta_{q,q-1}g_{q-1}),
\end{aligned} \tag{13}$$

where $h_j = x_{j+1}-x_j$, $j=0,\dots,N-1$, $q\ge 0$, $\alpha_1,\dots,\alpha_q$; $\beta_{10},\beta_{20},\beta_{21},\dots,\beta_{q0}$, $\beta_{q1},\dots,\beta_{q,q-1}$ are the parameters of the Runge–Kutta method. The solution increment $\Delta u_{h,j}$ is then computed by formula

$$\Delta u_{h,j} = \sum_{i=0}^{q} A_i g_i, \tag{14}$$

where $A_0, A_1,\dots,A_q$ are the parameters of the method.

Along with the approximate formula (14) for the solution increment we can write down the "exact" formula for solution increment Δu by using the Lagrange–Burmann expansion formula (5):

$$\Delta u_j = u(x_j+h_j)-u(x_j) = \sum_{k=1}^{m}\frac{h_j^k}{k!}\cdot\left\{\frac{d^{k-1}}{dx^{k-1}}\left[u'(x)\left(\frac{x-x_j}{\varphi_j(x-x_j)}\right)^k\right]\right\}_{x=x_j}, \tag{15}$$

where m is a given finite natural number, $m\ge q+1$. We now compose the difference

$$\delta u_j = \Delta u_j - \Delta u_{h,j} \tag{16}$$

and require the satisfaction of the relation $\delta u_j = O(h_j^p)$, where p is the maximum possible exponent for the given number of stages q. The maximization of exponent p is carried out at the expense of an appropriate selection of the parameters (α), (β), (A) in (13),(14).

Note that the form of the function $\varphi_j(\cdot)$ entering (13) can change at a passage from the jth node to the $(j+1)$th node. At the expense of this, one can reach an increase in the accuracy and stability of the method of the Runge–Kutta type (12),(13),(14).

In the simplest case of a uniform grid when $x_{j+1}-x_j = h = \text{const}\ \forall j$ one can also take the same function $\varphi(x)$ in all grid nodes. One can then compute the value $\varphi(h)$ once, prior to the computations by the Runge–Kutta method (12),(13),(14). Then it follows from (13) that the proposed method will require an amount of computations, which does not exceed the amount of computations by the classical Runge–Kutta method, for which $\varphi(h)=h$.

4 The First-Order Method

At $m = 1$ in (15) we obtain from (13) with regard for (8) the formula

$$u(x_j + h_j) = u(x_j) + \frac{u'(x_j)}{\varphi_j'(0)} \cdot \varphi_j(h_j). \tag{17}$$

With regard for (1) we can rewrite equality (17) in the form

$$u(x_j + h_j) = u(x_j) + \frac{\varphi_j(h_j)}{\varphi_j'(0)} \cdot f(x_j, u(x_j)). \tag{18}$$

In the particular case $\varphi_j(x) = x$ the difference scheme (18) goes over to the well-known explicit first-order Euler method [7,10,4]

$$u(x_j + h_j) = u(x_j) + h_j \cdot f(x_j, u(x_j)). \tag{19}$$

One can, therefore, consider the scheme (18) as an analog of the Euler method (19). We will refer to the method (18) for the sake of brevity as to the LB1 method (that is this is the first-order method obtained by using the expansion of the grid function into the Lagrange–Burmann series).

Let us introduce the notation

$$\gamma = \frac{\varphi_j(h_j)}{h_j \varphi_j'(0)} = \frac{h_j \tilde{\varphi}_j\left(\frac{x - x_j}{h_j}\right)\Big|_{x = x_j + h_j}}{h_j^2 \left[\frac{d}{dx}\tilde{\varphi}_j\left(\frac{x - x_j}{h_j}\right)\right]\Big|_{x = x_j}}. \tag{20}$$

Using the coefficient γ we can rewrite scheme (18) as

$$u(x_j + h_j) = u(x_j) + \gamma h_j \cdot f(x_j, u(x_j)). \tag{21}$$

To have the possibility of using the LB1 method in practical computations it is necessary to specify the function $\varphi(x)$. The following function was used in [18] for gas-dynamic computations:

$$\varphi_j(x - x_j) = h_j \tilde{\varphi}_j(\xi), \tag{22}$$

where $\xi = (x - x_j)/h_j$,

$$\tilde{\varphi}_j(\xi) = \tanh(\beta \xi), \tag{23}$$

β is a positive user-specified constant. The choice of the function $\tanh(\beta\xi)$ is due to that it can simulate both fast and slow solution components at the expense of the choice of the constant β. The presence of these two different components is typical of stiff ODE systems. Then we obtain in the case of specifying the function $\tilde{\varphi}_j(\xi)$ by formula (23): $\gamma = \frac{\tanh(\beta)}{\beta}$.

To estimate the local error of the LB1 method we set $m = 2$ in (15):

$$u(x_j + h_j) = u(x_j) + \frac{u'(x_j)}{\varphi_j'(0)} \varphi_j(h_j) + r_2 + O(h^3), \tag{24}$$

where according to (9)

$$r_2 = \Big[\frac{u''(x_j)}{2\varphi_j'(0)^2} - \frac{u'(x_j)\varphi_j''(0)}{2\varphi_j'(0)^3}\Big]\varphi_j^2(h). \tag{25}$$

It is seen from (25) that in the general case when the function $\varphi_j(x)$ is neither even nor odd and when $\varphi_j''(0) \neq 0$ the local error r_2 of the LB1 method may be both higher and lower than in the case of the Euler method (19), which depends both on the choice of the function $\varphi_j(x)$ and on the local behavior of the solution $u(x)$. We now consider the particular case when $\varphi_j(x)$ is an odd function. Then the following equalities are known to be satisfied: $\varphi^{(2m)}(0) = 0$, $m = 1, 2, \ldots$. Note that the function $\varphi_j(x) = x$ used in the Taylor expansion is also odd. Thus, in the case of the odd function $\varphi_j(x)$ formula (25) simplifies to

$$r_2 = (1/2)u''(x_j)[\varphi_j(h_j)/\varphi_j'(0)]^2. \tag{26}$$

In the case of the method (19), the corresponding error has the form $r_{2,RK1} = (h_j^2/2)u''(x_j)$. Consider the ratio $r_2/r_{2,RK1} = [\varphi_j(h_j)/(\varphi_j'(0)h_j)]^2$. To ensure the satisfaction of the inequality $r_2/r_{2,RK1} < 1$ it is necessary that

$$\varphi_j(h_j)/[\varphi'(0)h_j] < 1. \tag{27}$$

Let us take, for example, the function $\varphi_j(x)$ in the form (22)–(23). With regard for (10), the product $\varphi^{(3)}(0)\varphi_j^3(h_j)$ will enter the term of the order of smallness $O(\varphi_j^3(h_j))$. But if one uses for $\varphi_j(x)$ formulas (22)–(23), then one obtains: $\varphi_j^{(3)}(0) = -2\beta^3/(h_j^2)$, consequently, $\varphi_j^{(3)}(0)\varphi_j^3(h_j) = O(h_j)$. In order to ensure that the given product has the order of smallness $O(h_j^3)$ it is sufficient to ensure that $\varphi^{(3)}(0) = O(1)$. We will proceed as follows: let us assume that the nondimensionalization was carried out in equation (1) in such a way that $X - x_0 = O(1)$. Instead of the function (22)–(23) we introduce the following function:

$$\varphi_j(x) = b(x + b_1x^3), \tag{28}$$

where b and b_1 are constants, and $b > 0$. Then $\varphi_j^{(3)}(0) = 6bb_1$, $\varphi_j^{(k)} = 0$, $k > 3$. Let us choose the constant b_1 from the requirement that $\varphi_j(h) > 0$. To this end the constant b_1 must satisfy the inequality $b_1 > -h^{-2}$. To ensure the satisfaction of inequality (27) it is necessary that $b_1 < 0$ in (28).

Consider the Dahlquist's equation [7,16]

$$u'(x) = \lambda u(x), \quad u(x_0) = u_0, \quad \lambda \in \mathbb{C}. \tag{29}$$

Applying the method (21) to equation (29) we obtain:

$$u(x_j + h_j) = (1 + \gamma h_j\lambda)u(x_j). \tag{30}$$

Let $z = \lambda h_j$ and $R(z) = 1 + \gamma z$. Then we can rewrite (30) in the form

$$u(x_j + h_j) = R(z)u(x_j). \tag{31}$$

The region of *absolute stability* is defined as a region in the complex z-plane, in which $|R(z)| \le 1$. Polynomial $R(z)$, which arises at the application of the Runge–Kutta method to equation (29), is called the stability polynomial. In particular, we obtain from the condition $|1 + \gamma z| \le 1$ the following condition of the absolute stability of the LB1 scheme on the negative real axis: $-2 \le \gamma z \le 0$. This implies the inequalities

$$0 \le h \le \frac{2}{\gamma|\lambda|}. \tag{32}$$

We obtain from here that in order to ensure a larger region of the absolute stability of the LB1 method than in the case of the RK1 method the following inequalities must be satisfied:

$$0 < \gamma < 1. \tag{33}$$

Let us now find those values of b and b_1 in (28) at which inequalities (33) will be satisfied. Using formula (20), we obtain from (33) the inequalities

$$-1 < b_1 h^2 < 0. \tag{34}$$

To obtain the limitation from above for $|b_1|$ we substitute in (32) the quantity $\gamma = 1 + b_1 h^2$ and neglect the term $b_1 h^2$. We then obtain that the value of step h, which is maximally admissible from the viewpoint of the absolute stability, is $h_{\max} = 2/|\lambda|$. Substituting this value instead of h in (34), we obtain the inequalities

$$-(1/4)|\lambda|^2 < b_1 \le 0. \tag{35}$$

5 The Second-Order Method

Consider the two-stage Runge–Kutta method. For this purpose we set $q = 1$ in (13). The increment $\Delta u_{h,j}$ in (12) is then computed with regard for (14) as

$$\Delta u_{h,j} = A_0 g_0 + A_1 g_1, \tag{36}$$

where

$$g_0 = \varphi_j(h_j) f(x, u), \quad g_1 = \varphi_j(h_j) f(x + \alpha_1 h_j, u + \beta_{10} g_0). \tag{37}$$

In order to compare (36) with quantity (15) let us expand g_1 into the Lagrange–Burmann series at point (x, u) as a function of two variables in the powers of functions $\varphi_j(x - x_j)$ and $\varphi_j(u - u_j)$ (note that at the derivation of the classical explicit Runge–Kutta methods, the function $g_1(x, u)$ is expanded into the Taylor series in powers of the variables $(x - x_j)$ and $(u - u_j)$, see, for example, [7,10]). The general formula of the expansion of a bivariate function into a series in powers of two other arbitrary functions of two variables is presented, for example, in [5]. In accordance with the construction (13) of the method of the Runge–Kutta type, which was proposed in Section 3, in our case it is necessary to expand the function $g_1(x, u)$ into the series in powers of the same function

$\varphi(\cdot)$, but with different arguments. The general Lagrange formula then simplifies greatly:

$$g_1(x,u) = g_1(x_j,u_j) + \sum_{\substack{l=0 \\ l+k>0}}^{m} \sum_{k=0}^{m} B_{l,k}(x_j,u_j) \cdot \frac{[\varphi(x-x_j)]^l \cdot [\varphi(u-u_j)]^k}{l!k!}, \quad (38)$$

where we have introduced, for the sake of brevity, the notation $\varphi(\cdot) \equiv \varphi_j(\cdot)$, and

$$B_{l,k}(x_j,u_j) = \left\{ D_1^{l-1} D_2^{k-1} \left[\left(\frac{x-x_j}{\varphi(x-x_j)} \right)^l \left(\frac{u-u_j}{\varphi(u-u_j)} \right)^k D_1 D_2 g_1(x,u) \right] \right\}_{\substack{x=x_j, \\ u=u_j}} . \quad (39)$$

Here and in the following, $D_1 = \partial/\partial x$, $D_2 = \partial/\partial u$. It is seen from formulas (38) and (39) that the complexity of the computation of the Lagrange expansion of a bivariate function grows nonlinearly with increasing m. The use of the system *Mathematica* has proved to be very efficient here. However, at the symbolic computation of coefficients $B_{l,0}$ $(l \geq 1)$ and $B_{0,k}$ $(k \geq 1)$ there arises the difficulty related to the presence of the negative-order derivatives in (39). To solve this problem let us present the sum standing on the right-hand side of equality (38) in the form of three sums: $g_1(x,u) = g_1(x_j,u_j) + S_1 + S_2 + S_3$, where

$$S_1 = \sum_{l=1}^{m} B_{l,0} \frac{[\varphi(x-x_j)]^l}{l!}, \qquad S_2 = \sum_{k=1}^{m} B_{0,k} \frac{[\varphi(u-u_j)]^k}{k!},$$

$$S_3 = \sum_{l=1}^{m} \sum_{k=1}^{m} B_{l,k} \frac{[\varphi(x-x_j)]^l [(\varphi(u-u_j)]^k}{l!k!}.$$

We now transform the formula for $B_{j,0}$ by using the relation

$$D_2^{-1} D_1 D_2 = D_2^{-1} D_2 D_1 = D_1.$$

$$B_{l,0} = \left\{ D_1^{l-1} D_2^{-1} \left[\left(\frac{x-x_j}{\varphi(x-x_j)} \right)^l D_1 D_2 g_1 \right] \right\}_{x=x_j, u=u_j}$$
$$= \left\{ D_1^{l-1} \left[\left(\frac{x-x_j}{\varphi(x-x_j)} \right)^l D_1 g_1 \right] \right\}_{x=x_j, u=u_j}, \quad l \geq 1. \quad (40)$$

Similarly,

$$B_{0,k} = \left\{ D_2^{k-1} \left[\left(\frac{u-u_j}{\varphi(u-u_j)} \right)^k D_2 g_1 \right] \right\}_{x=x_j, u=u_j}, \quad k \geq 1. \quad (41)$$

Owing to formulas (40) and (41) the symbolic computation of the entire expansion (38) becomes a straightforward matter. For the sake of brevity, we present a fragment of the *Mathematica* code, which calculates the entries $B_{l,k}$ in S_3:

```
 m = 2; B = Table[0, {j, m + 1}, {k, m + 1}];
 Phi[x_]:= x*Sum[(D[fi[x], {x, k}] /. x -> 0)*x^(k - 1)/k!, {k, m + 1}];

w[x_, y_, j_, k_]:= (x/Phi[x])^j * (y/Phi[y])^k * f^(1,1)[x, y];

 Do[ Do[ b = D[w[x, y, j, k], {x, j - 1}, {y, k - 1}];
 b = b /. {x-> 0, y -> 0}; B[[j + 1, k + 1]] = b, {j, m}], {k, m}];

B = B/. {f^(i_,j_)[0, 0] -> f^(i,j)[x0, u0]};
```

After that, the quantity (36) is computed with the aid of the following *Mathematica* commands:

```
 g0 = fi[h]*f[x0,u0]; g1 = fi[h]*df/. {x-> x0 + a1*h, u-> u0 + b10*g0};
 g1 = g1/. {fi[b10 f[x0, u0] fi[h]]-> fi'[0]*b10*f[x0, u0]*fi[h],
 fi[a1 h] -> fi'[0]*a1*h}; duh = Expand[A0*g0 + A1*g1];
```

The quantity Δu_j (15) was calculated with the aid of a compact recursive *Mathematica* program presented in [17].

As a result of symbolic computations, the following expression was obtained for the quantity (16):

$$\delta u_j = r_1\varphi(h) + r_2\varphi^2(h) + r_3\varphi^3(h), \tag{42}$$

where we have introduced, for the sake of brevity, the notation $h \equiv h_j$, and

$$\begin{aligned} r_1 &= -A_0 - A_1 + [\varphi'(0)]^{-1}, \quad r_2 = r_{2,0}f + r_{2,1}f_x + r_{2,2}ff_u, \\ r_3 &= r_{30}f + r_{31}f_{xx} + r_{32}ff_{xu} + r_{33}f^2f_{uu} + r_{34}ff_u + r_{35}ff_u^2 \\ &\quad + r_{36}f^2f_u + r_{37}f_x + r_{38}f_xf_u. \end{aligned} \tag{43}$$

Here

$$\begin{aligned} &r_{2,0} = -\frac{\varphi''(0)}{2[\varphi'(0)]^3}, \; r_{2,1} = \frac{1}{2[\varphi'(0)]^2} - \frac{\alpha_1A_1h}{\varphi(h)}, \; r_{2,2} = \frac{1}{2[\varphi'(0)]^2} - A_1\beta_{10}, \\ &r_{3,0} = \frac{[\varphi''(0)]^2}{2[\varphi'(0)]^5} - \frac{\varphi^{(3)}(0)}{6[\varphi'(0)]^4}, \quad r_{3,1} = \frac{1}{6[\varphi'(0)]^3} - \frac{1}{2}\alpha_1^2A_1\frac{h^2}{\varphi^2(h)}, \\ &r_{3,2} = \frac{1}{3[\varphi'(0)]^3} - \alpha_1A_1\beta_{10}\frac{h}{\varphi(h)}, \quad r_{3,3} = \frac{1}{6[\varphi'(0)]^3} - \frac{1}{2}A_1\beta_{10}^2, \\ &r_{3,4} = \frac{\varphi''(0)}{2[\varphi'(0)]^4}; \; r_{35} = \frac{1}{6[\varphi'(0)]^3}, \quad r_{3,6} = \frac{A_1\beta_{10}^2\varphi''(0)}{2\varphi'(0)}, \\ &r_{3,7} = \frac{\varphi''(0)}{2[\varphi'(0)]^4} + \frac{\alpha_1^2A_1h^2\varphi''(0)}{2\varphi^2(h)\varphi'(0)}, \quad r_{3,8} = \frac{1}{6[\varphi'(0)]^3}. \end{aligned} \tag{44}$$

To ensure the satisfaction of the relation $\delta u_j = O(h^3)$ we require that the parameters $A_0, A_1, \alpha_1, \beta_{10}$ satisfy the equations $r_1 = 0, r_{21} = r_{22} = 0$. We will assume that $\varphi(x)$ is an odd function. Then we obtain the following determining equations for parameters $A_0, A_1, \alpha_1, \beta_{10}$ from (44):

$$A_0 + A_1 = \frac{1}{\varphi'(0)}; \quad \alpha_1 A_1 h = \frac{\varphi(h)}{2[\varphi'(0)]^2}; \quad A_1\beta_{10} = \frac{1}{2[\varphi'(0)]^2}. \tag{45}$$

The number of these equations is less than the number of unknowns. Following [10], let us express $A_0, \alpha_1, \beta_{10}$ as the functions of parameter A_1:

$$A_0 = \frac{1}{\varphi'(0)} - A_1; \quad \alpha_1 = \frac{\varphi(h)}{2[\varphi'(0)]^2 A_1 h}; \quad \beta_{10} = \frac{1}{2A_1[\varphi'(0)]^2}. \tag{46}$$

We will call the family of schemes (36),(37),(46) the LB2 method. In the particular case when $\varphi(x) = x$, equalities (46) go over to the equalities obtained in [10] for the classical explicit second-order Runge–Kutta method.

We can now try to find A_1 from the requirement of the minimization of the term r_3 in (42). To this end we substitute in $r_{3,1}$ and $r_{3,2}$ the expressions for α_1 and β_{10} and require that $r_{3,1} = r_{3,2} = 0$. We obtain from here the following expression for A_1: $A_1 = 3/[4\varphi'(0)]$. At this value of A_1 we also obtain that $r_{33} = 0$, and then, in the case of the odd function $\varphi(x)$, the expression for r_3 simplifies to the form: $r_3 = r_{3,0}f + r_{35}f_u^2 f + r_{38}f_x f_u$.

In the case of the classical Runge–Kutta second-order method and $A_1 = 3/4$, the local error has the form [10] $r_{3,RK2} = (h^3/6)f_u(f_x + ff_u)$. Consider the ratio $r_3\varphi^3(h)/r_{3,RK2}$. Then, as in the case of the LB1 method, we arrive at the conclusion that the above-described family of the second-order methods of the Runge–Kutta type has under the satisfaction of inequality (27) a lower local error than the classical Runge–Kutta method with $A_1 = 3/4$.

Returning to the second-order method being optimal in terms of accuracy (36),(37),(46) with $A_1 = 3/[4\varphi'(0)]$, we can write the computational formulas of this method in the form:

$$g_0 = \varphi(h)f(x,u), \;\; g_1 = \varphi(h)f\left(x + \frac{2\varphi(h)}{3\varphi'(0)}, u + \frac{2}{3\varphi'(0)}g_0\right), \;\; \Delta u_h = \frac{g_0 + 3g_1}{4\varphi'(0)}. \tag{47}$$

The stability polynomial $R(z)$ for the second-order scheme under consideration has the following form with regard for (45) and the notation (20)

$$R(z) = 1 + (A_0 + A_1)\frac{\varphi(h)}{h}z + A_1\left[\frac{\varphi(h)}{h}\right]^2\beta_{10}z^2 = 1 + \gamma z + \frac{1}{2}\gamma^2 z^2. \tag{48}$$

Note that at $\gamma < 1$ the coefficients of polynomial (48) affecting z^k decrease with increasing k faster than in the case of the classical second-order Runge–Kutta method (for which $\gamma = 1$). As was pointed out in [9], this is a prerequisite for an increased interval of absolute stability on the negative real axis. We obtain from inequality $|R(z)| \le 1$ the following condition of the absolute stability of the LB2 method on the negative real axis:

$$-(2/\gamma) \le \mathrm{Re}\,(\lambda h) \le 0. \tag{49}$$

For example, at $\gamma = 0.1$ we obtain a stability interval, which is ten times larger than in the case of the classical second-order Runge–Kutta method.

As an example let us consider the system [15]:

$$d\mathbf{u}/dt = J\mathbf{u}, \tag{50}$$

where t is the time, J is the 2×2 matrix,

$$J = \begin{pmatrix} -1000 & 999 \\ 1 & -2 \end{pmatrix}. \tag{51}$$

The eigenvalues of matrix J are as follows: $\lambda_1 = -1001$, $\lambda_2 = -1$. Thus, system (50),(51) is the example of a moderately stiff problem. The solution $\mathbf{u} = (n_1, n_2)$ of system (50), (51) has the form:

$$\begin{aligned} n_1(t) &= 0.999(n_1(0) - n_2(0))e^{-1001t} + (0.001n_1(0) + 0.999n_2(0))e^{-t}, \\ n_2(t) &= -0.001(n_1(0) - n_2(0))e^{-1001t} + (0.001n_1(0) + 0.999n_2(0))e^{-t}. \end{aligned} \tag{52}$$

Dividing the first equation of system (50), (51) by 10^3, we obtain

$$10^{-3}\frac{dn_1}{dt} = -n_1 + 0.999n_2, \tag{53}$$

$$\frac{dn_2}{dt} = n_1 - 2n_2. \tag{54}$$

Equation (53) contains a small parameter $\varepsilon = 10^{-3}$. The degenerate equation corresponding to equation (53) is obtained in the limit as $\varepsilon \to 0$:

$$-n_1 + 0.999n_2 = 0. \tag{55}$$

We have from (55):

$$n_1 = 0.999n_2. \tag{56}$$

Substituting this value in (54), we obtain the ordinary differential equation

$$dn_2/dt = -1.001n_2. \tag{57}$$

It is obvious that $\lambda_1 = -1.001 < 0$. The condition $\operatorname{Re}\lambda_j < 0$ is sufficient for the stability of the solution of the degenerate system. For a sufficiently small parameter ε, the tangents to the integral curves are nearly parallel with the n_1-axis. That is the gradient $|dn_1/dt|$ is large. The integral curve $n_1 = n_1(t)$ has two intervals of a different behavior. The first interval with a rapid change of the sought function shows that the integral curve tends to the graph of the function $\bar{n}_1 = \bar{n}_1(t)$, which was obtained from the solution of the degenerate system (56), (57). This interval is called the boundary layer.

In the second interval the solution derivatives are much less, and the integral curve practically coincides with the graph of $\bar{n}_1(t)$.

Let us return to solution (52). Inside the boundary layer the variable n_1 behaves much more actively than n_2. Therefore, $n_1(t)$ is sometimes called the fast component, and $n_2(t)$ is called the slow component. After the passage of the boundary layer the derivatives of the solution vector are relatively small and are determined by the exponential function with exponent λ_2.

We have carried out the numerical computations of problem (50)–(51) by the LB2 method (47) and, for comparison, by the classical Runge–Kutta method obtained from (47) in the particular case when $\varphi(x) = x$:

$$g_0 = hf(x, u),\; g_1 = hf\left(x + 2h/3, u + 2g_0/3\right),\; \Delta u_h = (g_0 + 3g_1)/4. \qquad (58)$$

In the case of system (50),(51), u is a column vector, $u = (n_1, n_2)^T$, where T is the transposition operation. We will call method (58) the RK2 method. The computations by the both methods were carried out on the same uniform grid in the interval $0 \le t \le 0.2$. Since the optimal expressions for the error terms in (42) were obtained for the case when the function $\varphi_j(x)$ belongs to the class of odd functions, we have used in our numerical computations by the LB2 method the odd function $\varphi_j(x)$ defined by equation (28). The grid step was specified with regard for (49) by formula

$$h = 2\theta/\lambda_{\max}, \qquad (59)$$

where θ is the safety factor, $0 < \theta \le 1$, $\lambda_{\max} = \max(|\lambda_1, |\lambda_2|) = 1001$.

As our computations of the model problem (50), (51) showed, the accuracy of the numerical solution was especially sensitive to the variation of b_1 in (28). We have implemented the algorithm for the choice of b_1, which was based on the inequalities (35). The coefficient b_1 was computed by formula $b_1 = -(\theta_1/4)\lambda_{\max}^2$, where the factor θ_1 was chosen within the interval $0 < \theta_1 < 1$, see Table 2.

For the purpose of executing the computations by both methods on the same grid we have not implemented any strategy of the computation of the variable step $h_j = t_{j+1} - t_j$, $t_j = jh$ (some of these strategies are mentioned, for example, in [7,12]). Let $n_1^{ex}(t)$, $n_2^{ex}(t)$ be the exact solution (52) of system (50),(51). Let us introduce the absolute local errors δn_{1j}, δn_{2j} by formulas:

$$\delta n_{kj} = |n_{kj} - n_{kj}^{ex}|, \quad k = 1, 2, \quad j = 0, \ldots, N, \qquad (60)$$

as well as the grid analogs of the L_2 norm of the numerical solution errors:

$$\Delta n_k = ||n_k - n_k^{ex}||_2 = \left[\frac{1}{t_N - t_0}\sum_{j=0}^{N-1}(n_{kj} - n_{kj}^{ex})^2(t_{j+1} - t_j)\right]^{0.5},\; k = 1, 2. \quad (61)$$

When using the value $\theta = 0.8$ in (59) the following values of the errors (61) were obtained by the RK2 method (58): $\Delta n_1 = 4.11 \cdot 10^{-2}$, $\Delta n_2 = 8.89 \cdot 10^{-5}$.

Table 1. The influence of the coefficient b_1 on the accuracy of the LB2 method

b_1	-10^4	$-5 \cdot 10^4$	$-7.5 \cdot 10^4$	-10^5
Δn_1	$3.77 \cdot 10^{-2}$	$3.00 \cdot 10^{-2}$	$2.89 \cdot 10^{-2}$	$3.02 \cdot 10^{-2}$
Δn_2	$1.98 \cdot 10^{-3}$	$1.03 \cdot 10^{-2}$	$1.56 \cdot 10^{-2}$	$2.09 \cdot 10^{-2}$

Table 1 presents the values of errors (61) at a fixed value $b = 4$ in (28), but at different b_1. It is seen from this table that the error Δn_1 at first drops with increasing $|b_1|$ and reaches its minimum at $b_1 \approx -7.5 \cdot 10^4$, and it remains for all b_1 lower than in the case of the classical RK2 method. At the same time, the error Δn_2 is much higher than in the case of the RK2 method.

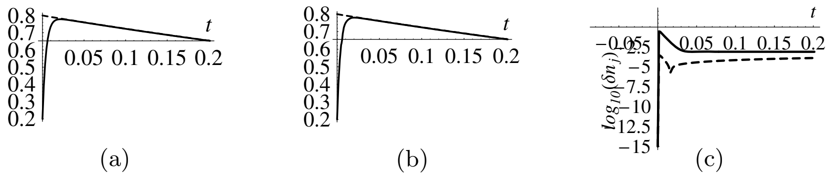


Fig. 1. Numerical solution of problem (50),(51): (a) the RK2 method (58); (b) the LB2 method (47) at $b = 4$, $b_1 = -10^4$ in (28); (c) the local errors (60) obtained at the computation by the RK2 method

The functions n_1 and n_2 are shown in Fig. 1, (a),(b) by the solid and dashed lines, respectively. The error δn_1 is shown in Fig. 1, (c) by the solid line, and the error δn_2 is shown by the dashed line.

For the purpose of a further increase in the accuracy of the LB2 method (47) we now consider its following minor modification. In the case of the odd function $\varphi(x)$ we have: $\varphi(h) = \varphi'(0)h + O(h^3)$. Then $\varphi'(0) = \frac{\varphi(h)}{h} + O(h^2)$. Let us replace $\varphi'(0)$ in the expression for Δu_h in (47) by formula $\varphi'(0) = \varphi(h)/h$:

$$g_0 = \varphi(h)f(x,u),\ g_1 = \varphi(h)f\Big(x + \frac{2\varphi(h)}{3\varphi'(0)}, u + \frac{2}{3\varphi'(0)}g_0\Big),\ \Delta u_h = \frac{(g_0 + 3g_1)h}{4\varphi(h)}. \tag{62}$$

It is clear that such a substitution does not reduce the order of local approximation of the LB2 method. We will call the method (62) the LB2M method, that is the modified LB2 method.

Table 2. The influence of the coefficient b_1 on the accuracy of the LB2M method

b_1	-10^4	$-5 \cdot 10^4$	-10^5	$-1.47 \cdot 10^5$	$-2 \cdot 10^5$
Δn_1	$3.66 \cdot 10^{-2}$	$2.24 \cdot 10^{-2}$	$9.60 \cdot 10^{-3}$	$8.10 \cdot 10^{-4}$	$9.61 \cdot 10^{-4}$
Δn_2	$8.85 \cdot 10^{-5}$	$9.02 \cdot 10^{-5}$	$9.62 \cdot 10^{-5}$	$1.04 \cdot 10^{-4}$	$1.13 \cdot 10^{-4}$

Table 2 presents the values of errors (61) at a fixed value $b = 4$ in (28), but at different b_1. It is seen from this table that the slow solution component n_2 is computed with the same accuracy as at the use of the classical RK2 method. The accuracy of the computation of the fast component n_1 is much higher than in the case of the RK2 method. It follows from Table 2 that the error Δn_1 reaches its minimum at $b_1 = -1.47 \cdot 10^5$, and it is then about 50 times lower than in the case of the RK2 method.

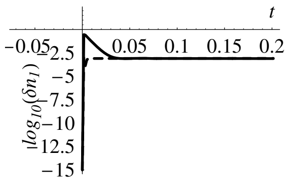


Fig. 2. Local error δn_1 for the RK2 method (solid line) and the LB2M method (dashed line) at $b = 4$, $b_1 = -1.47 \cdot 10^5$ in (28)

Figure 2 shows the local error δn_1 for the cases of the RK2 and LB2M methods. It can be seen that the both errors practically coincide outside the boundary layer. In the boundary layer region, the behavior of the local error is monotonous in the case of the LB2M method in contrast to the RK2 method, and the integral gain of the LB2M method in terms of the integral error Δn_1 is obtained right at the expense of a low local error of this method inside the boundary layer.

In [16], the package `OrderStar` was presented, which enables the obtaining of the graphical image of the regions of the absolute stability of both explicit and implicit methods for numerical integration of ODEs. The application of the package `OrderStar` to the RK2 method and to the LB2M method (at $b = 4$, $b_1 = -1.47 \cdot 10^5$ in (28)) has been implemented with the use of the *Mathematica* commands presented in [16].

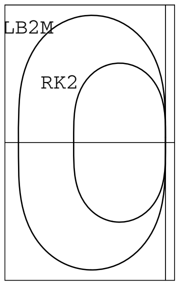


Fig. 3. Regions of the absolute stability of the RK2 and LB2M methods

Figure 3 shows the obtained regions of absolute stability. The absolute stability region of the LB2M method can be seen to be much larger than the absolute stability region of the RK2 method. The absolute stability interval of the LB2 and LB2M methods on the negative real axis increases with increasing $|b_1|$.

We now would like to discuss a number of aspects in connection with the computer implementation of the above-proposed methods LB2 and LB2M. First of all, these methods need the information about the local value of the eigenvalue of the Jacobi matrix J, which is largest in absolute value, so that the quantities $h_j\lambda_i$ for all i, for which $\mathrm{Re}(\lambda_i) < 0$, must lie inside the region of absolute stability [13]. This need is caused by the fact that the proposed methods are explicit, therefore, their regions of absolute stability are finite (although they may be much larger than in the case of the classical explicit Runge–Kutta methods). On the other hand, there are at present several numerical algorithms, which enable the determination of the largest eigenvalue of a matrix without computing all eigenvalues. For example, the call to the built-in *Mathematica* function `Eigenvalues[J,1]` gives the eigenvalue that is largest in absolute value. In the case of a big ODE system to be solved, the computation of the largest eigenvalue may be carried out not at every node t_j to reduce the amount of computational work [13]. Knowing the largest eigenvalue one can use automatic step size control in the computer codes implementing the Lagrange–Burmann methods.

Besides, one can estimate the local error of the proposed methods LB2 and LB2M with the aid of the Richardson's extrapolation similarly to the case of the classical Runge–Kutta methods [7].

And, finally, there is a question on the optimal choice of the form of the function $\varphi_j(x)$. The specification of this function in the form of a polynomial involving only the odd degrees of the variable x appears to be advantageous due to the fact that the coefficients of polynomial $\varphi_j(x)$ will then enter explicitly the expressions for the local error of the Lagrange–Burmann method (see, e.g., formulas (44)). For example, at the derivation of the one-step Lagrange–Burmann fourth-order method, the derivative $\varphi_j^{(5)}(0)$ will enter the local error formulas, and it will be advisable to take $\varphi_j(x)$ in the form $\varphi_j(x) = b_0x + b_1x^3 + b_2x^5$.

References

1. Abdulle, A.: Chebyshev methods based on orthogonal polynomials. Ph.D. Doctoral Dissertation No. 3266, Dept. Math., Univ. of Geneva (2001)
2. Abdulle, A.: Fourth order Chebyshev methods with recurrence relation. SIAM J. Sci. Comput. 23, 2041–2054 (2002)
3. Abramowitz, M., Stegun, I.A.: Hanbook of Mathematical Functions. National Bureau of Standards (1964)
4. Butcher, J.C.: The Numerical Analysis of Ordinary Differential Equations. Wiley, Chichester (1987)
5. Consul, P.C., Famoye, F.: Lagrangian Probability Distributions. Birkhäuser, Basel (2006)
6. Dahlquist, G.: A special stability problem for linear multistep methods. BIT 3, 27–43 (1963)
7. Hall, G., Watt, J.M. (eds.): Modern Numerical Methods for Ordinary Differential Equations. Clarendon Press, Oxford (1976)
8. van der Houwen, P., Sommeijer, B.: On the internal stability of explicit, m-stage Runge-Kutta methods for large m-values. Z. Angew. Math. Mech. 60, 479–485 (1980)
9. Hundsdorfer, W., Verwer, J.: Numerical Solutions of Time-Dependent Advection-Diffusion-Reaction Equations, 2nd edn. Springer, Heidelberg (2007)
10. Krylov, V.I., Bobkov, V.V., Monastyrnyi, P.I.: Numerical Methods, Nauka, Moscow, vol. II (1977) (in Russian)
11. Lavrentiev, M.A., Shabat, B.V.: Methods of the Theory of Functions of Complex Variable, 4th edn., Nauka, Moscow (1973) (in Russian)

12. Martin-Vaquero, J., Janssen, B.: Second-order stabilized explicit Runge–Kutta methods for stiff problems. Computer Phys. Communications 180, 1802–1810 (2009)
13. Oran, E.S., Boris, J.P.: Numerical Simulation of Reactive Flow. Elsevier, New York (1987)
14. Pinchukov, V.I., Shu, C.-W.: Numerical Methods of Higher Orders for Fluid Dynamics Problems. Published by the Siberian Branch of the Russian Acad. Sci., Novosibirsk (2000) (in Russian)
15. Rakitskii, Y.V., Ustinov, S.M., Chernorutskii, I.G.: Numerical Methods for Solving Stiff Systems, Nauka, Moscow (1979) (in Russian)
16. Sofroniou, M.: Order stars and linear stability theory. J. Symbolic Computation 21, 101–131 (1996)
17. Strampp, W., Ganzha, V., Vorozhtsov, E.: Höhere Mathematik mit Mathematica. In: Band 3: Differentialgleichungen und Numerik. Vieweg, Braunschweig/Wiesbaden (1997)
18. Vorozhtsov, E.V.: Construction of difference schemes for hyperbolic conservation laws with the aid of the Lagrange–Bürmann expansions. In: Mikhailov, G.A., Ilyin, V.P., Laevsky, Y.M. (eds.) Proc. Int. Conf. on Numerical Mathematics, Pt. I. Price-Courier, Novosibirsk, pp. 443–448 (2004) (in Russian)

Parametric Qualitative Analysis of Ordinary Differential Equations: Computer Algebra Methods for Excluding Oscillations (Extended Abstract) (Invited Talk)

Andreas Weber[1], Thomas Sturm[2], Werner M. Seiler[3], and Essam O. Abdel-Rahman[1]

[1] Institut für Informatik II, Universität Bonn, Römerstr. 164, 53117 Bonn, Germany
{weber,essam}@cs.uni-bonn.de

[2] Departamento de Matemáticas, Estadística y Computación, Facultad de Ciencias, Universidad de Cantabria, 39071 Santander, Spain
sturmt@unican.es

[3] Institut für Mathematik, Universität Kassel, Heinrich-Plett-Straße 40 34132 Kassel, Germany
seiler@mathematik.uni-kassel.de

1 Introduction

Investigating oscillations for parametric ordinary differential equations (ODEs) has many applications in science and engineering but is a very hard problem. Already for two dimensional polynomial systems this question is related to Hilbert's 16th problem, which is still unsolved [1].

Using the theory of Hopf-bifurcations some non-numeric algorithmic methods have been recently developed to determine ranges of parameters for which some small stable limit cycle will occur in the system [2,3,4,5,6,7,8]. These algorithms give exact conditions for the existence of fixed points undergoing a Poincaré-Andronov-Hopf bifurcation that give birth to a small stable limit cycle under some general conditions which can be made algorithmic, too. If these conditions are not satisfied, one can be sure that there are no such fixed points, but unfortunately one cannot conclude that there are no limit cycles—which could arise by other means. Nevertheless, it is tempting to conjecture even in these cases that there are no oscillations, as has been done e.g. in [5,6].

However, the ultimate goal of finding *exact algorithmic conditions* for the existence of oscillations, i. e. for determining for which parameter values there are non-constant limit cycles for a given system of parametric ordinary differential equations is a major challenge, so that considerable work has been spent—and will be also be invested in the future—for investigating sub-problems.

In this paper we deal with computer algebra methods for some of these sub-problems. Our techniques will be along the line of work reducing problems on the qualitative analysis of ordinary differential equations to semi-algebraic problems. This possibility might seem to be surprising on first sight, as even the description of flows induced by the simplest linear ordinary differential equations involves exponential functions. However, a significant part of the study of the qualitative behavior of differential equations can be done in the realm of algebraic or semi-algebraic sets: Starting from the rather trivial observation that for polynomial vector fields the study of the equilibria of the vector field is purely algebraic, also questions of the stability of the equilibria can in general be reduced to decidable questions on semi-algebraic sets (for polynomial vector fields) via the well known Routh-Hurwitz criterion [9]). Also the parametric question (for a parameterized polynomial vector field) whether fixed points undergo a *Hopf bifurcation* is not only known to be decidable but also lies within the realm of semi-algebraic sets [8,10,3].

We review some recently developed criteria which give sufficient conditions to exclude oscillations by reducing them to problems on semi-algebraic sets—for polynomial vector fields—we will give some examples and we will discuss possible future work in the form of problems to be solved. Some of these problems might be rather immediate to be solved, some others might pose major challenges.

2 Preliminaries

2.1 The Bendixson-Dulac Criterion for 2-Dimensional Vector Fields

Consider an autonomous planar vector field

$$\frac{dx}{dt} = F(x, y), \qquad \frac{dy}{dt} = G(x, y), \qquad (x, y) \in \mathbb{R}^2.$$

Bendixson in 1901 [11] was the first to give a criterion yielding sufficient conditions for excluding oscillations. For a modern proof we refer to [12, Theorem 1.8.2].

Theorem 1 (Bendixson's criterion). *If* $\operatorname{div}(F, G) = \frac{\partial(F)}{\partial x} + \frac{\partial(G)}{\partial y}$ *is not identically zero and does not change sign on a simply connected region* $D \subseteq \mathbb{R}^2$*, then* (F, G) *has no closed orbits lying entirely in* D*.*

Dulac in 1937 [13] was able to generalize the result of Bendixson as follows:

Theorem 2 (Dulac's criterion). *Let* $B(x, y)$ *be a scalar continuously differentiable function defined on a simply connected region* $D \subset \mathbb{R}^2$ *with no holes in it. If* $\frac{\partial(BF)}{\partial x} + \frac{\partial(BG)}{\partial y}$ *is not identically zero and does not change sign in* D*, then there are no periodic orbits lying entirely in* D*.*

Dulac's criterion is a generalization of Bendixson's criterion, which corresponds to $B(x, y) = 1$.

2.2 Quantifier Elimination and Positive Quantifier Elimination over the Ordered Field of the Reals

In order to summarize the basic idea of real quantifier elimination, we introduce first-order logic on top of polynomial equations and inequalities.

We consider multivariate polynomials $f(u, x)$ with rational coefficients, where $u = (u_1 \ldots, u_m)$ and $x = (x_1, \ldots, x_n)$. We call u *parameters* and we call x *variables*. Equations will be expressions of the form $f = 0$, inequalities are of the form $f \leq 0$, $f < 0$, $f \geq 0$, $f > 0$, or $f \neq 0$. Equations and inequalities are called *atomic formulae*. *Quantifier-free formulae* are Boolean combinations of atomic formulae by the logical operators "$\wedge$," "$\vee$," and "$\neg$." *Existential formulae* are of the form $\exists x_1 \ldots \exists x_n \psi(u, x)$, where ψ is a quantifier-free formula. Similarly, *universal formulae* are of the form $\forall x_1 \ldots \forall x_n \psi(u, x)$. A general (prenex) *first-order formula* has several alternating blocks of existential and universal quantifiers in front of a quantifier-free formula.

The real *quantifier elimination problem* can be phrased as follows: Given a formula φ, find a quantifier-free formula φ' such that both φ and φ' are equivalent in the domain of the real numbers. A procedure computing such a φ' from φ is called a real *quantifier elimination procedure*.

Although real quantifier elimination is known to be a computationally hard problem [14,15], there has been considerable and quite successful research on efficient implementations during the past decades, which has resulted in three major systems:

1. The commercial computer algebra system Mathematica includes an efficient implementation of CAD-based real quantifier elimination by Strzebonski [16,17], the development of which started around 2000.
2. QEPCAD B [18], which implements partial cylindrical algebraic decomposition (CAD). The development of QEPCAD B started with the early work of Collins and his collaborators on CAD around 1973 and continues until today. QEPCAD B is supplemented by another software called SLFQ for simplifying quantifier-free formulas using CAD. Both QEPCAD B and SLFQ are freely available.[1]
3. REDLOG[2] [19,20], which had been originally driven by the efficient implementation of quantifier elimination based on virtual substitution methods [14,21,22]. Meanwhile REDLOG includes also CAD and Hermitian quantifier elimination [23,24,25] for the reals as well as quantifier elimination for various other domains [26] including the integers [27,28]. The development of REDLOG has been started in 1992 by one of the authors (T. Sturm) of this paper and continues until today. REDLOG is included in the computer algebra system REDUCE, which is open source.[3]

Besides regular quantifier elimination methods for the reals, REDLOG includes several variants of quantifier elimination. This includes in particular *extended quantifier elimination* [29], which yields in addition sample solutions for existential quantifiers, and *positive quantifier elimination* [4,2], which includes powerful simplification techniques based on the knowledge that all considered variables are restricted to positive values.

1 `http://www.usna.edu/Users/cs/qepcad/B/QEPCAD.html`
2 `http://www.redlog.eu/`
3 `http://reduce-algebra.sourceforge.net/`

As in many applications the region of interest is the positive cone of the state variables, and also the parameters of interest are known to be positive, the positive quantifier elimination is of special importance and will be used for many of the examples given below.

3 Some Algorithmic Global Criteria for Excluding Oscillations

The algorithmic criteria discussed in the following can be seen as generalizations of the Bendixson-Dulac criterion for 2-dimensional vector fields to arbitrary dimensions.

3.1 Muldowney's Criteria

The following theorem was proved by Muldowney [30, Theorem 4.1]: Suppose that one of the inequalities

$$\mu\left(\frac{\partial f^{[2]}}{\partial x}\right) < 0, \qquad \mu\left(-\frac{\partial f^{[2]}}{\partial x}\right) < 0 \tag{1}$$

holds for all $x \in \mathbb{R}^n$. Then the autonomous system with vector field $f : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ has no nonconstant periodic solutions. Here μ is some Lozinskiĭ norm and $f^{[2]}$ is one of the "compound matrices" of the Jacobian of the vector field f defined in [30]. As as also shown in [30] the criterion given in [30, Theorem 4.1] also holds when $x \in C$, where $C \subseteq \mathbb{R}^n$ is open and convex.

Remark. When $n = 2, \partial f^{[2]}/\partial x = \mathrm{Trace}\, \partial f/\partial x = \mathrm{div} f$, so that [30, Theorem 4.1] gives the results of Bendixson, i.e. the criterion of Muldowney can be seen as a generalization of the criterion of Bendixson from the planar case to arbitrary dimensions.

According to [30, (2.2)], any of the following expressions may be used as $\mu\left(\partial f^{[2]}/\partial x\right)$ in [30, Theorem 4.1].

$$\max\left\{ \frac{\partial f_r}{\partial x_r} + \frac{\partial f_s}{\partial x_s} + \sum_{q \neq r,s} \left|\frac{\partial f_q}{\partial x_r}\right| + \left|\frac{\partial f_q}{\partial x_s}\right| \;:\; r,s = 1,\ldots,n,\ r \neq s \right\}, \tag{2}$$

$$\max\left\{ \frac{\partial f_r}{\partial x_r} + \frac{\partial f_s}{\partial x_s} + \sum_{q \neq r,s} \left|\frac{\partial f_r}{\partial x_q}\right| + \left|\frac{\partial f_s}{\partial x_q}\right| \;:\; r,s = 1,\ldots,n,\ r \neq s \right\}. \tag{3}$$

Thus for a formula γ over the reals defining an open convex subset C of $\mathbb{R}^n$ and an autonomous polynomial vector field $f : \mathbb{R}^n \to \mathbb{R}^n$ the following first-order formula over the real closed field defines a sufficient condition such that the vector field defined by f has no non-constant periodic solution on C:

$$\begin{aligned}
\varphi \equiv \forall x_1 \forall x_2 \cdots \forall x_n \Bigg(\gamma \Longrightarrow \qquad\qquad (4)\\
\max\left\{ \frac{\partial f_r}{\partial x_r} + \frac{\partial f_s}{\partial x_s} + \sum_{q \neq r,s} \left|\frac{\partial f_q}{\partial x_r}\right| + \left|\frac{\partial f_q}{\partial x_s}\right| \, : \, r,s = 1,\ldots,n, \, r \neq s \right\} < 0 \Bigg)\\
\vee \forall x_1 \forall x_2 \cdots \forall x_n \Bigg(\gamma \Longrightarrow\\
\max\left\{ -\frac{\partial f_r}{\partial x_r} - \frac{\partial f_s}{\partial x_s} + \sum_{q \neq r,s} \left|\frac{\partial f_q}{\partial x_r}\right| + \left|\frac{\partial f_q}{\partial x_s}\right| \, : \, r,s = 1,\ldots,n, \, r \neq s \right\} < 0 \Bigg)\\
\vee \forall x_1 \forall x_2 \cdots \forall x_n \Bigg(\gamma \Longrightarrow\\
\max\left\{ \frac{\partial f_r}{\partial x_r} + \frac{\partial f_s}{\partial x_s} + \sum_{q \neq r,s} \left|\frac{\partial f_r}{\partial x_q}\right| + \left|\frac{\partial f_s}{\partial x_q}\right| \, : \, r,s = 1,\ldots,n, \, r \neq s \right\} < 0 \Bigg)\\
\vee \forall x_1 \forall x_2 \cdots \forall x_n \Bigg(\gamma \Longrightarrow\\
\max\left\{ -\frac{\partial f_r}{\partial x_r} - \frac{\partial f_s}{\partial x_s} + \sum_{q \neq r,s} \left|\frac{\partial f_r}{\partial x_q}\right| + \left|\frac{\partial f_s}{\partial x_q}\right| \, : \, r,s = 1,\ldots,n, \, r \neq s \right\} < 0 \Bigg).
\end{aligned}$$

The maximum and absolute value functions are included in the language of ordered rings as it is commonly used for real quantifier elimination. They are, however, definable.

In [31] the problem of efficient automatic resolution of maxima and absolute values is addressed and computation examples are given.

3.2 An Algorithmic Global Criterion Excluding Oscillations Based on Algebraic First-Integrals

Computing algebraic first-integrals. Another algorithmic method to parametrically investigate the absence of oscillations relies on the possibility to compute algebraic first integrals of polynomial vector fields. Those do not necessarily exist, but if there exists such first integrals up to a certain degree these can be computed by the following method described in the book by Goriely [32].

Consider an n-dimensional polynomial vector field $\mathbf{G}$ of degree d. In general, this vector field may depend on a certain number of parameters, say $(\nu_1, \ldots, \nu_p)$. The problem consists of finding the values of $(\nu_1, \ldots, \nu_p)$ such that the vector field admits a time-independent polynomial first integral of a given degree $\mathbf{D}$.

1. Start with $\mathbf{D} = 1$.
2. Consider the most general form of a polynomial first integral of degree $\mathbf{D}$

$$\mathbf{I}(\mathbf{x}) = \sum_{i,|i|=1}^{|i|=D} c_i \mathbf{x}^i. \qquad (5)$$

3. Compute the time derivative of $\mathbf{I}(\mathbf{x})$.

$$\delta_{\mathbf{G}}\mathbf{I} = \sum_{i,|i|=0}^{|i|=D+d-1} \mathbf{Q}_i \mathbf{x}^i. \tag{6}$$

4. Since we are looking for $\mathbf{I}$ such that $\delta_{\mathbf{G}}\mathbf{I} = 0$, we have $\mathbf{Q}_i = 0$. This system of equations is a linear system for the coefficients c_i of dimension at most $\begin{pmatrix} n+d+D-1 \\ n \end{pmatrix}$. So, if there exist values of the parameter $(\nu_1, \dots, \nu_p)$ and a set of constants c_i that are not all zero, such that $\mathbf{Q}_i = 0$ for all i, then $\mathbf{I}(\mathbf{x})$ is a first integral. Otherwise increase $\mathbf{D}$ by 1 and return to Step 2.

Notice that the linear system of equations constructed above are under determined in general, so that several different first integrals might arise when solving these systems.

A generalization of the Bendixson-Dulac criterion involving first-integrals. For our algorithmic criteria we use the following generalization of the Bendixson-Dulac criterion for 2d-vector-fields to arbitrary dimensions proved by Tóth [33, Theorem 3.1]:

Theorem 3. *Let $M \in \{2, 3, 4, \dots\}$ and let $T \subset \mathbb{R} \times \mathbb{R}^{M-1}$ be a domain such that for all $\bar{x} \in \mathbb{R}$ the set*

$$\Im(\bar{x}) := \{y \in \mathbb{R}^{M-1} \mid (\bar{x}, y) \in T\}$$

is convex. *Let $J : T \longrightarrow \mathbb{R}^M$ be continuous and suppose that there exists a sufficiently smooth function $P = (P_1, , ..., P_{M-1}) : T \longrightarrow \mathbb{R}^{M-1}$ such that its coordinate functions are (global) first integrals of the equation*

$$\dot{x} = J \circ x, \tag{7}$$

and let us suppose that for all $\bar{x} \in \mathbb{R}$ and $y^1, ..., y^{M-1} \in \Im(\bar{x})$

$$\begin{vmatrix} \partial_2 P_1(\bar{x}, y^1) \\ \vdots \\ \partial_2 P_{M-1}(\bar{x}, y^{M-1}) \end{vmatrix} \neq 0 \tag{8}$$

Then the differential equation (7) has no periodic solution.

If the convex set $\Im(\bar{x})$ is semi-algebraic and one can compute sufficiently many algebraic first-integrals then Theorem 3 yields a quantifier elimination problems over the ordered field of the reals.

4 Computation Examples

We have extended the Maple library QEHOPFLIB[4] implementing the methods in [8] by the following algorithmic methods:

[4] http://cg.cs.uni-bonn.de/project-pages/symbolicanalysis/

1. Algorithms to produce from a system of ordinary differential equations essentially a formula corresponding to $\varphi_{\text{Muldowney}}$ as described by Equation (4). The formula can be generated for arbitrary vector fields, its symbolic analysis by quantifier elimination over the reals is only possible for systems of differential equations with a vector field described by parameterized multivariate rational functions.
2. Algorithms trying to compute algebraic first integrals up to a degree bound and to produce from a system of ordinary differential equations essentially a formula corresponding to the universally quantified condition in Theorem 3. Let us call this formula φ_{Toth}. The formula can be generated only for vector fields of dimension n which have at least $n-1$ algebraic first integrals up to the used degree bound d. Notice that the vector field can be described by parameterized multivariate rational functions.

In the examples discussed below we use the positive cone of the real n-space as convex subset, or the entire real n-space.

Actually, we produce the logical negation $\neg\varphi$ of φ rather than φ itself (for $\varphi_{\text{Muldowney}}$ as well as for φ_{Toth}), since the implementation of positive quantifier elimination in the current stable branch of REDLOG is restricted to existential formulas. It is, however, not hard to see that applying positive quantifier elimination to $\neg\varphi$ yielding, say, ψ and then positively simplifying $\neg\psi$, which involves re-adding the positivity conditions on all variables, yields exactly the desired result of applying to φ quantifier elimination subject to positivity assumptions on all variables. We are usually going to refer to this final result as φ'.

4.1 A Non-parametric Example

As a first simple example we take the following simple reaction that was already studied in [33]:

Simple reaction system: It is the induced kinetic differential equation (cf. [34]) of the reaction

$$3\,\mathrm{OH} \xrightarrow{1} \mathrm{H_2O} + \mathrm{HO_2}, \qquad \mathrm{H_2O} + \mathrm{HO_2} \xrightarrow{1} 3\,\mathrm{OH} \tag{9}$$

i.e.

$$\dot{x} = -3x^3 + 3yz, \qquad \dot{y} = x^3 - yz, \qquad \dot{z} = x^3 - yz \tag{10}$$

where the concentrations of the components are denoted by x, y and z with $x := [\mathrm{OH}], y := [\mathrm{H_2O}], z := [\mathrm{HO_2}]$.

This model does not seem to have oscillatory behavior. Although it does not depend on parameters the question of a *proof* that there are no oscillations for any values of the concentrations of the reactants, i.e. the state variables, is already beyond the scope of pure numerical computations.

Using the undetermined coefficients method one find two first-integrals of degree one algorithmically. The generated first-order formula φ_{Toth} describing the negation of the criterion of Tóth can be reduced by REDLOG to *false* within

```
eq1:=diff(x(t),t)=y(t);
eq2:=diff(y(t),t)=z(t);
eq3:=diff(z(t),t)=-alpha*y(t) - beta*z(t);
fun:={x(t),y(t),z(t)};
par:={alpha,beta};
paramcondlist:={};
DELimitCycleMuldowney({eq1,eq2,eq3},fun,par,[],paramcondlist);
degpoly:=4:; dime:=nops(fun):
DELimitCycleToth({eq1,eq2,eq3},fun,par,[],paramcondlist,degpoly);
```

Fig. 1. A Maple script for generating the first-order formulas $\neg\varphi$ discussed in Section 3.1 and Section 4.2

some milliseconds on a current standard PC. Thus this example can be solved fully algorithmically by the method computing first integrals.

Using the criterion of Muldowney [30, Theorem 4.1] (for the L^1 norm and L^∞ norm) one can also come up easily with a first order-formula (describing the negation of the Muldowney criterion for excluding oscillations). This formula $\varphi_{\text{Muldowney}}$ can be reduced by REDLOG to *true* within a few milliseconds, which unfortunately is the non-conclusive answer: one cannot prove the absence of oscillations in this way—a result not contradicting the result stated above, as the Muldowney criterion is a sufficient but not a necessary condition for the absence of oscillations.

4.2 A Parametric Example

The question whether there are oscillations or not is a parametric question in general. As the generated formula φ' can be parametric and the result of the quantifier elimination procedure will be a condition on the parameters in general—i.e. a first-order formula involving the parameters only.

Simple parametric example: Let us consider the system

$$\dot{x} = y, \qquad \dot{y} = z, \qquad \dot{z} = -\alpha y - \beta z \tag{11}$$

The global criteria excluding the presence of oscillations can be generated by the Maple script in Figure 1.

We find 5 (independent) polynomial first integrals of degree 4 for this system. The generated first-order formula can be shown to be equivalent to *false*—independent of the values of the parameters—by REDLOG within less than 1 sec of computation time using standard quantifier elimination. Thus we can exclude the possibility of oscillations all parameter values.

In the hand computations done in [33] only one linear first-integral is found, so that some further theoretical development was undertaken.

Using the criterion of Muldowney [30, Theorem 4.1] (for the L^1 norm and L^∞ norm) in its negated form the generated formula can be reduced by REDLOG to *true* within a few milliseconds, which unfortunately is again the non-conclusive answer.

```
eq1 := diff(x(t),t) = s - mu*x(t) - beta*x(t)*v(t);
eq2 := diff(y(t),t) = beta*x(t)*v(t) - alpha*y(t);
eq3 := diff(v(t),t) = c*y(t)-gamma1*v(t);
fcns:={x(t),y(t),v(t)};
params:={s,mu,beta,alpha,c,gamma1};
paramcondlist:={s>0,mu>0,beta>0,alpha>0,c>0,gamma1>0};
DELimitCycleMuldowney({eq1,eq2,eq3},
      fcns,params,{},paramcondlist);
od;
```

Fig. 2. A Maple script for generating the first-order formulas $\neg\varphi$ for the three-component model of viral dynamics from [35].

Although this example is simple—in fact it consists of a linear system—the criterion of Muldowney (for the L^1 norm and L^∞ norm) was too weak to give useful insights in this case.

However, in the following much more complex example the situation is different.

4.3 Models of Genetic Circuits

For the family of examples arising out of a simple quasi-steady state approximation of a model of genetic circuits investigated in [5] the Muldowney criteria in its realization of the framework of [31] can proof the absence of oscillations for several relevant values of parameters. We refer to [5,31] for an exposition of the models and to [31] for the results.

Using the first-integral based method described in this paper we could not come up with any conclusive result for any of the examples from [5].

4.4 A Model of Viral Dynamics

The following example is also discussed in more depth in [31]. It consists of a simple mathematical model for the populations dynamics of the human immunodefficiency type 1 virus (HIV-1) investigated in [35]. There a three-component model is described involving uninfected CD4 + T-cells, infected such cells and free virus, whose densities at time t are denoted by $x(t), y(t), v(t)$.

In [35] a simplified two-component model employed by Bonhoeffer et al. [36] is investigated analytically.

For the two-component model the equilibria are computed analytically for biologically relevant non-negative parameter values and their local stability properties are parametrically investigated in [35]. Moreover, using the general Bendixson-Dulac criteria for 2D-vector fields with an ad hoc Dulac function $B(x, y) = 1/y$ it is shown that there are no periodic solutions for the system for positive parameter values and positive values of the state variables, i.e. the biologically relevant ones.

Using our algorithms, we can easily construct the formula for the Muldowney criteria even for the three-component model, cf. Figure 2—but we could not compute first-integrals for this system. Using REDLOG quantifier elimination and formula simplification of the obtained first-order formula can be performed within some milliseconds. Unfortunately, the obtained result for the negated Muldowney criteria is a non-parametric *true*, i. e. the non-conclusive answer, as they give sufficient conditions for excluding oscillations, but no indication about a necessary condition.

Also when applying our framework to the two-component model, we obtain the non-conclusive *true* within some milliseconds of computation time.

In our framework we can easily use Dulac functions for 2D-cases, too. When using the Dulac function $B(x, y) = 1/y$ for the two-component model, we obtain the conclusive *false* as an answer, i.e. we can prove that there are no oscillations for the two-component model (for any values of the parameters).

So the hand computations using Dulac functions can be widely simplified by our framework—one just has to specify the Dulac function in addition to the vector field. Unfortunately, finding a suitable Dulac function is still a non-algorithmic step—although testing various possibilities is now easily possible.

5 Some Possible Future Directions

In the examples given above sometimes one of the given criteria was successful, sometimes the other one, and very often none of them. So a first problem is the following.

Problem 1. What is the relative strength of the Muldowney criteria for different norms? What are their combined strengths compared to the criteria involving first integrals?

In one of our computation examples (cf. Sect. 4.4) it was necessary to use an appropriate Dulac function in order to come up with a criterion proving the absence of oscillations.

An inspection of the proof of [30, Theorem 4.1] seems to indicate that the answer to the following problem is “yes”.

Problem 2. Are there generalizations of the criterion of Muldowney involving Dulac functions?

In the positive case one might ask how to find appropriate Dulac functions. For polynomial functions (or rational functions) one could use the approach to specify those with undetermined coefficients up to a certain degree—and then use these in the quantifier-elimination step. However, in its naive realization the computational complexity does not only seem to be prohibitive under worst case considerations, but also for most but the most trivial cases. So the following problem occurs:

Problem 3. Are there constructive and efficient ways for generating appropriate Dulac functions for the criterion of Muldowney?

The Bendixson-Dulac criteria are not only generalizable using first-integrals as has been done in [33] or also e.g. in [37], but also to systems with invariant hypersurfaces.

Problem 4. Specify algorithmic methods for excluding oscillations using algebraic invariant hypersurfaces.

A standard technique for excluding oscillations in hand computations is to find Lyapunov functions, which also prohibit the existence of oscillations. As the existence of Lyapunov functions of certain form can also be proven by quantifier elimination techniques [38] the following problem shall be formulated:

Problem 5. Are there constructive and efficient ways for generating appropriate Lyapunov functions? Can these be defined semi-algebraically for polynomial vector fields?

Finally the following problems, which are presumably much more challenging, shall be posed:

Problem 6. For autonomous polynomial vector fields are there algorithmic criteria that are *sufficient and necessary* for excluding oscillations?

All of the questions also generalize to differential algebraic equations [39,40]. Although having an additional "algebraic part" seems to be compatible with the semi-algebraic context, which the qualitative investigations of the ODEs have been reduced to, many new definitional and theoretical problems arise. Of particular interest is here the possibility of various forms of singularities [41] leading for example to singularity induced bifurcations.

Problem 7. Generalize the problems to differential algebraic equations (possibly with singularities).

Acknowledgement

We are grateful to Vladimir Gerdt for several helpful discussions.

References

1. Ilyashenko, Y.: Centennial history of Hilbert's 16th Problem. Bull. Am. Math. Soc., New Ser. 39(3), 301–354 (2002)
2. Sturm, T., Weber, A., Abdel-Rahman, E.O., El Kahoui, M.: Investigating algebraic and logical algorithms to solve Hopf bifurcation problems in algebraic biology. Mathematics in Computer Science, Special issue on 'Symbolic Computation in Biology' 2(3), 493–515 (2009)
3. Niu, W., Wang, D.: Algebraic approaches to stability analysis of biological systems. Mathematics in Computer Science 1(3), 507–539 (2008)
4. Sturm, T., Weber, A.: Investigating generic methods to solve Hopf bifurcation problems in algebraic biology. In: Horimoto, K., Regensburger, G., Rosenkranz, M., Yoshida, H. (eds.) AB 2008. LNCS, vol. 5147, pp. 200–215. Springer, Heidelberg (2008)

5. Boulier, F., Lefranc, M., Lemaire, F., Morant, P., Ürgüplü, A.: On proving the absence of oscillations in models of genetic circuits. In: Anai, H., Horimoto, K., Kutsia, T. (eds.) Ab 2007. LNCS, vol. 4545, pp. 66–80. Springer, Heidelberg (2007)
6. Boulier, F., Lefranc, M., Lemaire, F., Morant, P.E.: Applying a rigorous quasi-steady state approximation method for proving the absence of oscillations in models of genetic circuits. In: Horimoto, K., Regensburger, G., Rosenkranz, M., Yoshida, H. (eds.) AB 2008. LNCS, vol. 5147, pp. 56–64. Springer, Heidelberg (2008)
7. El Kahoui, M., Weber, A.: Symbolic equilibrium point analysis in parameterized polynomial vector fields. In: Ganzha, V.G., Mayr, E.W., Vorozhtsov, E.V. (eds.) Computer Algebra in Scientific Computing (CASC 2002), Yalta, Ukraine, pp. 71–83 (September 2002)
8. El Kahoui, M., Weber, A.: Deciding Hopf bifurcations by quantifier elimination in a software-component architecture. Journal of Symbolic Computation 30(2), 161–179 (2000)
9. Hong, H., Liska, R., Steinberg, S.: Testing stability by quantifier elimination. Journal of Symbolic Computation 24(2), 161–187 (1997)
10. Gatermann, K., Eiswirth, M., Sensse, A.: Toric ideals and graph theory to analyze hopf bifurcations in mass action systems. Journal of Symbolic Computation 40(6), 1361–1382 (2005)
11. Bendixson, I.: Sur les curbes définiés par des équations différentielles. Acta Math. 24, 1–88 (1901)
12. Guckenheimer, J., Holmes, P.: Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields. Applied Mathematical Sciences, vol. 42. Springer, Heidelberg (1983)
13. Dulac, H.: Recherche des cycles limites. CR Acad. Sci. Paris 204, 1703–1706 (1937)
14. Weispfenning, V.: The complexity of linear problems in fields. Journal of Symbolic Computation 5(1&2), 3–27 (1988)
15. Davenport, J.H., Heintz, J.: Real quantifier elimination is doubly exponential. Journal of Symbolic Computation 5(1-2), 29–35 (1988)
16. Strzebonski, A.: Solving systems of strict polynomial inequalities. Journal of Symbolic Computation 29(3), 471–480 (2000)
17. Strzebonski, A.W.: Cylindrical algebraic decomposition using validated numerics. J. Symb. Comput. 41(9), 1021–1038 (2006)
18. Brown, C.W.: QEPCAD B: A system for computing with semi-algebraic sets via cylindrical algebraic decomposition. ACM SIGSAM Bulletin 38(1), 23–24 (2004)
19. Dolzmann, A., Sturm, T.: REDLOG: Computer algebra meets computer logic. ACM SIGSAM Bulletin 31(2), 2–9 (1997)
20. Sturm, T.: Redlog online resources for applied quantifier elimination. Acta Academiae Aboensis, Ser. B 67(2), 177–191 (2007)
21. Weispfenning, V.: Quantifier elimination for real algebra—the quadratic case and beyond. Applicable Algebra in Engineering Communication and Computing 8(2), 85–101 (1997)
22. Dolzmann, A., Sturm, T.: Simplification of quantifier-free formulae over ordered fields. Journal of Symbolic Computation 24(2), 209–231 (1997)
23. Weispfenning, V.: A new approach to quantifier elimination for real algebra. In: Caviness, B., Johnson, J. (eds.) Quantifier Elimination and Cylindrical Algebraic Decomposition. Texts and Monographs in Symbolic Computation, pp. 376–392. Springer, Wien (1998)
24. Gilch, L.A.: Effiziente Hermitesche Quantorenelimination. Diploma thesis, Universität Passau, D-94030 Passau, Germany (September 2003)

25. Dolzmann, A., Gilch, L.A.: Generic Hermitian quantifier elimination. In: Buchberger, B., Campbell, J. (eds.) AISC 2004. LNCS (LNAI), vol. 3249, pp. 80–93. Springer, Heidelberg (2004)
26. Sturm, T.: New domains for applied quantifier elimination. In: Ganzha, V.G., Mayr, E.W., Vorozhtsov, E.V. (eds.) CASC 2006. LNCS, vol. 4194, pp. 295–301. Springer, Heidelberg (2006)
27. Lasaruk, A., Sturm, T.: Weak quantifier elimination for the full linear theory of the integers. A uniform generalization of Presburger arithmetic. Applicable Algebra in Engineering, Communication and Computing 18(6), 545–574 (2007)
28. Lasaruk, A., Sturm, T.: Weak integer quantifier elimination beyond the linear case. In: Ganzha, V.G., Mayr, E.W., Vorozhtsov, E.V. (eds.) CASC 2007. LNCS, vol. 4770, pp. 275–294. Springer, Heidelberg (2007)
29. Weispfenning, V.: Simulation and optimization by quantifier elimination. Journal of Symbolic Computation, Special issue on applications of quantifier elimination 24(2), 189–208 (1997)
30. Muldowney, J.S.: Compound matrices and ordinary differential equations. Rocky Mt. J. Math. 20(4), 857–872 (1990)
31. Weber, A., Sturm, T., Abdel-Rahman, E.O.: Algorithmic global criteria for excluding oscillations. Bulletin of Mathematical Biology (2010); Accepted for publication. Special issue on “Algebraic Biology”
32. Goriely, A.: Integrability and nonintegrability of dynamical systems. World Scientific, Singapore (2001)
33. Tóth, J.: Bendixson-type theorems with applications. Z. Angew. Math. Mech. 67, 31–35 (1987)
34. Hars, V., Tóth, J.: On the inverse problem of reaction kinetics. In: Farkas, M. (ed.) Colloquia Mathematica Societatis Janos Bolyai, Qualitative Theory of Differential Equations, Szeged, Hungary, pp. 363–379 (1981)
35. Tuckwell, H.C., Wan, F.Y.M.: On the behavior of solutions in viral dynamical models. BioSystems 73(3), 157–161 (2004)
36. Bonhoeffer, S., Coffin, J.M., Nowak, M.A.: Human immunodeficiency virus drug therapy and virus load. The Journal of Virology 71(4), 3275 (1997)
37. Feckan, M.: A generalization of Bendixson’s criterion. Proceedings American Mathematical Society 129(11), 3395–3400 (2001)
38. She, Z., Xia, B., Xiao, R., Zheng, Z.: A semi-algebraic approach for asymptotic stability analysis. Nonlinear Analysis: Hybrid Systems 3(4), 588–596 (2009)
39. Rabier, P.J., Rheinboldt, W.C.: Theoretical and numerical analysis of differential-algebraic equations. In: Ciarlet, P.G., Lions, J.L. (eds.) Handbook of Numerical Analysis, vol. VIII, pp. 183–540. North-Holland, Amsterdam (2002)
40. Riaza, R.: Differential-Algebraic Systems. World Scientific, Hackensack (2008)
41. Seiler, W.M.: Involution — The Formal Theory of Differential Equations and its Applications in Computer Algebra. In: Algorithms and Computation in Mathematics, vol. 24. Springer, Berlin (2009)

An Analytical Model for the Probability Characteristics of a Crack Hitting an Encapsulated Self-healing Agent in Concrete

Serguey V. Zemskov, Henk M. Jonkers, and Fred J. Vermolen

Delft University of Technology
s.zemskov@tudelft.nl

Abstract. The present study is performed in the framework of the investigation of the potential of bacteria to act as a catalyst of the self-healing process in concrete, i.e. their ability to repair occurring cracks autonomously. Spherical clay capsules containing the healing agent (calcium lactate) are embedded in the concrete structure. Water entering a freshly formed crack releases the healing agent and activates the bacteria which will seal the crack through the process of metabolically mediated calcium carbonate precipitation. In the paper, an analytic formalism is developed for the computation of the probability that a crack hits an encapsulated particle, i.e. the probability that the self-healing process starts. Most computations are performed in closed algebraic form in the computer algebra system *Mathematica* which allows to perform the last step of calculations numerically with a higher accuracy.

1 Introduction

Self-healing materials [10] are smart materials with the capability to repair themselves autonomously after being damaged. Since cracks are common in materials like concrete, polymers, etc., a self-healing mechanism is crucial for the sustainability of the material.

Since the pioneering paper by White et al. [5] on the self-healing of fracture surfaces in epoxy systems containing spherical encapsulated particles filled with a healing agent, there has been a lot of research on materials based on this principle. A key principle is the concept of spherical particles (containing self-healing agent) that break if they are intersected by a crack. Once the capsules are broken, they start releasing the healing agent by which the crack is healed. The amount of healing agent, i.e., the self-healing capacity, is determined by the number of capsules per unit of volume and capsule size. Applications are for epoxies and polymers [7,9] as well for concrete [1], where capsules are filled with nutrients for dispersed bacteria that convert nutrients into limestone, thereby sealing off the crack and protecting the steel reinforcement from corrosion.

To increase the healing capacity several studies using cylindrical particles and artificial vascular systems have been investigated as an alternative to spherical particles. A disadvantage of using a vascular system [4, 6] is the possibility of *bleeding*, that is an excessive release of self-healing agent such that later occurring cracks cannot be healed. However, the use of ellipsoidal and cylindrical particles, when arranged in the right orientation, increases the probability that an occurring crack is healed. A probabilistic study using Monte Carlo simulations to predict the release of healing agent for liquid-based self-healing systems when comparing cylindrical particles with spherical ones was carried out by Mookhoek et al [2].

Probabilistic simulations can give insight into the behavior of the probability that capsules are hit by a crack in terms of capsule density, capsule size, and crack length. These simulations will give directions for the manufacturing of self-healing materials with encapsulated pebbles or particles containing the healing agent.

Self-healing polymer systems are mostly based on *micro*capsules (2–8 μm). By such a size, the most important factor of crack healing is the total volume of healing agent flowing out into the crack plane. In self-healing concrete, however, *macro*capsules of diameter 2–4 mm are used. So, even one capsule hit by a (relatively) small crack may release a sufficient amount of the healing agent to seal off the crack.

In this paper, an analytic formalism is developed for the computation of the probability that a crack hits an encapsulated *macro*particle. This analytic model is applicable to the spherical particles and is based on elementary principles from probability and geometry, though it is original in its form and application. Most computations are performed in closed algebraic form in the computer algebra system *Mathematica* [8]. Further, the method has been validated using Monte Carlo techniques (implemented by the authors in *Mathematica* too), which, in turn, allow a broader range of particle geometries and orientations. However, an advantage of the currently developed analytical model is the lack of need of expensive Monte Carlo simulations and the direct availability of the probability as an explicit function of parameters like particle size, crack size, and healing agent content ratio. By this direct availability of the probability as a function of the parameters involved, a sensitivity analysis and optimization can be carried out easily. Hence, this method increases the amount of theoretical insight of the hitting probability and, therefore, it can be used easily for the design of materials with optimal self-healing properties.

2 Basic Notions and Model

Let us consider a cubic region containing spherical capsules (Fig. 1 (*left*)). We denote the total volume of the cube by V, and the integrated volume of incorporated capsules by V_n. It is assumed that the ratio

$$p = \frac{V_n}{V}$$

is known, as well as the average capsule volume:

$$V_c = \frac{4}{3}\pi r^3 \ .$$

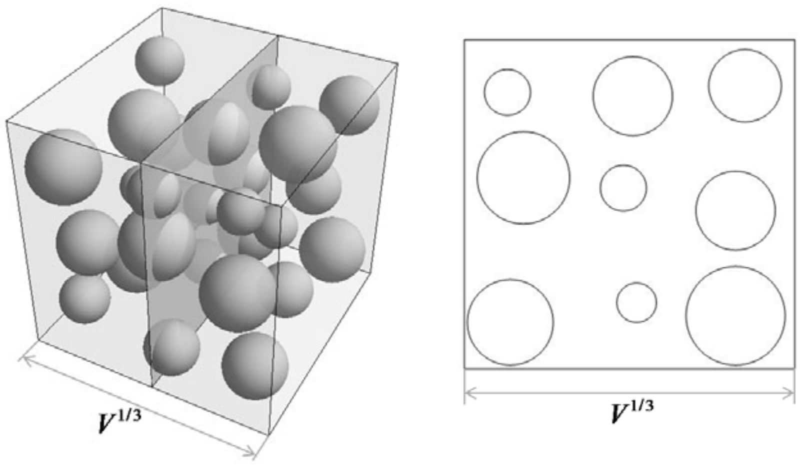


Fig. 1. Concrete cube with spatially dispersed spherical capsules and its right section

Then, the average number of capsules in the cube is

$$n_{cube}(V) = \frac{V_n}{V_c} = \frac{3}{4}\frac{pV}{\pi r^3} \ .$$

Let Ω be a square with side length $V^{1/3}$ representing a right section of the concrete cube described above (Fig. 1 (*right*)). For the calculations, it is convenient to use an integer number of capsules in the cube: $n_{cube} = n^3, n \in \mathbb{N}$. The volume

$$V = \frac{4}{3}\frac{\pi n^3 r^3}{p}, \ n \in \mathbb{N} \ , \tag{1}$$

satisfies this condition.

In order to develop the model for the probability of a crack hitting a capsule, we make the following simplifying assumptions:

- all capsules are identical full-spheres of radius r and distributed homogeneously within the cube of the volume V fulfilling (1);
- all capsule sections in Ω are equal and have the radius $R < r$;
- for the two-dimensional model, $N = n^2$ capsule sections are considered in the square

$$\Omega = \left[-\frac{V^{1/3}}{2}, \frac{V^{1/3}}{2}\right] \times \left[-R, V^{1/3} - R\right] \subset \mathbb{R}^2 \ ,$$

  sketched in Fig. 2;
- the transverse section of the crack is a line segment of a given length (crack depth) l which is perpendicular to the concrete surface (Fig. 2);

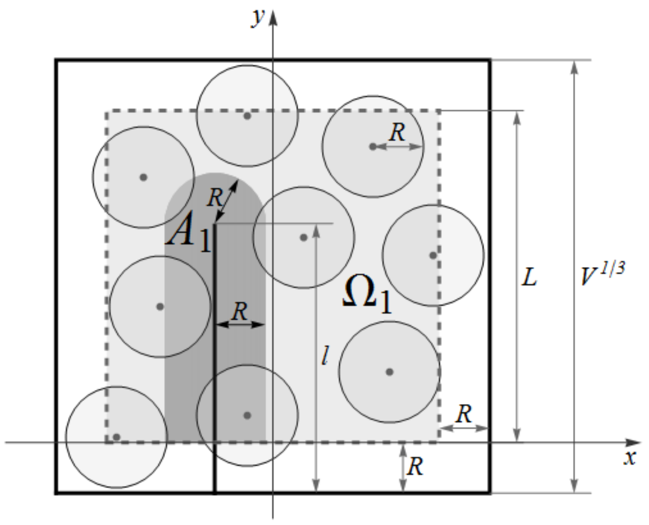


Fig. 2. Two-dimensional model of the crack in the concrete cube

Since Ω contains an integer number of capsules (circles of radius R), we may denote a geometric locus of all possible centers of capsules as Ω_1 (dashed square with side length $L = V^{1/3} - 2R$ in Fig. 2).

To complete our model, we consider a domain A such that any capsule with the center from A is hit by the crack. We denote the intersection of A and Ω_1 by $A_1 = A \bigcap \Omega_1$ (dark gray region in Fig. 2).

Under the assumption that there is only one capsule in Ω, the geometric probability [3] of hitting this capsule by the crack is

$$P_1 = \frac{S(A_1)}{S(\Omega_1)} = \frac{2R\,(l-R) + \frac{\pi R^2}{2}}{(V^{1/3} - 2R)^2} \ , \tag{2}$$

where $S(\cdot)$ denotes the area of the corresponding domain.

It should be noted that $S(A_1)$ in (2) is calculated for $l \geqslant R$. From now on, we assume that the crack depth exceeds the capsule radius. Calculations for $l < R$ are similar with due regard to another type of geometry of A_1.

Having N capsules in Ω, we denote their centers belonging to Ω_1 as

$$\mathbf{x}_i = (x_i, y_i), \ i = 1, ..., N \ .$$

The stochastic event that $\mathbf{x}_i \in A_1$ (i.e., hitting ith capsule by the crack) is denoted by

$$\Lambda_i^N = \Lambda_i^N(\mathbf{x}_1, ..., \mathbf{x}_N) \ .$$

So, for N random points $\mathbf{x}_1, ..., \mathbf{x}_N \in \Omega_1$ occurrence of $\Lambda_i^N(\mathbf{x}_1, ..., \mathbf{x}_N)$ means:

- $\mathbf{x}_i = (x_i, y_i) \in A_1$;
- $\mathbf{x}_j = (x_j, y_j) \in \Omega_1 \;$ for $\; j \neq i$;
- $d(\mathbf{x}_j, \mathbf{x}_k) = \sqrt{(x_j - x_k) + (y_j - y_k)} \in [2R, \sqrt{2}L]$ for $j, k = 1, ..., N$.

The latter constituent of the event results from the fact that the capsules are not allowed to overlap and, therefore, the distance between any two capsule centers can not be less than $2R$.

Then the event H_N of hitting *any* capsule by the crack is equal to the following union:

$$H_N = \bigcup_{i=1}^{N} \Lambda_i^N \ .$$

The geometric probability of this event can be expressed in the following way:

$$P(H_N) = P\left(\bigcup_{i=1}^{N} \Lambda_i^N\right) = \sum_{k_1=1}^{N} P(\Lambda_{k_1}^N) - \sum_{k_1<k_2} P(\Lambda_{k_1}^N \cap \Lambda_{k_2}^N) + \sum_{k_1<k_2<k_3} P(\Lambda_{k_1}^N \cap \Lambda_{k_2}^N \cap \Lambda_{k_2}^N) - \ldots + (-1)^{N-1} P(\Lambda_{k_1}^N \cap \ldots \cap \Lambda_N^N) \ . \quad (3)$$

Within the framework of the described 2D model, (3) represents a general theoretical solution of the problem of hitting any capsule by the crack.

In case of two capsules in Ω, the probability (3) takes the following form:

$$P(H_2) = P(\Lambda_1^2) + P(\Lambda_2^2) - P(\Lambda_1^2 \cap \Lambda_2^2) \ , \quad (4)$$

where $P(\Lambda_1^2) = P(\Lambda_2^2)$ (note that in general $P(\Lambda_i^k) = P(\Lambda_j^k)$ for any $i, j = 1, ..., k$ due to symmetry).

For $N = 3$, (3) looks like

$$P(H_3) = P(\Lambda_1^3) + P(\Lambda_2^3) + P(\Lambda_3^3) - P(\Lambda_1^3 \cap \Lambda_2^3) - P(\Lambda_1^3 \cap \Lambda_3^3) - P(\Lambda_2^3 \cap \Lambda_3^3) + P(\Lambda_1^3 \cap \Lambda_2^3 \cap \Lambda_3^3) \ , \quad (5)$$

where, for example,

$$\Lambda_1^3 = \Lambda_1^3(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) \ ,$$

means

- $\mathbf{x}_1 = (x_1, y_1) \in A_1$;
- $\mathbf{x}_2, \mathbf{x}_3 \in \Omega_1$;
- $d(\mathbf{x}_j, \mathbf{x}_k) = \sqrt{(x_j - x_k) + (y_j - y_k)} \in [2R, \sqrt{2}L]$ for $j, k = 1, 2, 3$,

which, in turn, means the simultaneous occurrence of three joint events for each pair of capsule centers:

$$\Lambda_1^3(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) = \Lambda_1^2(\mathbf{x}_1, \mathbf{x}_2) \cap \Lambda_1^2(\mathbf{x}_1, \mathbf{x}_3) \cap \mathrm{M}^2(\mathbf{x}_2, \mathbf{x}_3) \ , \quad (6)$$

in which $\mathrm{M}^2(\mathbf{x}_2, \mathbf{x}_3)$ is a stochastic event for two poins within Ω_1 consisting in

$$d(\mathbf{x}_2, \mathbf{x}_3) = \sqrt{(x_2 - x_3) + (y_2 - y_3)} \in [2R, \sqrt{2}L] \ .$$

In a similar way, we may express the other combination of events from (5) in terms of Λ_i^2. For example:

$$\Lambda_1^3 \cap \Lambda_2^3 = \Lambda_1^3(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) \cap \Lambda_2^3(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) = \\ (\Lambda_1^2(\mathbf{x}_1, \mathbf{x}_2) \cap \Lambda_2^2(\mathbf{x}_1, \mathbf{x}_2)) \cap \Lambda_1^2(\mathbf{x}_1, \mathbf{x}_3) \cap \Lambda_1^2(\mathbf{x}_2, \mathbf{x}_3) \ .$$

In a similar manner, the probability formulae for any number of capsules may theoretically be written using the probabilities of events Λ_i^2. So, once the components of (4) are evaluated, they can be used to find the probability for the case of more capsules in Ω.

3 Computational Aspects of the Probabilities

In this section we are going to present, as an example, the calculation sequence that leads to finding probability of the stochastic event Λ_1^2.

3.1 Theoretical Formula of the Probability

The probability of the event $\Lambda_1^2(\mathbf{x}_1, \mathbf{x}_2)$ for $\mathbf{x}_1$, $\mathbf{x}_2 \in \Omega_1$ is equal to the probability of the following intersection of events:

$$P(\Lambda_1^2(\mathbf{x}_1, \mathbf{x}_2)) = P\big((\mathbf{x}_1 \in A_1) \cap (d(\mathbf{x}_1, \mathbf{x}_2) \geqslant 2R)\big) \ .$$

First, we rewrite the probability of the intersection of events using the concept of conditional probability:

$$P\big((\mathbf{x}_1 \in A_1) \cap (d(\mathbf{x}_1, \mathbf{x}_2) \geqslant 2R)\big) = \\ P(\, d(\mathbf{x}_1, \mathbf{x}_2) \geqslant 2R \mid \mathbf{x}_1 \in A_1 \,) \cdot P(\mathbf{x}_1 \in A_1) \ . \quad (7)$$

As was already mentioned, the probability of hitting the only capsule by the crack (i.e., second factor in the right-hand side of (7)) can be evaluated easily by (2).

In order to find the conditional probability from (7) we consider new stochastic variables

$$X = x_1 - x_2 \in \left[-\frac{L}{2} - R, \frac{L}{2} + R\right] \quad \text{and} \quad Y = y_1 - y_2 \in [-L, l]$$

for $\mathbf{x}_1 = (x_1, y_1) \in A_1$, $\mathbf{x}_2 = (x_2, y_2) \in \Omega_1$.

The rectangular domain D of possible pairs (X, Y) is shown in Fig. 3 (*left*). The length of a radius vector of any point $(X, Y) = (x_1 - x_2, y_1 - y_2)$ from this domain is equal to the distance between (x_1, y_1) and (x_2, y_2) (Fig. 3 (*right*)).

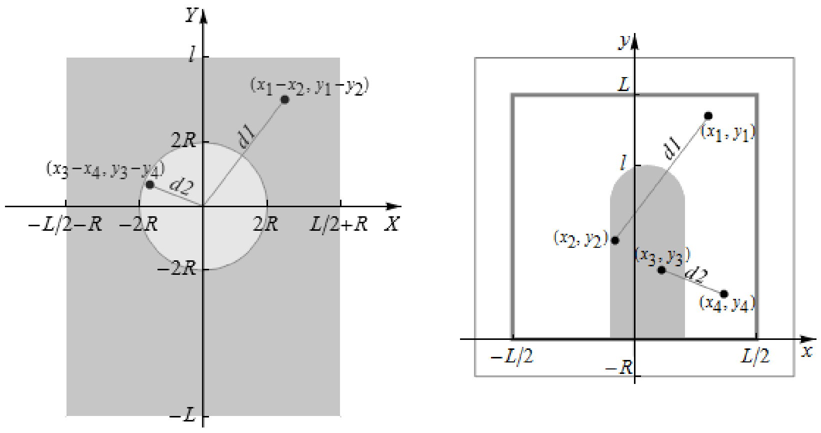


Fig. 3. Two points (x_1-x_2, y_1-y_2) and (x_3-x_4, y_3-y_4) in the domain D (*left*), and the corresponding pairs of random points $\mathbf{x}_1 \in A_1$, $\mathbf{x}_2 \in \Omega_1$ and $\mathbf{x}_3 \in A_1$, $\mathbf{x}_4 \in \Omega_1$ (*right*)

So, the fulfillment of the condition $d(\mathbf{x}_1, \mathbf{x}_2) \geqslant 2R$ is equivalent to the location of (X, Y) being outside the circle of radius $2R$ with the center in the origin. Hence,

$$P(\, d(\mathbf{x}_1, \mathbf{x}_2) \geqslant 2R \mid \mathbf{x}_1 \in A_1\,) = \iint\limits_{\substack{(X,Y)\in D,\\ X^2+Y^2>4R^2}} f_X(X) f_Y(Y)\, dX\, dY \;\;, \qquad (8)$$

where f_X and f_Y are distribution densities of X and Y, respectively.

3.2 Distribution Densities of Coordinates of Capsule Centers

It is known that for absolutely continuous independent random variables ξ_1 and ξ_2 with distribution densities f_{ξ_1} and f_{ξ_2} the distribution density of their sum $\xi_1 + \xi_2$ is equal to the convolution of their densities:

$$f_{\xi_1+\xi_2}(t) = \int_{-\infty}^{\infty} f_{\xi_1}(u) f_{\xi_2}(t-u)\, du \;\; .$$

Then in our case we obtain:

$$f_X(t) = \int_{-\infty}^{\infty} f_{x_1}(u) f_{-x_2}(t-u)\, du, \qquad f_Y(t) = \int_{-\infty}^{\infty} f_{y_1}(u) f_{-y_2}(t-u)\, du \;\; , \quad (9)$$

where f_{x_1}, f_{x_2}, f_{y_1}, and f_{y_2} are distribution densities for coordinates x_1, x_2, y_1, and y_2, respectively.

We find first the explicit formulae for distribution densities f_{x_1}, f_{x_2}, f_{y_1}, and f_{y_2}, and then use them to evaluate integrals (9) in symbolic form, which includes, besides independent variable t, capsule radius R, healing agent content ratio p, and crack depth l as parameters.

For $(x_2, y_2) \in \Omega_1$, the non-zero parts of the density functions are the same:

$$f_{x_2}(t) = \frac{1}{L},\ t \in \left[-\frac{L}{2} - R, \frac{L}{2} + R\right], \qquad f_{y_2}(t) = \frac{1}{L},\ t \in [0, L] \ .$$

To obtain the density functions for x_1 and y_1 we assume that A_1 is situated symmetrically with respect to the vertical axis $x = 0$. Then the non-zero parts of the density functions f_{x_1} and f_{y_1} are, respectively, as follows:

$$f_{x_1}(t) = \frac{d}{dt}\left(\frac{1}{S(A_1)} \int_{-R}^{t} \sqrt{R^2 - x^2} + (l - R)\, dx\right) = \frac{2\left(l - R + \sqrt{R^2 - t^2}\right)}{R\,(4\,l + R\,(\pi - 4))},\ t \in [-R, R] \ ,$$

and

$$f_{y_1}(t) = \begin{cases} f_{y_1}^{(1)}(t), & t \in [0, l - R]; \\ f_{y_1}^{(2)}(t), & t \in [l - R, l] \ , \end{cases}$$

where

$$f_{y_1}^{(1)}(t) = \frac{4}{4\,l + R\,(\pi - 4)},\quad f_{y_1}^{(2)}(t) = \frac{4\sqrt{(t - l)(l - 2R - t)}}{R\,(4\,l + R\,(\pi - 4))} \ .$$

3.3 Finding $f_X(t)$ and $f_Y(t)$

Integrals (9) are to be evaluated with regard to different combinations of algebraic expressions appearing on the corresponding intervals of integration as t runs over $\left[-\frac{L}{2} - R, \frac{L}{2} + R\right]$. For the distribution density f_X we obtain:

$$f_X(R, p, l, t) = \int_{-\infty}^{\infty} f_{x_1}(u) f_{-x_2}(t - u)\, du = \begin{cases} 0, & t \in (-\infty, -L/2 - R)\,; \\ \int_{-R}^{t+L/2} f_{x_1}(u) \cdot \frac{1}{L}\, du, & t \in [-L/2 - R, -L/2 + R)\,; \\ \int_{-R}^{R} f_{x_1}(u) \cdot \frac{1}{L}\, du, & t \in [-L/2 + R, L/2 - R)\,; \\ \int_{t-L/2}^{R} f_{x_1}(u) \cdot \frac{1}{L}\, du, & t \in [L/2 - R, L/2 + R)\,; \\ 0, & t \in [L/2 + R, \infty) \ , \end{cases}$$

and for f_Y:

$$f_Y(R,p,l,t) = \int_{-\infty}^{\infty} f_{x_1}(u) f_{-x_2}(t-u)\, du =$$

$$\begin{cases} 0, & t \in (-\infty, -L)\,; \\ \int_0^{t+L} f_{y_1}^{(1)}(u) \cdot \frac{1}{L}\, du, & t \in [-L, -L+l-R)\,; \\ \int_0^{l-R} f_{y_1}^{(1)}(u) \cdot \frac{1}{L}\, du + \int_{l-R}^{t+L} f_{y_1}^{(2)}(u) \cdot \frac{1}{L}\, du, & t \in [-L+l-R, -L+l)\,; \\ \int_0^{l-R} f_{y_1}^{(1)}(u) \cdot \frac{1}{L}\, du + \int_{l-R}^{l} f_{y_1}^{(2)}(u) \cdot \frac{1}{L}\, du, & t \in [-L+l, 0)\,; \\ \int_t^{l-R} f_{y_1}^{(1)}(u) \cdot \frac{1}{L}\, du + \int_{l-R}^{l} f_{y_1}^{(2)}(u) \cdot \frac{1}{L}\, du, & t \in [0, l-R)\,; \\ \int_t^{l} f_{y_1}^{(2)}(u) \cdot \frac{1}{L}\, du, & t \in [l-R, l)\,; \\ 0, & t \in [l, \infty)\ . \end{cases}$$

Plots of $f_X(t)$ and $f_Y(t)$ for some given values of R, p, and l are shown in Fig. 4. As an example, the resulting formulae for $f_Y(t)$ are explicitly given in Appendix A.

Similar computations are performed to evaluate the distribution densities pairs $f_{X^*}(t)$, $f_{Y^*}(t)$ and $f_{X^{**}}(t)$, $f_{Y^{**}}(t)$ when both points $(x_1, y_1), (x_2, y_2)$ belong to A_1 and Ω_1, respectively.

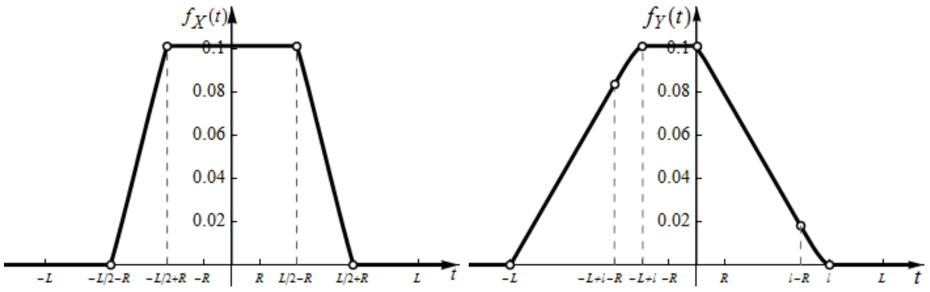


Fig. 4. Plots of distribution densities of $X = x_1 - x_2$ (left) and $Y = y_1 - y_2$ (right), $(x_1, y_1) \in A_1$, $(x_2, y_2) \in \Omega_1$, for $R = 1.5$ mm, $p = 0.18$, $l = 7$ mm

Up to now, all calculations have been performed algebraically and we obtained the expressions for $f_X(t)$ and $f_Y(t)$ in exact form that includes variables R, p, and l as well. To find the conditional probability (8), the resulting integral is to be evaluated numerically for each given set of values of R, p, and l because the exact calculations on this last step are a matter of some difficulty (and seem not to be practical). However, from behind of preliminary result in symbolic form the last integration can be done with any desirable accuracy.

The probability of the stochastic event $\Lambda_1^2(\mathbf{x}_1, \mathbf{x}_2)$ can now be evaluated according to (7).

In a similar way, we find numerically the probabilities of $\Lambda_1^2 \cap \Lambda_2^2$ from (4) and M^2 from (6).

4 Results and Conclusions

We considered the concrete cube containing $n_{cube} = 27$ capsules ($n = 3$) with the healing agent. The number of capsule sections in the square Ω in this case is equal to $N = 9$.

The hitting probability function

$$P(H_9) = f(R, p, l)$$

is evaluated numerically, and its plots for various values of p are shown in Fig. 5.

The obtained probability function allows to obtain the expected value of hitting probability for different combinations of parameter values. For example, the plot in Fig. 5 for $p = 0.18$ (black one) shows that the crack of depth equal to five average capsule radii will hit at least one capsule (or in other words, will start the process of self-healing) with the mean probability about 95% (more exactly 0.956421%).

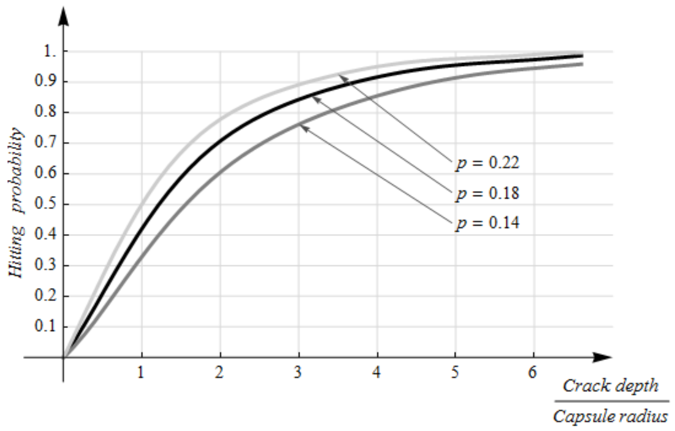


Fig. 5. Plots of the probability function $P(H_9)$ normalized with respect to the capsule section radius R for different values of healing agent content ratio p

At each step the calculations performed were validated with numerical Monte Carlo tests. One example of results of such a test is presented in Fig. 6 together with the plot of the probability function for corresponding values of parameters. It can be seen that a good agreement with the Monte Carlo simulations is obtained.

Thus, the mathematical model to calculate the probability of hitting a capsule with healing agent by the crack is developed and implemented successfully for two-dimensional case. We are going to generalize the model for three-dimensional case what is expected to allow to get rid of some simplifying assumptions and reach better physical reliability.

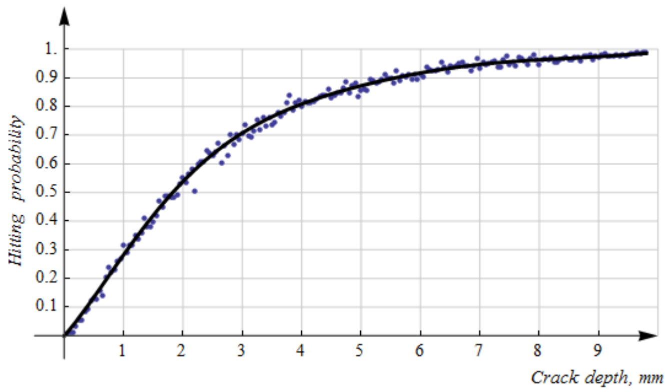


Fig. 6. Plot of the probability function $P(H_9)$ for $R = 1.5$ mm, $p = 0.18$ in comparison with results of numerical test (500 experiments for each value of crack depth l from 0.05 mm to 9.8 mm with step 0.05 mm)

References

1. Jonkers, H.M., Thijssen, A., Muyzer, G., Copuroglu, O., Schlangen, E.: Application of Bacteria as Self-Healing Agent for the Development of Sustainable Concrete. Ecological Engineering 36(2), 230–235 (2010)
2. Mookhoek, S.D., Fischer, H.R., Zwaag, S.v.d.: A Numerical Study into the Effects of Elongated Capsules on the Healing Efficiency of Liquid-Based Systems. Computational Materials Science 47(2), 506–511 (2009)
3. Rényi, A.: Wahrscheinlichkeitsrechnung, 2. Aufl., Berlin (1966)
4. Toohey, K.S., Sottos, N.R., Lewis, J.A., Moor, J.S., White, S.R.: Self-Healing Materials with Microvascular Networks. Nature Materials 6, 581–585 (2007)
5. White, S.R., Sottos, N.R., Geubelle, P.H., Moore, J.S., Kessler, M.R., Sriram, S.R., Brown, E.N., Viswanathan, S.: Autonomic Healing of Polymer Composites. Nature 409, 794–797 (2001)
6. Williams, G., Trask, R., Bond, I.: A Self-Healing Carbon Fibre Reinforced Polymer for Aerospace Applications. Composites Part A: Applied Science and Manufacturing 38(6), 1525–1532 (2007)
7. Wilson, G.O., Moore, J.S., White, S.R., Sottos, N.R., Andersson, H.M.: Autonomic Healing of Epoxy Vinyl Esters via Ring Opening Metathesis Polymerization. Advanced Functional Materials 18(1), 44–52 (2008)
8. Wolfram, S.: The Mathematica Book, 5th edn. Wolfram Media, Inc. (2003)
9. Yin, T., Rong, M.Z., Zhang, M.Q., Yang, G.C.: Self-Healing Epoxy Composites — Preparation and Effect of the Healant Consisting of Microencapsulated Epoxy and Latent Curing Agent. Composites Science and Technology 67, 201–212 (2007)
10. Zwaag, S.v.d.: Self Healing Materials: An Alternative Approach to 20 Centuries of Materials Science. Springer Series in Materials Science, vol. 100 (2007)

Appendix A

The distribution density for the stochastic variable

$$Y = y_1 - y_2,\ (x_1, y_1) \in A_1, (x_2, y_2) \in \Omega_1$$

is given by

$$f_Y(R,p,l,t) = \begin{cases} 0, & t \in (-\infty, -L)\,; \\ f_Y^{(1)}(R,p,l,t), & t \in [-L, -L+l-R)\,; \\ f_Y^{(2)}(R,p,l,t), & t \in [-L+l-R, -L+l)\,; \\ f_Y^{(3)}(R,p,l,t), & t \in [-L+l, 0)\,; \\ f_Y^{(4)}(R,p,l,t), & t \in [0, l-R)\,; \\ f_Y^{(5)}(R,p,l,t), & t \in [l-R, l)\,; \\ 0, & t \in [l, \infty)\ \ , \end{cases}$$

where

$$f_Y^{(1)}(R,p,l,t) = \frac{4\left(\left(\frac{36\pi}{p}\right)^{1/3} R - 2R + t\right)}{(4l + R(\pi-4))\left(\left(\frac{36\pi}{p}\right)^{1/3} R - 2R\right)}\ \ ,$$

and

$$f_Y^{(2)}(R,p,l,t) = \frac{2}{\left(\left(\frac{36\pi}{p}\right)^{1/3} - 2\right) R(4l + R(\pi-4))} \times$$

$$\left(- Ri \left(2R^2 \ln\left(\sqrt{l - t - R\left(\frac{36\pi}{p}\right)^{1/3}} + \sqrt{l - t + R\left(2 - \left(\frac{36\pi}{p}\right)^{1/3}\right)} \right) + \right.\right.$$

$$\sqrt{l - t - R\left(\frac{36\pi}{p}\right)^{1/3}} \sqrt{l - t + R\left(2 - \left(\frac{36\pi}{p}\right)^{1/3}\right)} \times$$

$$\left.\left. \left(t - l + R\left(\left(\frac{36\pi}{p}\right)^{1/3} - 1\right)\right) \right) + 2(l-R) + iR(\ln(1+i) + \ln R) \right)\ \ ,$$

and

$$f_Y^{(3)}(R,p,l,t) = \frac{4l - 4\left(\left(\frac{36\pi}{p}\right)^{1/3} R - 2R\right) + \left(\pi - 12 + 4\left(\frac{36\pi}{p}\right)^{1/3}\right) R}{(4l + R(\pi-4))\left(\left(\frac{36\pi}{p}\right)^{1/3} R - 2R\right)}\ \ ,$$

and

$$f_Y^{(4)}(R,p,l,t) = \frac{1 - \frac{4t}{4l + R(\pi-4)}}{\left(\frac{36\pi}{p}\right)^{1/3} R - 2R}\ \ ,$$

and

$$f_Y^{(5)}(R,p,l,t) = \frac{2}{\left(\left(\frac{36\pi}{p}\right)^{1/3} - 2\right) R^2(4l + (\pi - 4)R)} \times$$
$$\left(\left(-i\ln(2R) - 2i\ln\left(\sqrt{l-t} + \sqrt{l-2R-t}\right) + \pi\right) R^2 + \right.$$
$$\left. (R+t-l)\sqrt{(l-t)(t+2R-l)} \right) \; .$$

Extending Clause Learning of SAT Solvers with Boolean Gröbner Bases

Christoph Zengler and Wolfgang Küchlin

Symbolic Computation Group, W. Schickard-Institute for Informatics,
Universität Tübingen, Germany
{zengler,kuechlin}@informatik.uni-tuebingen.de
http://www-sr.informatik.uni-tuebingen.de

Abstract. We extend clause learning as performed by most modern SAT Solvers by integrating the computation of Boolean Gröbner bases into the conflict learning process. Instead of learning only one clause per conflict, we compute and learn additional binary clauses from a Gröbner basis of the current conflict. We used the Gröbner basis engine of the logic package Redlog contained in the computer algebra system Reduce to extend the SAT solver MiniSAT with Gröbner basis learning. Our approach shows a significant reduction of conflicts and a reduction of restarts and computation time on many hard problems from the SAT 2009 competition.

1 Introduction

In the last years SAT solvers became a vital tool in computing solutions for problems e.g. in computational biology [1,2] or formal verification [3,4]. The vast majority of SAT solvers successfully applied to real-world problems uses the *DPLL* [5] approach. DPLL is basically a complete search in the search space of all 2^n variable assignments with early cuts in the search tree when an unsatisfiable branch is detected.

It has been observed in the past that it can be profitable to enrich pure DPLL-style SAT-solving by some deduction. The extra effort of deducing new lemmas pays off when the lemmas prune the original DPLL search-tree enough to yield a net profit. However, it is clear that unrestricted deduction will choke a solver with useless clauses. This situation suggests research into suitable combinations.

Both deduction by Resolution and by the computation of Gröbner bases have been applied in the literature. The use of Resolution goes back to the original Davis-Putnam approach [5], and Buchberger's algorithm [6] has been used for theorem-proving at least since 1985 [7]. The combination approaches have been either static (in a preprocessing step) or dynamic (in the course of the search), and there are several schemes for confining deduction.

An important break-through was the development of clause learning SAT solvers (see [8, chapter 4]), which use an optimized form of resolution dynamically in conflict situations to deduce ("learn") a limited number of lemmas, so called *conflict clauses*. Resolution may deduce valuable (short) clauses as lemmas but may blow up memory, DPLL style SAT-Solving leaves memory constant but may fail to find valuable lemmas, while BDDs, or BGBs compile an axiom system at great cost in both memory and time into an efficient canonical form for repeated use in proving theorems. Today, the dominating scheme for clause learning is *1-UIP clause learning* where a single conflict clause is derived which is guaranteed to prune the backtracking tree immediately (non-chronological backtracking).

Van Gelder [9] combined conflict driven backjumping with dynamic resolution based deduction, confined to variable-elimination resolution and binary-clause resolution, and he identified (and eliminated) equivalent literals. Bacchus [10] added binary hyper-resolution and refined the implementation. Condrat and Kalla [11] take a static approach using Gröbner basis computations in a pre-processing step. They convert all derived polynomials to clauses, but confine deduction to computationally feasible subsets of the input clauses.

In contrast, we experiment with a dynamic approach, where Gröbner basis computations are confined to conflict situations and only short polynomials are kept as clauses. We start with the clauses associated with the conflict and compile them into a suitable Boolean Gröbner basis (BGB). The idea is that the interreduction process, which is inherent in the Gröbner basis algorithm and which assures that the Gröbner basis is in (minimal) canonical form with respect to a variable ordering, will also yield short (and therefore valuable) lemmas. The approach is sound because the computation of a Boolean Gröbner basis produces only Boolean polynomials which are valid consequences of the input. Our experiments with problems from the 2009 SAT competition show that the approach is valuable on many hard problems, by yielding small clauses as lemmas which reduce conflicts and therefore save time in the DPLL search.

Clearly, both the learning of additional clauses and the use of BGBs for deduction have been investigated in the past. The approach of Condrat and Kalla [11] seems to be most similar to ours. However, they only compute BGBs before executing the DPLL solver and therefore have only static information about the problem, while we use dynamically derived conflicts. We also use a variant of polynomial algebra which is better suited for representing clauses. Dershowitz et al. [12] propose an approach for satisfiability solving based on Boolean rings. This method does not incorporate learning but relies on heavy simplification of the formula during the solving process. However, they do not compare their results with state of the art SAT solvers on real problems. In [13] they describe a method for quantifier elimination in Boolen Algebras where SAT solving is a special case. To the best of our knowledge, our specific approach proposed here is novel and merits further research.

The plan of this paper as follows: In Section 2 and Section 3 we summarize the important properties of DPLL-style SAT solving and Boolean Gröbner bases up to an extent necessary to follow the rest of the paper. Section 4 presents our approach of integrating the computation of Boolean Gröbner bases in the SAT solving process. We show extensive benchmarks of our implementation in Section 5. Section 6 finally points at some future research directions.

2 SAT Solving

We use the standard notation of propositional logic with propositional variables from an infinite set $\mathcal{V}$, Boolean operators $\neg$, $\vee$, $\wedge$, and Boolean constants "**T**" and "**F**." An *assignment* is a partial function $\alpha : \mathcal{V} \to \{0, 1\}$ mapping variables to truth values. We write $x \leftarrow b$ for $\alpha(x) = b$. $\text{vars}(\varphi)$ denotes the finite set of variables occurring in a formula φ. A *conjunctive normal form* (CNF) is a conjunction of *clauses* $(\lambda_1 \vee \cdots \vee \lambda_n)$. Each *literal* λ_i is either a variable $x_i \in \mathcal{V}$ or its logical negation $\neg x_i$. It is convenient to identify a CNF with the set of all clauses contained in it. An *empty clause* is a clause where all x_i have been assigned truth values in such a way that all corresponding λ_i evaluate to 0 and therefore the whole clause is 0. It is obvious that once reaching an empty clause, no extension of the corresponding assignment can satisfy the CNF formula containing that clause. We call the occurrence of an empty clause a *conflict*. A *unit clause* is a clause where all but one x_i have been assigned, all λ_j for $j \neq i$ evaluate to 0, and therefore in order to satisfy the clause the remaining *unit variable* x_i must be assigned such that λ_i becomes 1. The process of detecting unit clauses and fixing the corresponding values for the unit variables is called *unit propagation*, which plays an important role in modern SAT solvers. Algorithm 1 summarizes the basic algorithm.

```
   Input: $C$, a set of clauses
   Output: SAT or UNSAT
 1 level := 0;
 2 $\alpha := \emptyset$;
 3 while true do
 4     unitPropagation();
 5     if $C$ contains an empty clause then
 6         level := analyseConflict();
 7         if level = 0 then
 8             return UNSAT
 9         backtrack(level);
10     else
11         if formula is satisfied then
12             return SAT
13         level := level + 1;
14         choose an $x \in \text{vars}(C) \setminus \text{dom}(\alpha)$;
15         $\alpha := \alpha \cup \{x \leftarrow 0\}$;
```

Algorithm 1. The basic SAT solving algorithm

For each variable we save its *decision level* starting with level 1 and incrementing with each assignment. If a conflict occurs at level 0, i.e. before any assignment, the formula is obviously not satisfiable. If a conflict occurs at a higher level, the method analyseConflict() is used to compute a backtrack level

Original clauses:
$r_1 = (u \vee \neg w)$
$r_2 = (\neg u \vee \neg z)$
$r_3 = (w \vee \neg y)$
$r_4 = (w \vee \neg x)$
$r_5 = (x \vee y \vee z)$

Conflict:
$(x, \neg x)$

New learned clause:
$\neg(u \wedge \neg w) = (\neg u \vee w)$

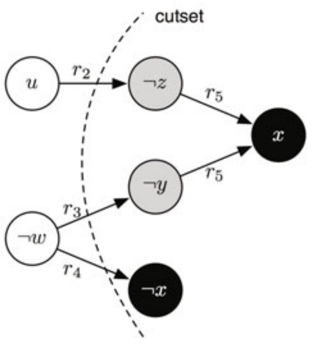


Fig. 1. An example implication graph for a given set of clauses

and the corresponding non-chronological backtrack is performed. If no conflict is detected and the formula is not yet satisfied, then the next variable is chosen and assigned a truth value. The method analyseConflict() is the place where the clause learning is actually implemented. In Figure 1 we illustrate the situation by means of an implication graph [8, chapter 4]. A node x indicates a current assignment $x \leftarrow 1$, and a node $\neg x$ indicates a current assignment of $x \leftarrow 0$. Nodes without incoming edges (white nodes) are variables which were chosen by the algorithm in line 14; they are referred to as *decision nodes*. Nodes with incoming edges (grey nodes) result from unit propagations (line 4). The clause annotated to each edge is the clause which caused the unit propagation. For example when assigning $u \leftarrow 1$ the clause $r_2 = (\neg u \vee \neg z)$ is the reason that $z \leftarrow 0$ during unit propagation. The two black nodes indicate *conflicting nodes* since from the current assignment of the variables it can be concluded that x has to be assigned 0 as well as 1. To *learn* a new clause which avoids this assignment for the future we have to find a *cut* dividing the conflicting nodes from the decision nodes. The dashed line in Figure 1 shows one example for such a cut. According to this cut we have to avoid the simultaneous assignment $u \leftarrow 1$ and $w \leftarrow 0$ in order to avoid the conflicting nodes x and $\neg x$. We thus add to our set of clauses the new clause $(\neg u \vee w)$, which is equivalent to the more natural condition $\neg(u \wedge \neg w)$. In general, new learned clauses are obtained via resolution over all clauses on the paths from conflict nodes to decision nodes.

Following the 1-UIP strategy, the backtrack level returned by analyseConflict() is determined in such a way that the new learned clause turns out to be a unit clause after backtracking to this level and therefore is used for unit propagation immediately. For more details on clause learning and different strategies to compute the cut between conflict nodes and decision nodes see [14]. After a certain number of conflicts, all assignments are taken back (learnt clauses are preserved) and the SAT solver is restarted.

3 Boolean Gröbner Bases

A *Boolean Ring* $\mathcal{B}$ is a ring in which every element $a \in \mathcal{B}$ is *idempotent*, i.e. $a^2 = a$. Both $\mathcal{B}^{\oplus} = (\oplus, \wedge, \mathbf{F}, \mathbf{T})$ and $\mathcal{B}^{\leftrightarrow} = (\leftrightarrow, \vee, \mathbf{T}, \mathbf{F})$ with universes $\{0, 1\}$ are Boolean rings.[1] Let $\boldsymbol{x} = (x_1, \dots, x_n)$. Recall that polynomials are elements of a polynomial ring $\mathcal{R}[\boldsymbol{x}]$ over a coefficient ring $\mathcal{R}$. Important rings of Boolean polynomials are $\mathcal{B}^{\oplus}[\boldsymbol{x}]$, the ring of Stone polynomials [15] in *xor normal form* XNF, and $\mathcal{B}^{\leftrightarrow}[\boldsymbol{x}]$, the ring of polynomials in *equivalence normal form* ENF. For our presentation here we consider the ring $\mathcal{B}^{\leftrightarrow}[x_1, \dots, x_n] / \operatorname{Id}(x_1^2 + x_1, \dots, x_n^2 + x_n)$. In contrast to the polynomial rings mentioned above, this residue class ring is a Boolean ring itself. Since the basis of the ideal $\operatorname{Id}(x_1^2 + x_1, \dots, x_n^2 + x_n)$ is a Gröbner basis, a reduction with this basis yields unique normal forms. It is easy to see that the corresponding normal forms have a degreebound of one on every single variable. We choose these normal forms as representatives for our residue classes. For a given ordering of variables, both XNF and ENF are canonical normal forms: $A \equiv B$ if, and only if, $A^{\mathrm{XNF}} = B^{\mathrm{XNF}}$, and likewise $A^{\mathrm{ENF}} = B^{\mathrm{ENF}}$.

Let A_f be an expression denoting f. For the algebraic approach, we may convert A_f into A_f^{XNF} or into A_f^{ENF}. In our case we start with a set C of clauses, representing the conjunction $A_f = \bigwedge_{c_i \in C} c_i$. In this case it is not useful to convert the entire A_f into a single polynomial: Using Stone polynomials, all occurrences of $x_1 \vee x_2$ in the clauses must be eliminated by the transformation $x_1 + x_2 + x_1 \cdot x_2$, while the outer conjunctions lead to an analogous blow-up when using ENF-polynomials. Therefore we only convert each clause

$$c_i = (x_1 \vee \dots \vee x_n \vee \neg y_1 \vee \dots \vee \neg y_m) = (x_1 \vee \dots \vee x_n \vee (y_1 \leftrightarrow \mathbf{F}) \vee \dots \vee (y_m \leftrightarrow \mathbf{F}))$$

into a separate ENF-polynomial

$$p^{\mathrm{ENF}}(c_i) = x_1 \cdot \ \dots \ \cdot x_n \cdot (y_1 + 1) \cdot \ \dots \ \cdot (y_m + 1),$$

and then compute a Boolean Gröbner basis A_f^{EGB} for the set $\{p^{\mathrm{ENF}}(c_i) \mid c_i \in C\}$. Note that using Stone polynomials for the clauses (as in [7,11]) would lead to a blow-up in the polynomials prior to the Gröbner basis computation.

A_f^{EGB} is a canonical representative for the variety (the set of common roots) of the polynomials $p^{\mathrm{ENF}}(c_i)$. Hence A_f^{EGB} represents the associated Boolean function f in a minimal way, according to some ordering. In this application, we are not interested in the canonical form produced, but rather in the deduction of new polynomials in the course of the Gröbner basis computation. These are always reduced and therefore show some promise of representing small and valuable lemmas. Every new polynomial g introduced by the Gröbner basis computation must have its roots in the original variety. In logical terms, this means that $f = \bigwedge c_i \equiv \bigwedge c_i \wedge g$. Hence $\bigwedge c_i \Rightarrow g$, i.e. Gröbner basis computation produces valid lemmas [7].

1 $\oplus$ denotes the XOR operator.

Recently, significant progress has been made in efficient custom implementations of BGBs [16]. For this initial study, however, we used a standard implementation of Buchberger's algorithm for general polynomial rings. This is possible by adding, for each variable x_i, an idempotency polynomial $x_i^2 + x_i$ to the set of clause polynomials. The mathematics for our specific approach is discussed in [17]. All general mathematical background can be found in [18,19].

4 Combining SAT Solving and Gröbner Bases

In Section 2 we demonstrated how a new clause c is deduced from a current conflict by means of an implication graph. As mentioned before new learned clauses are obtained via resolution over all clauses on the paths from conflict nodes to decision nodes. In order to perform these resolutions, solvers store a reason clause $r(x)$ for each variable x implied by unit propagation. We perform learning as usual (following the 1-UIP strategy) and add for each (unsubstituted) reason clause $r(x)_i$ involved in the conflict at hand its corresponding polynomial $p^{\text{ENF}}(r(x)_i)$ to a set R. After learning the 1-UIP clause, we compute a Gröbner basis $G = \text{gb}(R \cup \mathcal{F})$, where $\mathcal{F}$ is the set of idempotency polynomials $x_i^2 + x_i$ for all variables x_i in $\bigcup_{p \in R} \text{vars}(p)$. We collect all polynomials $p_i \in G \setminus R$ with $|\,\text{vars}(p_i)| = 2$ and add their corresponding clause representation clause(p_i) to a set L. At the next restart of the solver, we add all clauses of L to the original clause set C. We have seen in the last section that this step does not change the semantics of the original clause set C.

Experiments have shown that current Gröbner basis packages cannot cope with large polynomial systems. Therefore we have to restrict the set R in number and length of polynomials. Instead of computing gb(R), we compute only gb(R') with $R' = \{p_i \in R \mid 2 \leq |\,\text{vars}(p_i)| \leq 8\} \subseteq R$. Additionally we only perform the computation gb(R') when $4 \leq |R'| \leq 6$. This means we compute only Gröbner bases of subsets $R' \subseteq R$, where there are between 4 and 6 underlying clauses with $2-8$ literals.

We found these numbers by extensive testing. Choosing more or larger reason clauses often leads to long BGB computations and therefore slows down the overall solving process. Taking fewer or shorter reason clauses does often not produce new binary clauses and therefore does not speed up the solving process. However, we assume that these numbers strongly dependend on the chosen Gröbner basis package.

Furthermore we do not compute the Gröbner basis for every R' where the condition holds. Our tests showed that for large problems these calls to the Gröbner basis algorithm are too expensive. Therefore we only compute each 2^n-th Gröbner basis where n is the number of restarts. These two heuristics turned out to be best performing among all alternatives and therefore were chosen for the following benchmarks.

Remark 1. The additional clauses computed by the Gröbner basis algorithm could also be deduced by resolution. But there are two important points why we use Boolean Gröbner bases at this point. First, we make use of the interreduction process, which is inherent in the Gröbner basis algorithm and which assures that the Gröbner basis is in (minimal) canonical form with respect to a variable ordering. Second, in our line of research (see also Section 6) we plan to interweave SAT solving and Boolean Gröbner bases more extensively. If one computes a Gröbner basis of clauses of more than one conflict, the result cannot be yielded by resolution anymore. At this point Gröbner bases can play off their strength.

Original clauses:
$r_1 = (y \vee f)$
$r_2 = (\neg f \vee g)$
$r_3 = (\neg f \vee \neg g \vee \neg h)$
$r_4 = (\neg u \vee \neg m \vee h)$
$r_5 = (\neg u \vee \neg f \vee m)$

Conflict:
$(m, \neg m)$

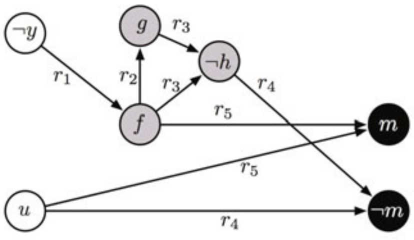


Fig. 2. An example for the Gröbner basis integration

Example 2 (Learning binary clauses with Gröbner bases). We consider a conflict with an implication graph like in Figure 2 and reason clauses $r_1, \dots, r_5$. We translate each of these clauses r_i into its corresponding Boolean polynomial $p^{\mathrm{ENF}}(r_i)$ and obtain the set $R = \{y \cdot f, (f+1) \cdot g, (f+1) \cdot (g+1) \cdot (h+1), (u+1) \cdot (m+1) \cdot h, (u+1) \cdot (f+1) \cdot m\}$. Obviously in this case $R' = R$. $\mathcal{F} = \{y^2 + y, u^2 + u, f^2 + f, m^2 + m, h^2 + h, g^2 + g\}$. We compute the Gröbner basis G of $R \cup \mathcal{F}$ and get $G \setminus R = \{y \cdot (u+1), y \cdot g, y \cdot (h+1), (u+1) \cdot (f+1), (f+1) \cdot (h+1)\}$. For all five polynomials p_i it holds that $|\operatorname{vars}(p_i)| = 2$. We add their corresponding clauses $(y \vee \neg u), (y \vee g), (y \vee \neg h), (\neg u \vee \neg f)$, and $(\neg f \vee \neg h)$ to L and add them to the clause set C at the beginning of the next restart. □

5 Implementation and Benchmarks

The publically available 2007 version of MiniSat[2] [20] served as a basis for our implementation *MiniSat+GB*. For the Gröbner bases computations we used the package `cgb` with lexicographical term ordering for all computations. `cgb` is implemented in the open-source Computer Algebra system Reduce[3] and it is used within the logic package Redlog [21] for various quantifier elimination procedures and for a simplifier based on Gröbner bases. So far we used `cgb` as a black box and therefore other (more efficient Boolean) Gröbner basis implementations could easily be substituted. However, with the current heuristics, we spend only about 1/1000 of the overall time in the Gröbner basis computations.

For our benchmarks we used PSL-based Reduce and version 070721 of MiniSat on one core of an Apple Mac Pro (with two 2.8 GHz Quad-Core Intel Xeon

[2] `http://minisat.se/MiniSat.html`
[3] `http://reduce-algebra.sourceforge.net`

Table 1. Comparison between MiniSat and MiniSat+GB

		MiniSat		MiniSat+GB		% saved	
benchmark	# inst	time	conflicts	time	conflicts	time	conflicts
aprove09	17	62.5	129782	**51.8**	**96765**	17.1	25.4
bioinf	17	5245.4	5373507	**4899.4**	**5165204**	6.6	3.9
bitverif	12	7382.9	4857477	**5317.7**	**4126969**	28.0	15.0
c32sat	4	11921.9	2435435	**8539.3**	**1891845**	28.4	22.3
crypto	10	3167.9	408098	**2917.6**	**397982**	7.9	2.48
palacios/uts	6	1949.9	1179959	**1395.9**	**926299**	28.4	21.5
parity-games	18	2418.4	2220381	**1469.7**	**1707849**	39.2	23.1

Processors). MiniSat+GB calls the C library `libreduce`, which in turn communicates via sockets with a separate Reduce process.

Table 1 presents the results. As benchmark set we chose all instances of the SAT 2009 competition[4], which could be solved by MiniSat in less than 10000 s. These are 84 instances over all. We give the name of the benchmark family, the number of instances in this set, the CPU time in seconds and the number of conflicts for both MiniSat and MiniSat+GB. The last two columns state the percentage of saved time and saved conflicts.

One can clearly see that MiniSat+GB outperforms MiniSat w. r. t. the accumulated time for each benchmark family. This is mainly because MiniSat+GB performs especially well on the large and hard instances of every family. Taking all instances of all benchmark families, we produce 13.8% fewer conflicts and therefore save 23.5% of solving time.

Figure 1 gives a diagram of all instances we benchmarked. We can observe three interesting areas: ① For instances taking less than 1 s of CPU time the overhead of computing the Gröbner bases bears no relation to the saved conflicts. Therefore our implementation always performs worse than MiniSat. ② These are instances from the `bioinf` benchmark, where we could not learn any new binary clauses at all. ③ For all instances taking more than 1000 s of CPU time MiniSat+GB clearly outperforms MiniSat (note the logarithmic scale). One of the clearest examples is the `minxorminand064` benchmark where we achieved a speedup factor of 3.3 (621.2 s vs. 2195.3 s).

6 Future Work

There are still many open questions. Are there better heuristics when to compute a Gröbner basis, based not only on the number and length of clauses? Can we profit from results [11] about the impact of term orderings? Can we learn slightly longer clauses that are still useful? How much improvement is possible by going to a dedicated implementation of Boolean Gröbner bases? We want to compute the Gröbner bases not only of the clauses of one conflict but also of the collected clauses of different conflicts, or of the most active clauses. These Gröbner bases

[4] `http://www.satcompetition.org/2009/`

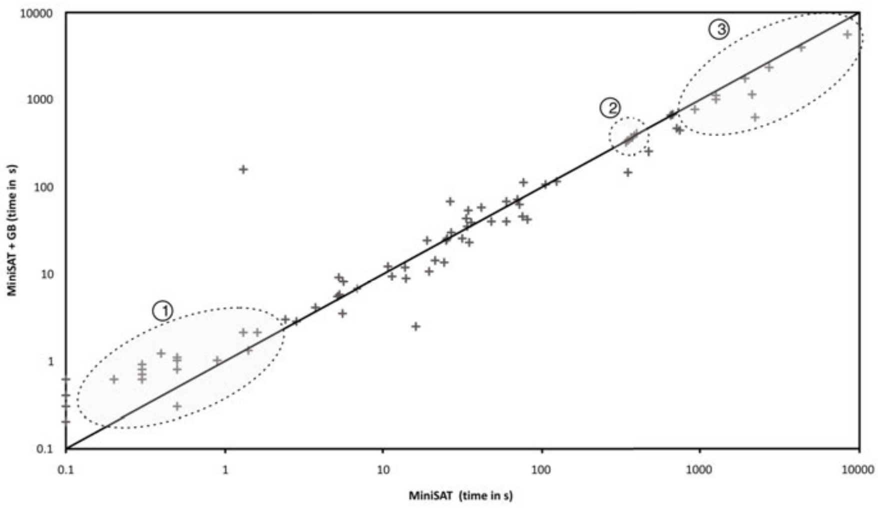


Fig. 3. Graphical comparison between MiniSat and MiniSat+GB

could then be used to perform simplifications of the problem as described in [12]. Our approach also provides an opportunity for parallel SAT solving where one machine performs the basic SAT solving and other machines compute Gröbner bases which they communicate to the SAT solving process. On the application side we want to examine the impact of our improvements on incremental SAT solving and parametric SAT solving [22].

Acknowledgment

Our undergraduate student Wolfgang Braun provided valuable help with the implementation.

References

1. Lynce, I., Marques da Silva, J.P.: SAT in bioinformatics: Making the case with haplotype inference. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 136–141. Springer, Heidelberg (2006)
2. Bonet, M.L., John, K.S.: Efficiently calculating evolutionary tree measures using SAT. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 4–17. Springer, Heidelberg (2009)
3. Clarke, E.M., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. Formal Methods in System Design 19(1), 7–34 (2001)
4. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. In: Zelkowitz, M. (ed.) Highly Dependable Software. Advances in Computers, vol. 58. Academic Press, San Diego (2003)

5. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Communications of the ACM 5(7), 394–397 (1962)
6. Buchberger, B.: Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal. PhD thesis, Universität Innsbruck (1965)
7. Kapur, D., Narendran, P.: An equational approach to theorem proving in first-order predicate calculus. ACM SIGSOFT Notes 10(4), 63–66 (1985)
8. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability. Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press, Amsterdam (2009)
9. Van Gelder, A.: Combining preorder and postorder resolution in a satisfiability solver. Electronic Notes in Discrete Mathematics 9, 115–128 (2001)
10. Bacchus, F.: Enhancing davis putnam with extended binary clause reasoning. In: 18th National Conference on Artificial Intelligence, pp. 613–619. AAAI Press, Menlo Park (2002)
11. Condrat, C., Kalla, P.: A gröbner basis approach to CNF-formulae preprocessing. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 618–631. Springer, Heidelberg (2007)
12. Dershowitz, N., Hsiang, J., Huang, G.S., Kaiss, D.: Boolean rings for intersection-based satisfiability. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS (LNAI), vol. 4246, pp. 482–496. Springer, Heidelberg (2006)
13. Seidl, A.M., Sturm, T.: Boolean quantification in a first-order context. In: Proceedings of the CASC 2003. Institut für Informatik, Technische Universität München, Garching, pp. 329–345 (2003)
14. Beame, P., Kautz, H., Sabharwal, A.: Towards understanding and harnessing the potential of clause learning. JAIR 22(1), 319–351 (2004)
15. Stone, M.H.: The theory of representations for boolean algebras. Transactions of the American Mathematical Society 40, 37–111 (1936)
16. Brickenstein, M., Dreyer, A., Greuel, G.M., Wedler, M.: New developments in the theory of gröbner bases and applications to formal verification. Journal of Pure and Applied Algebra 213, 1612–1635 (2009)
17. Küchlin, W.: Canonical hardware representation using Gröbner bases. In: Proceedings of the A3L 2005, Passau, Germany, pp. 147–154 (2005)
18. Becker, T., Weispfenning, V.: Gröbner Bases: A Computational Approach to Commutative Algebra. Springer, New York (1993)
19. Cox, D., Little, J., O'Shea, D.: Ideals, Varieties, and Algorithms, 3rd edn. Springer, New York (2007)
20. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
21. Dolzmann, A., Sturm, T.: Redlog: Computer algebra meets computer logic. ACM SIGSAM Bulletin 31(2), 2–9 (1997)
22. Sturm, T., Zengler, C.: Parametric quantified SAT solving. In: Proceedings of the ISSAC 2010. ACM, New York (2010)

Author Index