The IMA Volumes in Mathematics and its Applications

Volume 148

Series Editors
Douglas N. Arnold Arnd Scheel

Institute for Mathematics and its Applications (IMA)

The Institute for Mathematics and its Applications was established by a grant from the National Science Foundation to the University of Minnesota in 1982. The primary mission of the IMA is to foster research of a truly interdisciplinary nature, establishing links between mathematics of the highest caliber and important scientific and technological problems from other disciplines and industries. To this end, the IMA organizes a wide variety of programs, ranging from short intense workshops in areas of exceptional interest and opportunity to extensive thematic programs lasting a year. IMA Volumes are used to communicate results of these programs that we believe are of particular value to the broader scientific community.

The full list of IMA books can be found at the Web site of the Institute for Mathematics and its Applications:

http://www.ima.umn.edu/springer/volumes.html

Presentation materials from the IMA talks are available at

http://www.ima.umn.edu/talks/

Douglas N. Arnold, Director of the IMA

* * * * * * * * * *

IMA ANNUAL PROGRAMS

1982–1983	Statistical and Continuum Approaches to Phase Transition
1983–1984	Mathematical Models for the Economics of Decentralized Resource Allocation
1984–1985	Continuum Physics and Partial Differential Equations
1985–1986	Stochastic Differential Equations and Their Applications
1986–1987	Scientific Computation
1987–1988	Applied Combinatorics
1988–1989	Nonlinear Waves
1989–1990	Dynamical Systems and Their Applications
1990–1991	Phase Transitions and Free Boundaries
1991–1992	Applied Linear Algebra
1992–1993	Control Theory and its Applications
1993–1994	Emerging Applications of Probability
1994–1995	Waves and Scattering
1995–1996	Mathematical Methods in Material Science
1996–1997	Mathematics of High Performance Computing

(Continued at the back)

Michael Stillman Nobuki Takayama
Jan Verschelde
Editors

Software for Algebraic Geometry

Springer

Editors
Michael Stillman
Department of Mathematics
Cornell University
Ithaca, NY 14853-4201
USA

Nobuki Takayama
Department of Mathematics
Kobe University
Rokko, Kobe 657-8501
Japan

Jan Verschelde
Department of Mathematics
University of Illinois
Chicago, IL 60607-7045
USA

Series Editors
Douglas N. Arnold
Arnd Scheel
Institute for Mathematics and its
Applications
University of Minnesota
Minneapolis, MN 55455
USA

ISBN: 978-0-387-78132-7 e-ISBN: 978-0-387-78133-4
DOI: 10.1007/978-0-387-78133-4

Library of Congress Control Number: 2008923357

Mathematics Subject Classification (2000): 11R09, 11Y99, 12D05, 13P10, 14P05, 14Q05, 14Q10
52A39, 52B20, 52B55, 65D18, 65F15, 65F20, 65F22, 65H10, 65H20, 65-04, 91-08

Camera-ready copy provided by the IMA.

9 8 7 6 5 4 3 2 1

springer.com

FOREWORD

This IMA Volume in Mathematics and its Applications

SOFTWARE FOR ALGEBRAIC GEOMETRY

contains papers presented at a highly successful one-week workshop on the same title. The event was an integral part of the 2006-2007 IMA Thematic Year on "Applications of Algebraic Geometry." We are grateful to all the participants for making this workshop a very productive and stimulating event. Special thanks to Michael E. Stillman (Department of Mathematics, Cornell University), Nobuki Takayama (Department of Mathematics, Kobe University), and Jan Verschelde (Department of Mathematics, Statistics and Computer Science, University of Illinois at Chicago) for their superb role as workshop organizers and editors of these proceedings.

We take this opportunity to thank the National Science Foundation for its support of the IMA.

Series Editors

Douglas N. Arnold, Director of the IMA

Arnd Scheel, Deputy Director of the IMA

PREFACE

The workshop on "Software for Algebraic Geometry" was held in the week from 23 to 27 October 2006, as the second workshop in the thematic year on Applications of Algebraic Geometry at the IMA.

Algorithms in algebraic geometry go hand in hand with software packages that implement them. Together they have established the modern field of computational algebraic geometry which has come to play a major role in both theoretical advances and applications. Over the past fifteen years, several excellent general purpose packages for computations in algebraic geometry have been developed, such as CoCoA, Singular and Macaulay 2. While these packages evolve continuously, incorporating new mathematical advances, they both motivate and demand the creation of new mathematics and smarter algorithms.

Surrounding the general packages, a host of specialized packages for dedicated and focused computations have created a platform for the interaction of algebraic geometry with numerous other areas of mathematics including optimization, combinatorics, polyhedral geometry, numerical analysis and computer science. The workshop brought together a wide array of theoreticians and practitioners interested in the development of algorithms and software in algebraic geometry at this workshop. Such interactions are essential for dramatic increases in the power and applicability of algorithms in the field.

There were 89 registered participants at the workshop. At four talks a day, 20 regular 50 minutes talks were scheduled. On Monday evening, 10 posters were presented. On Wednesday and Thursday evening we had respectively 5 and 6 software demonstrations. The list of featured software packages includes Macaulay 2, SAGE, HomLab, Bertini, APAtools, PHClab, PHCmaple, PHCpack, KNOPPIX/Math, D-modules for Macaulay 2, Singular, Risa/Asir, CRACK, diffalg, RIFsimp, Gambit, FGb/RS, CoCoALib, 4ti2, PHoMpara, SYNAPS, DEMiCs, Magma, Kronecker, SOSTOOLS, Gfan, Maple 11.

The IMA systems group had installed many of these programs on the computers at the IMA. At the poster session, the participants were given the opportunity to install the featured software systems on their laptop. A demonstration cluster computer of Rocketcalc was running during the poster session and accessible to all participants during the workshop.

The evening before the workshop dinner on Tuesday started with a problem session. Prior to this session we made a list of problem descriptions available on the web site. The workshop ended on Friday evening with some additional problems, discussion on the posted problems, and a presentation of Jiawang Nie about the application to semidefinite programming to solve systems of algebraic equations which arise from differential equations. We

are also happy that several new research projects were stimulated by this problem session. Some results are going to appear elsewhere.

Instead of the "second chances" (usual for IMA workshops), the participants were given the opportunity to test the software systems on Wednesday and Thursday evening. The evening session started with a one hour plenary session, where each software system on demo in the evening was briefly explained. Following this plenary session, the participants moved to the 4th floor of Lind Hall, to experience the software systems on the computers in the open poster area, or in parallel, in the classroom 409.

The IMA systems group worked hard in the weeks leading up to the workshop to install the software systems. Their effort benefited not only the workshop participants, but all all subsequent participants to the thematic year, as they found their workstations equipped with the latest software tools in algebraic geometry.

The papers in this volume describe the software packages Bertini, PHClab, Gfan, DEMiCs, SYNAPS, TrIm, Gambit, ApaTools, and the application of Risa/Asir to a conjecture on multiple zeta values. We thank the participants to the workshop, the authors and the anonymous referees. We are grateful to the editorial staff of the IMA, Patricia V. Brick and Dzung N. Nguyen, for their dedication and care.

Michael E. Stillman
Department of Mathematics
Cornell University
http://www.math.cornell.edu/People/Faculty/stillman.html

Nobuki Takayama
Department of Mathematics
Kobe University
http://www.math.sci.kobe-u.ac.jp/ taka/

Jan Verschelde
Department of Mathematics, Statistics and Computer Science
University of Illinois at Chicago
http://www2.math.uic.edu/ jan/

CONTENTS

SOFTWARE FOR NUMERICAL ALGEBRAIC GEOMETRY: A PARADIGM AND PROGRESS TOWARDS ITS IMPLEMENTATION

DANIEL J. BATES*, JONATHAN D. HAUENSTEIN†, ANDREW J. SOMMESE‡, AND CHARLES W. WAMPLER II§

Abstract. Though numerical methods to find all the isolated solutions of nonlinear systems of multivariate polynomials go back 30 years, it is only over the last decade that numerical methods have been devised for the computation and manipulation of algebraic sets coming from polynomial systems over the complex numbers. Collectively, these algorithms and the underlying theory have come to be known as numerical algebraic geometry. Several software packages are capable of carrying out some of the operations of numerical algebraic geometry, although no one package provides all such capabilities. This paper contains an enumeration of the operations that an ideal software package in this field would allow. The current and upcoming capabilities of Bertini, the most recently released package in this field, are also described.

Key words. Homotopy continuation, numerical algebraic geometry, polynomial systems, software, Bertini.

AMS(MOS) subject classifications. 65H10, 65H20, 65-04, 14Q99.

1. Introduction.

Numerical algebraic geometry refers to the application of numerical methods to compute the solution sets of polynomial systems, generally over $\mathbb{C}$. In particular, basic numerical algebraic geometry embodies probability one algorithms for computing all isolated solutions of a polynomial system as well as the numerical irreducible decomposition of an algebraic set, i.e., one or more points on each irreducible component in each dimension. More recently, numerical algebraic geometry has grown to include more advanced techniques which make use of the basic methods in order to compute data of interest in both real-world applications and pure algebraic geometry.

*Institute for Mathematics and its Applications, University of Minnesota, Minneapolis, MN 55122 (dbates1@nd.edu, www.nd.edu/~dbates1). This author was supported by the Duncan Chair of the University of Notre Dame, the University of Notre Dame, NSF grant DMS-0410047, the Arthur J. Schmitt Foundation, and the Institute for Mathematics and its Applications in Minneapolis (IMA).

†Department of Mathematics, University of Notre Dame, Notre Dame, IN 46556 (jhauenst@nd.edu, www.nd.edu/~jhauenst). This author was supported by the Duncan Chair of the University of Notre Dame, NSF grant DMS-0410047, and the Institute for Mathematics and its Applications in Minneapolis (IMA).

‡Department of Mathematics, University of Notre Dame, Notre Dame, IN 46556 (sommese@nd.edu, www.nd.edu/~sommese). This author was supported by the Duncan Chair of the University of Notre Dame, the University of Notre Dame, NSF grant DMS-0410047, and the Institute for Mathematics and its Applications in Minneapolis (IMA).

§General Motors Research and Development, Mail Code 480-106-359, 30500 Mound Road, Warren, MI 48090 (Charles.W.Wampler@gm.com, www.nd.edu/~cwample1). This author was supported by NSF grant DMS-0410047 and the Institute for Mathematics and its Applications in Minneapolis (IMA).

One of the key tools used in the algorithms of numerical algebraic geometry is homotopy continuation [1, 16], a method for finding all zero-dimensional solutions of a polynomial system. Given a polynomial system $f : \mathbb{C}^N \to \mathbb{C}^n$ to be solved by homotopy continuation, one first forms a polynomial system g that is related to f in a prescribed way but has known, or easily computable, solutions. The systems g and f are combined to form a homotopy, such as the linear homotopy $H(x,t) = f \cdot (1-t) + \gamma \cdot t \cdot g$ where $\gamma \in \mathbb{C}$ is randomly chosen. For a properly formed homotopy, there are continuous solution paths leading from the solutions of g to those of f which may be followed using predictor-corrector methods. Singular solutions cause numerical difficulties, so singular endgames [17, 18, 19] are typically employed. Zero-dimensional solving is discussed further in Section 2.2.

Numerical algebraic geometry treats both zero-dimensional (isolated) solutions and positive dimensional solution sets. The building blocks of the solution set of a set of equations are the irreducible components, i.e., the algebraic subsets of the solution set that consist of connected sets of points with neighborhoods biholomorphic to a neighborhood of a Euclidean space. The solution set breaks up into a union of a finite number of irreducible components, none of which is contained in the union of the remaining components. In numerical algebraic geometry, we associate to each component a witness set, which is the basic data structure used to numerically describe and manipulate positive dimensional solution sets. Given a multiplicity one irreducible component Z of a system of polynomials $f(x) = 0$, a witness set consists of a triple (f, L, W), where L is a random linear space of dimension complementary to that of Z and where W is a set of points such that $W := Z \cap L$. There is a slightly more involved definition for the case of components of multiplicity greater than one as described in [24].

Many of the algorithms of numerical algebraic geometry make abundant use of homotopy continuation. For example, the cascade algorithm [20], one of the basic methods involved in computing the numerical irreducible decomposition of an algebraic set, uses repeated applications of homotopy continuation at different dimensions in order to produce points on each component in each dimension. Monodromy and trace tests then lead to the complete numerical irreducible decomposition of the solution set of f. In the end, the user obtains from the methods of numerical algebraic geometry a wealth of information regarding the characteristics of the solution set of a given polynomial system, some of which may be difficult to procure by purely symbolic means. For good references on zero-dimensional solving see [13, 16] and for numerical algebraic geometry see [24].

There are several software packages available to the public which carry out some of the operations of numerical algebraic geometry. However, no one package contains all such capabilities. These packages include HOM4PS [6], PHoM [9], POLSYS [27], PHCpack [28], and HomLab [30]. The most recently released package, Bertini [2], is under ongoing develop-

ment by the authors. Although all software packages were developed at different times for different reasons, they share the goal of solving polynomial systems by numerical means.

The purpose of the present paper is two-fold. One purpose is to present a paradigm for software in the field of numerical algebraic geometry. The following section contains an elaboration on the aforementioned algorithms and an enumeration of the various operations required for carrying out those algorithms, broken into four levels. Implementation-specific details, such as data structures, are omitted. The other purpose is to provide a brief introduction to Bertini and indicate its partial fulfillment of the paradigm of Section 2. That is the content of Section 3, which is also broken up into four levels to mirror Section 2. The final section includes planned extensions of the Bertini software package.

2. A paradigm for numerical algebraic geometry software. All good software packages share several characteristics. In particular, good software should be reliable (i.e., it provides correct output with clear signals upon failure), as fast as possible with estimates of time remaining for large jobs, modular for easy modification, and user-friendly. In addition, good *numerical* software must be accurate and provide error estimates for all solutions. Reliability and accuracy generally have an adverse impact on speed, so while efficiency is important, it should not be emphasized over finding correct answers.

Numerical accuracy may be approached in two ways. One approach is to use a fixed level of precision and find as accurate a solution as possible, possibly using higher levels of precision for subsequent runs to attain more accuracy, if necessary. The other approach is to select an accuracy before the run and adjust the precision during the run to attain that accuracy. Either way, it has recently become generally accepted that it is important to have available multiple levels of precision when implementing numerical routines.

The purpose of this section is to provide an enumerated paradigm for software specifically in the field of numerical algebraic geometry. This detailed list is broken into four levels, beginning with very basic operations not specific to polynomial system solving at level 0 in Section 2.1 and moving through extensions of basic numerical algebraic geometry at level 3 in Section 2.4. The operations of each level build upon the capabilities of the previous level. Each of the following four sections begins with a discussion of the necessary operations of the given level and the resulting capabilities of the software. Each section then concludes with a brief list of the main operations to be implemented at that level. All operations should be implemented for various levels of precision, ideally for arbitrarily high precision.

2.1. Level 0: Basic operations. At the very core of a numerical polynomial solver, one must of course have access to basic arithmetic both

of complex numbers and complex matrices. It is important to optimize the efficiency of this arithmetic as much as possible, particularly in high precision, as most operations in numerical algebraic geometry rely heavily upon arithmetic. In addition to basic matrix arithmetic, standard techniques from numerical linear algebra are needed. Among the most important are Gaussian elimination for linear solving, QR factorization for least squares and orthogonal complements, and the SVD, or a related technique such as the efficient method of [14], for finding numerical ranks. See [5, 26] for general references on numerical linear algebra, while [14] provides a more efficient method for determining the numerical rank of a matrix.

Random numbers play a key role in numerical algebraic geometry as many statements hold generically, i.e., for almost all random choices, thereby making the resulting algorithms hold with probability one. Any standard random number generator will suffice, although it is best to have the chosen random complex numbers of approximately unit modulus, for stability. It is also important to have a consistent mechanism for extending the precision of a randomly chosen complex number. In particular, upon extending the precision of a random number α to make $\hat{\alpha}$, truncating back to lower precision, and then again extending the precision, one should once again obtain $\hat{\alpha}$.

It is of course necessary to somehow obtain the polynomials of interest from the user, although the specific procedure for doing so is implementation-specific. To build a general solver, it is important to allow the functions to be defined as expressions built from subexpressions. This is beneficial not only for ease of use, but also for efficiency and numerical stability. It is also necessary for generality to allow for homotopies that depend on parameters, including analytic expressions as discussed in more detail in the following section. If the user is specifying the entire homotopy, it is also necessary to have a way for the user to provide solutions to the start system g. Otherwise, the automatically generated start system should be solved by the software.

Regardless of how the input data is provided, parsed, and stored, it is at times necessary for the software to automatically homogenize the polynomials provided by the user. Homogenization is simply the mechanism for moving from a product of one or more complex spaces to a product of complex projective spaces. This is a purely symbolic operation which should be implemented in such a way as to be easily reversed in case the need arises. Suppose the homogenized system involves the cross product of v projective spaces. Then v random nonhomogeneous linear equations, one for each projective space in the product, should be appended to the system in order to choose a patch on each complex projective space. These v linear equations are known as the patch polynomials.

Basic path tracking, a level 1 operation, makes heavy use of both function evaluation and Jacobian evaluation. Function evaluation is straightforward, although it could be optimized via specialized techniques such as

Horner's method. The Jacobian (i.e., the matrix of partial derivatives of the polynomials) should be computed automatically by some form of automatic differentiation so that the user does not need to provide it as input. It is generally believed that the Jacobian should be computed explicitly rather than approximated numerically, for stability.

Some of the operations at this level are common to numerical software in general, regardless of the specific application. It should be noted that existing libraries, such as LAPACK, provide robust implementations of some of these operations, albeit in one level of precision.

Summary of level 0 operations:

- Complex scalar and matrix arithmetic
- Matrix operations (e.g., linear solving, numerical ranks, and orthogonal complements)
- Random number generation
- Parsing of input data
- Function homogenization
- Function evaluation
- Jacobian evaluation
- All numerical operations should be available in multiprecision.

2.2. Level 1: Basic tracking and zero-dimensional solving. It is now possible to glue together the capabilities of level 0 to build algorithms leading up to a full zero-dimensional solver, i.e., a method for computing all isolated solutions of f. The concept of "solving" a system could be interpreted two ways. Given a desired accuracy $\epsilon \in \mathbb{R}^+$, let $S := \{s \in \mathbb{C}^N : |f(s)| \leq \epsilon\}$. This set may break up into k disconnected pieces, say $S = S_1 \cup \ldots \cup S_k$. Then one interpretation of solving is to find a $z \in S_i$ for each i. The second way to interpret "solving" f is to find, for each $s \in \mathbb{C}^N$ such that $f(s) = 0$ and s has multiplicity m as a root of f, a set $\{z_1, ..., z_m\} \subset \mathbb{C}^N$ such that $|z_i - s| \leq \epsilon$ for all i. The solutions in the former sense of solving depend upon the scaling of the polynomials while those of the latter depend upon the scaling of the variable. The latter is more widely accepted as correct and is thus the definition used throughout this paper.

As discussed in Section 1, homotopy continuation casts the system f in a family of polynomial systems, another of which is g. Such parameterized families sometimes arise naturally and are of great interest in some applications. Other families are artificially constructed for the specific purpose of solving f by continuation, in a so-called *ab initio* homotopy. Often a naturally parameterized family can be used to define a homotopy that has fewer paths to follow than an *ab initio* homotopy. As a result, software in numerical algebraic geometry should address both *ab initio* and natural parameter homotopies. The natural parameter spaces may be complex analytic and thus the need for evaluating complex analytic expressions of parameters and complex numbers as mentioned in the previous section.

At its core, homotopy continuation is simply a sequence of steps advancing the path variable and updating the solution vector x to match. Each step forward consists of a predictor step followed by a sequence of corrector steps. The prediction is commonly carried out by Euler's method, which steps along the tangent to the path. This is generally regarded as an acceptable approach, although secant prediction can also be employed, as can higher order methods such as Runge-Kutta. Once a prediction is made to a new value of t, it is necessary to refine the point back towards the solution curve. This may be accomplished by fixing t and running a Newton-like corrector until the norm of the Newton residual has dropped below a prespecified tolerance.

Naturally, there are times when steps will fail, where failure is declared when the corrector does not converge sufficiently fast. Such step failures need not trigger path failure. Rather, adaptive steplength should be utilized. Upon step failure when using adaptive steplength, the steplength is decreased by a prespecified factor in the hope that convergence will occur for a smaller step. Only if progress along the path becomes excessively slow is the path declared a failure. Conversely, if the steps are progressing well, it is worthwhile to try to increase the steplength. More details regarding prediction, correction, and adaptive steplength may be found in [1, 16, 24].

Path failure may occur for a number of reasons, but the presence of a singularity, particularly at $t = 0$, is a common cause. There are several sophisticated algorithms known as endgames that help to speed convergence at $t = 0$ for both nonsingular and singular endpoints. These endgames are typically employed for every path, so it is important to have at least one implemented in any software package for numerical algebraic geometry. Details regarding endgames may be found in [17, 18, 19].

For zero-dimensional solving, polynomial systems given by the user could be nonsquare with $n > N$. Fortunately, Bertini's theorem [24] guarantees that a new system consisting of N generic linear combinations of the original n polynomials will have among its solutions all the isolated solutions of the original system, though possibly with increased multiplicity. It may also have nonsingular isolated extraneous solutions. The extraneous solutions are easily detected as they do not satisfy the original system.

Unless the user chooses to specify a parameter homotopy, the software must be able to automatically produce an appropriate start system g, solve it, and attach it to the homogeneous, square system f in order to create the homotopy H. There are several methods for producing start systems, although the general rule is that the computational cost increases in order to produce start systems with fewer paths to track. Among the common choices, total degree start systems, consisting of polynomials of the form $x_i^{d_i} - 1$, where d_i is the degree of the i^{th} polynomial in f, are trivial to build and solve but have the largest number of paths to be tracked (that being the product of the degrees of the functions). At the other end of the spectrum, the construction and solution of sophisticated polyhedral homo-

topies involve far more computation time but may result in far fewer paths (the number of which is the mixed volume). It is not clear *a priori* which type of start system is best-suited for an arbitrary polynomial system, so it is important to have multiple types of start systems available.

Once all (or most) of the aforementioned operations have been implemented, it is possible to compute the zero-dimensional solutions of a given polynomial system. Two other useful tools belong at this level. First, deflation [11] is a means of constructing a new polynomial system $\hat{f}$ from f such that $\hat{f}$ has a nonsingular solution in place of a particular singular solution of f. This makes it possible to compute singular solutions more accurately without relying on higher precision. The major drawback of implementing deflation is that decisions must be made about the rank of the Jacobian matrix at the solution point before the solution point is known accurately. The use of endgames can improve the accuracy of the solution estimate before deflation, helping to ensure that the correct deflation sequence is performed but adding the cost of endgame computations. Exploration of the numerical stability and efficiency of deflation and endgames is a topic of ongoing research. The big advantage of deflation comes when dealing with positive dimensional components of multiplicity greater than one.

The other useful tool at this level is a post-processor to manipulate and display the solutions computed by the solver as well as any statistics gathered during tracking. As the functionality of such a tool is application-specific, no more details will be discussed here.

Summary of level 1 operations:

- Differential equation solving, e.g., Euler's method
- Newton's method
- Basic path tracking with adaptive steplength control
- Adaptive precision path tracking
- Squaring of systems
- Start system and homotopy generation
- Start system solving
- Full zero-dimensional solving
- Endgames
- Deflation
- Post-processing of zero-dimensional data.

2.3. Level 2: Positive-dimensional solving. The solution set Z of a polynomial system f may have several components and these may not all have the same dimension. Letting $D := \dim Z$, the irreducible decomposition may be written as $Z = \cup_{i=0}^{D} Z_i = \cup_{i=0}^{D} \cup_{j \in \mathbb{I}_i} Z_{i,j}$, where each $Z_{i,j}$ is an irreducible component of dimension i, and accordingly each Z_i is the pure i-dimensional component of Z. (Symbol $\mathbb{I}_i$ in the above is just an index set for the components of dimension.)

One of the key objectives in numerical algebraic geometry is to find a numerical irreducible decomposition of Z, which consists of sets $W_{i,j} =$

$Z_{i,j} \cap L_{N-i}$, where L_{N-i} is a generic linear subspace of dimension $N-i$. $W_{i,j}$, together with L_{N-i}, is known as a witness set for $Z_{i,j}$, and by abuse of notation, $W_i = \cup_{j \in \mathbb{I}_i} W_{i,j}$ is called a witness set for Z_i. We briefly describe the algorithms for computing the irreducible decomposition below. Full details may be found in the references cited below or in [24].

The main steps in computing a numerical irreducible decomposition are:

- find witness supersets $\hat{W}_i \supset W_i$ for each dimension i,
- prune out "junk points" from $\hat{W}_i$ to extract the witness sets W_i, and
- break W_i into distinct sets $W_{i,j}$, the witness sets for the irreducible components $Z_{i,j}$.

The witness supersets $\hat{W}_i$ are generated by the application of zero-dimensional solving to find the isolated points in the slice $Z \cap L_{N-i}$. All of the $\hat{W}_i$, for $0 \leq i \leq D$, can be obtained using the cascade algorithm [20], starting at $i = D$ and cascading sequentially down to $i = 0$. The junk points in $\hat{W}_i$ must lie on some Z_j with $j > i$. Thus, the junk may be removed by testing each point $p \in \hat{W}_i$ for membership in a higher-dimensional component. This can be done using continuation on slices to see if any of the witness sets W_j, $j > i$ connect to p as the slicing linear space L_{N-i} is moved continuously until it contains p. The final break up of W_i into irreducibles is accomplished by first using monodromy, which comes down to discovering connections between witness points as the linear slicing space is moved around a closed loop in the associated Grassmannian [21]. This is followed by checking if the connected groups so discovered are complete, by means of the trace test [22]. The trace method may also be used to complete a partial decomposition. Both monodromy and the trace method involve specialized continuation of slices and careful bookkeeping.

Squaring is a concern for positive-dimensional solving just as it is for zero-dimensional solving. Although much carries over, one difference is that the size to which the system should be squared depends on the dimension of the component, e.g., for components of dimension k, the defining system should be randomized to a system of $N-k$ equations.

Given witness data for an algebraic set Z, there are three operations of particular interest for users. First, a user might want to find many points on a specific component. This is known as sampling and is very closely related to monodromy, as both use the continuation of slices to move witness points around on the component. Second, a user might want to know if a given point lies on an irreducible component of Z. This is the same component membership test used in the junk removal stage of computing an irreducible decomposition. Finally, a user might want the accuracy of some endpoint sharpened. This is just a matter of running Newton's method appropriately, perhaps after deflating f. Of course, as in

the previous section, a post-processor would be appropriate, although the exact functionality is again application-specific.

Deflation is particularly valuable as a method for multiple components, e.g., to track intersections of a multiplicity greater than one component with a one-parameter family of linear spaces of complementary dimension, as is done to sample a multiple component. Roughly speaking, if Z is a k-dimensional component of the solution set of a system $f = 0$, then the computation of the deflation of $Z \cap L$ for f restricted to a generic k-codimensional linear space gives rise to a "deflation" of the whole component. At the expense of increasing the number of variables, this allows us to numerically treat all components as multiplicity one components.

Summary of level 2 operations:

- Continuation of slices
- Monodromy
- Traces
- Squaring of systems for positive-dimensional solving
- Cascade algorithm
- Full numerical irreducible decomposition
- Sampling
- Component membership
- Endpoint sharpening
- Post-processing of witness data
- Deflation for components.

2.4. Level 3: Extensions and applications of the basics. This highest level consists of operations that make use of the basic numerical algebraic geometry maneuvers described in the previous three levels. For example, for two algebraic sets X and Y that are the solution sets of polynomial systems f and g, respectively, suppose one has witness sets W_X and W_Y for X and Y but would like a witness set W for $X \cap Y$. There is a now a method [23] for computing the numerical irreducible decomposition of such an intersection, the inclusion of which, while not essential for basic software in numerical algebraic geometry, will be important as the field continues to develop.

There are several other advanced algorithms that should be included in a complete state-of-the-art implementation. Another such technique is that of [25], which provides a way of finding exceptional sets, i.e., the sets of points in the parameter space above which the fiber has dimension higher than the generic fiber dimension. Fiber products play a key role in this algorithm and therefore need to be available in the software before the method for finding exceptional sets may be implemented. Also, in real-world applications, real solutions are often of more interest than complex solutions, so the extraction of real solutions from the numerical irreducible decomposition, for example, by an extension of the method of [15], would be very useful.

This is not intended to be a complete list, and it is anticipated that many more operations could be added in the near future. One capability, though, that is important now and will only become more essential over time, is parallelization. Although not every algorithm of numerical algebraic geometry is fully parallelizable, basic path tracking is easily parallelized so great savings can be made throughout levels 1, 2, and 3 by doing so [10, 12, 27, 29].

Summary of level 3 operations:

- Intersection of components
- Fiber products
- Finding exceptional sets
- Extracting real solution sets from complex components
- Parallelization.

3. Bertini. Bertini [2] is a new software package under ongoing development by the authors for computation in the field of numerical algebraic geometry. Bertini itself has evolved from a program called Polysolve created by Bates, C. Monico (Texas Tech University), Sommese and Wampler, although nothing substantial remains in Bertini from Polysolve.

Bertini is written in the C programming language and makes use of several specialized libraries, particularly lex and yacc for parsing and GMP and MPFR for multiple precision support. The beta version of Bertini was released in October 2006 to coincide with the Software for Algebraic Geometry workshop of the Institute for Mathematics and its Applications. It is currently anticipated that Bertini 1.0 will be made available to the public sometime in 2007. Bertini is currently only available as an executable file for 32- or 64-bit Linux and for Windows, via Cygwin. Specific instructions for using Bertini are included with the distribution and on the Bertini website.

Among other things, Bertini is capable of producing all complex isolated solutions of a given polynomial and witness sets for each positive-dimensional irreducible component. The goal of the Bertini development team is to eventually include in Bertini all operations described above in Sections 2.1 through 2.4. The purpose of this section is to indicate briefly which of those operations are already available in the beta version of Bertini. The specific algorithms implemented are also described, when appropriate. Details regarding the development plans for Bertini 1.0 and beyond may be found in Sections 4.1 and 4.2, respectively.

3.1. Level 0. By default, Bertini uses IEEE double precision, although it also allows for any fixed level of precision available in MPFR. In particular, precision is available starting from 64 bits, increasing in 32 bit increments. Furthermore, the beta version of Bertini allows the user to select adaptive precision for zero-dimensional solving. In adaptive precision mode, Bertini begins in double precision and increases precision as necessary, determined by the algorithm described in [3].

All matrix operations described in Section 2.1 have already been implemented in Bertini with the exception of the QR algorithm (i.e., the QR decomposition of a matrix), which will be implemented by the next release. Gaussian elimination with scaled partial pivoting is used for linear solving. The SVD algorithm is based on the description provided in [26].

Bertini uses the basic random number generator found in the C standard library. Each random number is scaled to be of the appropriate modulus and stored as an exact rational number with the denominator a power of ten. Thus, to increase precision, all new digits are set to zero, although this is not ideal. It would be better to increase precision using randomly chosen digits such that subsequent truncations and extensions result in the same additional digits. This will be changed in a future version of Bertini.

The user provides polynomials (either the entire homotopy or just the target system) to Bertini in a file which is parsed using lex and yacc. The polynomials are stored in straight-line format for easy automatic differentiation and homogenization as well as efficient evaluation. Any optimization, such as using Horner's method, is the responsibility of the user. Homogenization is carried out as the polynomials are being parsed, and differentiation is carried out after the entire system has been parsed, currently via forward automatic differentiation as described in [8].

Bertini allows the user to include constants and subfunctions when defining the target system and also the path variable and parameters when defining a homotopy. There are various restrictions placed on the homogeneous structures allowed depending upon the scenario, but multiple variable groups are generally supported. All coefficients, constants, and other numbers are stored as exact rational numbers, as described for random numbers above, and are assumed to be exact. As a result, function and Jacobian evaluation are available at any level of precision. Analytic functions of constants and parameters are not yet supported.

3.2. Level 1. All level 1 operations of Section 2.2 have been implemented in Bertini with the exception of deflation. Euler's method is used as the predictor, and Newton's method as the corrector. The fractional power series endgame [19] is available in basic precision, fixed higher precision, and adaptive precision.

All polynomial systems are automatically squared to the appropriate number of polynomials and variables, although the details of the squaring mechanism are excluded from the present paper for brevity. If the user specifies only the target system and variable groupings, Bertini multihomogenizes the system according to the groupings and attaches a compatible m-homogeneous start system via a linear homotopy. Each polynomial in the start system consists of a product of random linear functions matching the homogeneous structure of the corresponding target polynomial. If the user specifies only one variable group, the result is a total degree start system. Bertini also has a post-processor which collects the endpoint data for

the run and creates a number of files, some of which are human-readable and some of which are better suited for machine reading. The files provide lists of singular endpoints, nonsingular endpoints, real endpoints, and so on. Important path characteristics, such as an estimate of the error in the endpoint, are also supplied in these files. Bertini also indicates whether any paths failed (and why they did) and whether any path crossing occurred prior to the endgame.

3.3. Level 2. All of the operations of level 2 described in Section 2.3 have been implemented in double precision in Bertini, with the exceptions of endpoint sharpening and deflation. Users have the option of computing the numerical irreducible decomposition of the polynomial system input file, sampling a component of a system for which witness data has previously been computed, or performing component membership on such a system. The algorithms are then carried out as described in Section 2.3 and [24].

In the case of computing the numerical irreducible decomposition, Bertini stores all data in memory until the end of the run. At the end of the run, a postprocessor sorts the data and creates a number of files, as in the case of a zero-dimensional run. The structure of the algebraic set is also provided in a table on the screen, namely the number of irreducible components of each degree in each dimension. Additional details may be found in the Bertini documentation available in [2].

3.4. Level 3. None of the level 3 operations described in Section 2.4 are available in the beta version of Bertini. Much of the further development of Bertini will involve level 3 operations.

4. The future for Bertini. It is of course impossible to predict what will happen with either Bertini or numerical algebraic geometry in the distant future. However, there is a short-term development plan for Bertini. The following sections describe the development plan for Bertini 1.0 and further anticipated developments beyond Bertini 1.0.

4.1. Bertini 1.0. There are several key developments currently in the process of being implemented in order to move Bertini from the beta version to version 1.0. The four main improvements are the inclusion of the cascade algorithm in fixed multiple precision as well as adaptive precision, deflation in both the zero- and positive-dimensional cases, endpoint sharpening (as described in Section 2.2), and the QR decomposition. Other improvements are minor, such as several small changes to the output files and an improved post-processor. The goal is to have most of the operations of levels 0, 1, and 2 available in Bertini 1.0 in fixed user-specified multiprecision and adaptive precision.

4.2. Beyond Bertini 1.0. While there will certainly be a number of changes to the core functions of Bertini after version 1.0 is released (such as improving the way that random numbers are handled), many of the major

changes will involve the implementation of the operations at level 3 and new algorithms currently under development.

Other planned extensions include allowing the user to specify analytic functions of parameters and constants, the inclusion of polyhedral methods, e.g., using MixedVol [7], and various forms of parallelization. Bertini currently operates by setting up files with input data, calling Bertini, and reading the results from files. It is anticipated that Bertini will eventually include a scripting language or make use of software that provides an interactive environment. Bertini will also undergo a move from the C programming language to C++ sometime after the release of version 1.0, both for increased modularity and for easier extension. Finally, Bertini will eventually make use of more efficient multiprecision numerical libraries as they become available and will include interfaces to other software packages commonly used in the mathematics community.

REFERENCES

[1] E.L. Allgower and K. Georg, *Introduction to numerical continuation methods*, Classics in Applied Mathematics, **45**, SIAM, Philadelphia, 2003. Reprint of the 1990 edition (Springer-Verlag, Berlin).

[2] D.J. Bates, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler, *Bertini: Software for Numerical Algebraic Geometry*, available at www.nd.edu/~sommese/bertini.

[3] ———, *Adaptive multiprecision path tracking*, to appear SIAM J. Num. Anal.

[4] D.F. Davidenko, *On a new method of numerical solution of systems of nonlinear equations*, Doklady Acad. Nauk SSSR (N.S.), **88**:601–602, 1953.

[5] J.W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, 1997.

[6] T. Gao and T.Y. Li, *HOM4PS*, available at www.csulb.edu/~tgao.

[7] T. Gao, T.Y. Li, and M. Wu, *Algorithm 846: MixedVol: A software package for mixed-volume computation*, ACM Trans. Math Software, **31**(4):555–560, 2005. Software available at www.csulb.edu/~tgao.

[8] A. Griewank, *Evaluating derivatives. Principles and techniques of algorithmic differentiation*, Frontiers in Applied Mathematics, **19**, SIAM, Philadelphia, 2000.

[9] T. Gunji, S. Kim, M. Kojima, A. Takeda, K. Fujisawa, and T. Mizutani, *PHoM—a polyhedral homotopy continuation method for polynomial systems*, Computing, **73**(1):55–77, 2004. Software available at www.is.titech.ac.jp/~kojima.

[10] T. Gunji, S. Kim, K. Fujisawa, and M. Kojima, *PHoMpara—parallel implemenation of the Polyhedral Homotopy continuation Method for polynomial systems*, Computing, **77**(4):387–411, 2006.

[11] A. Leykin, J. Verschelde, and A. Zhao, *Evaluation of Jacobian matrices for Newton's method with deflation for isolated singularities of polynomial systems*, in proceedings of SNC 2005 (International Workshop on Symbolic-Numeric Computation, Xi'an, China, July 19–21, 2005), edited by Dongming Wang and Lihong Zhi, pp. 19–28, 2005.

[12] A. Leykin, J. Verschelde, and Y. Zhuang, *Parallel Homotopy Algorithms to Solve Polynomial Systems*, in proceedings of ICMS 2006 (Second International Congress on Mathematical Software, Castro Urdiales, Spain, September 1–3, 2006), Lecture Notes in Computer Science, **4151**:225–234, 2006.

[13] T.Y. Li, *Numerical solution of multivariate polynomial systems by homotopy continuation methods*, Acta Numer., **6**:399–436, 1997.

[14] T.Y. LI AND Z. ZENG, *A rank-revealing method with updating, downdating, and applications*, SIAM J. Matrix Anal. Appl., **26**(4):918–946, 2005.

[15] Y. LU, D.J. BATES, A.J. SOMMESE, AND C.W. WAMPLER, *Finding all real points of a complex curve*, to appear Contemporary Mathematics.

[16] A.P. MORGAN, *Solving polynomial systems using continuation for engineering and scientific problems*, Prentice-Hall, Englewood Cliffs, N.J., 1987.

[17] A.P. MORGAN, A.J. SOMMESE, AND C.W. WAMPLER, *Computing singular solutions to nonlinear analytic systems*, Numer. Math., **58**(7):669–684, 1991.

[18] ———, *Computing singular solutions to polynomial systems*, Adv. Appl. Math., **13**(3):305–327, 1992.

[19] ———, *A power series method for computing singular solutions to nonlinear analytic systems*, Numer. Math., **63**(3):391–409, 1992.

[20] A.J. SOMMESE AND J. VERSCHELDE, *Numerical homotopies to compute generic points on positive dimensional algebraic sets*, J. Complexity, **16**(3):572–602, 2000.

[21] A.J. SOMMESE, J. VERSCHELDE, AND C.W. WAMPLER, *Using monodromy to decompose solution sets of polynomial systems into irreducible components*, in proceedings of Applications of algebraic geometry to coding theory, physics and computation (Eilat, 2001), NATO Sci. Ser. II Math. Phys. Chem., **36**, 2001.

[22] ———, *Symmetric functions applied to decomposing solution sets of polynomial systems*, SIAM J. Num. Anal., **40**(6):2026–2046, 2002.

[23] ———, *Homotopies for intersecting solution components of polynomial systems*, SIAM J. Num. Anal., **42**(4):1552–1571, 2004.

[24] A.J. SOMMESE AND C.W. WAMPLER, *Numerical solution of systems of polynomials arising in engineering and science*, World Scientific, Singapore, 2005.

[25] ———, *Exceptional sets and fiber products*, to appear Foundations of Computational Mathematics.

[26] G.W. STEWART, *Matrix algorithms. Vol. I. Basic decompositions*, SIAM, Philadelphia, 1998.

[27] H.-J. SU, J. MCCARTHY, M. SOSONKINA, AND L.T. WATSON, *Algorithm 857: POLSYS_GLP – A parallel general linear product homotopy code for solving polynomial systems of equations*, ACM Trans. Math. Software, **32**(4):561–579, 2006. Software available at www.vrac.iastate.edu/~haijunsu.

[28] J. VERSCHELDE, *Algorithm 795: PHCpack: a general-purpose solver for polynomial systems by homotopy continuation*, ACM Trans. Math. Software, **25**(2):251–276, 1999. Software available at www.math.uic.edu/~jan.

[29] J. VERSCHELDE AND Y. ZHUANG, *Parallel implementation of the polyhedral homotopy method*, in proceedings of ICPPW '06 (International Conference Workshops on Parallel Processing), IEEE Computer Society, Washington, DC, 481–488, 2006.

[30] C.W. WAMPLER, *HomLab: Homotopy Continuation Lab*, available at www.nd.edu/~cwample1.

PHCLAB: A MATLAB/OCTAVE INTERFACE TO PHCPACK*

YUN GUAN† AND JAN VERSCHELDE‡

Abstract. PHCpack is a software package for Polynomial Homotopy Continuation, to numerically solve systems of polynomial equations. The executable program "phc" produced by PHCpack has several options (the most popular one "-b" offers a blackbox solver) and is menu driven. PHClab is a collection of scripts which call phc from within a MATLAB or Octave session. It provides an interface to the blackbox solver for finding isolated solutions. We executed the PHClab functions on our cluster computer using the MPI ToolBox (MPITB) for Octave to solve a list of polynomial systems. PHClab also interfaces to the numerical irreducible decomposition, giving access to the tools to represent, factor, and intersect positive dimensional solution sets.

1. Introduction. Polynomial systems arise in various fields of science and engineering, e.g.: the design of a robot arm [13] so its hands passes through a prescribed sequence of points in space requires the solution of a polynomial system. Homotopy continuation methods are efficient numerical algorithms to approximate all isolated solutions of a polynomial system [10]. Recently homotopies have been developed to describe positive dimensional solution sets [20].

This paper documents an interface PHClab to use the functionality provided by PHCpack [21] from within a MATLAB or Octave session. The main executable program provided by PHCpack is `phc`, available for downloading on a wide variety of computers and operating systems. The program phc requires no compilation. Its most popular mode of operation is via the blackbox option, i.e.: as `phc -b input output`. Recently the program has been updated with tools to compute a numerical irreducible decomposion [18].

The main motivation for PHClab is to make it easier to use `phc` by automatic conversions of the formats for polynomial systems (on input) and solutions (on output). Manual or adhoc conversions can be tedious and lead to errors. As PHCpack has no scripting language on its own, the second advantage of PHClab is help the user to systematically use the full capabilities of PHCpack. As MATLAB (and its freely available counterpart Octave) is a very popular scientific software system, PHClab will be a useful addition to PHCpack.

*This material is based upon work supported by the National Science Foundation under Grant No. 0105739 and Grant No. 0134611.

†Department of Mathematics, Statistics, and Computer Science, University of Illinois at Chicago, 851 South Morgan (M/C 249), Chicago, IL 60607-7045, USA (guan@math.uic.edu).

‡Department of Mathematics, Statistics, and Computer Science, University of Illinois at Chicago, 851 South Morgan (M/C 249), Chicago, IL 60607-7045, USA (jan@math.uic.edu or jan.verschelde@na-net.ornl.gov; http://www.math.uic.edu/~jan).

Another feature of PHClab is the possibility of developing parallel code at a high level, when using the MPI ToolBox (MPITB [2]) for Octave.

2. The design of PHClab. The first interface from a C program to `phc` was written by Nobuki Takayama and is still available via the PHCpack download web site. Via this interface, `phc` became part of OpenXM [11] (see also [12]). We used the same idea to build `PHCmaple` [7, 8], defining a Maple interface to PHCpack.

All that is needed to make the interface work is the executable form of `phc`. PHClab is a collection of scripts written in the language of MATLAB and Octave. These scripts call `phc` with the appropriate options and menu choices.

3. Downloading and installing. PHClab was tested on Matlab 6.5 and Octave 2.1.64 on computers running Windows and Linux. On an Apple laptop running Mac OS X version 10.3.7, we executed PHClab in Octave 2.1.57.

The most recent version of PHCpack and PHClab can be retrieved from

`http://www.math.uic.edu/~jan/download.html`

which we from now on call the download web site. To install and use PHClab, execute the following steps:

1. From the download web site, either download the source code for `phc` (a makefile is provided with the code), or select an executable version of `phc`. Currently, `phc` is available in executable form on Windows, workstations from IBM (running AIX 5.3) and SUN (running SunOS 5.8), and PCs running Linux and Mac OS X 10.3. Except for Windows (which comes just as a plain `phc.exe`), one has to run `gunzip` followed by `tar xpf` on the downloaded file.
2. The PHClab distribution is available as `PHClab.tar.gz` from the download web site. To install PHClab in the directory /tmp, save `PHClab.tar.gz` first in `/tmp`, and then execute the following sequence of commands:
   `cd /tmp; mkdir PHClab; mv /tmp/PHClab.tar.gz PHClab;`
   `cd /tmp/PHClab; gunzip PHClab.tar.gz; tar xpf PHClab.tar.`
3. Either launch MATLAB or Octave in the directory PHClab, or add the name of the directory which contains PHClab to the path of MATLAB or Octave.

The first command of PHClab to be executed is **set_phcpath**. This command takes one argument: the full path name of the file name which contains the executable program `phc`. For example, if `phc` was saved in `/tmp`, then a session with PHClab starts with **set_phcpath**(`'/tmp/phc'`).

4. Solving polynomial systems. In this section we define the basic commands to solve polynomial systems using PHClab. We first define the input/output formats, introducing the function **make_system** to convert a

matrix format for a polynomial system into a symbolic input format to phc. The blackbox solver of PHCpack is called by the command **solve_system**. Besides the solution vectors, the solver returns extra diagnostical information about the quality of each solution.

Path tracking typically starts from a generic system (without any singular solutions) to a more specific system. We use the system we first solved by the blackbox solver as start system to solve a system with specific coefficients, using the function **track**. Because of our specific choice of the coefficients, we generated a polynomial system with a double solution, i.e.: of multiplicity two. Via the function **refine_sols** and **deflation**, we respectively refine a solution and deflate its multiplicity into a regular problem.

The last function we introduce in this section is **mixed_volume**, to compute the mixed volume for a polynomial system and (optionally) create and solve a random coefficient start system. The mixed volume equals the number of roots without zero components of a polynomial system with sufficiently generic coefficients. The function **mixed_volume** calls the translated form of the code MixedVol [3].

4.1. I/O formats and the blackbox solver. The input to the solver is a system of multivariate equations with complex floating-point coefficients. For example, consider the system $g(\mathbf{x}) = \mathbf{0}$:

$$g(x_1, x_2) = \begin{cases} 1.3x_1^2 + 4.7x_2^2 - 3.1 + 2.3i = 0 \\ 2.1x_2^2 - 1.9x_1 = 0 \end{cases}, \quad \text{with } i = \sqrt{-1}. \tag{4.1}$$

This system is encoded as a matrix, with in its rows the terms of each polynomial. A zero row in the matrix marks the end of a polynomial in the system. A nonzero row in the matrix represents a term as the coefficient followed by the exponents for each variable. For example $4.7x_2^2$ is represented by the row `4.7 0 2`. If n is the number of variables and m the total number of terms, then the matrix encoding the system has $m+n$ rows and $n+1$ columns.

To solve the system $g(\mathbf{x}) = \mathbf{0}$ using PHClab, we may execute the following sequence of instructions:

```
% tableau input for a system :
t = [1.3 2 0; 4.7 0 2; -3.1 + 2.3*i 0 0; 0 0 0;
     2.1 0 2; -1.9 1 0; 0 0 0];
make_system(t)           % shows symbolic format of the system
s = solve_system(t);     % call the blackbox solver
ns = size(s,2)           % check the number of solutions
s3 = s(3)                % look at the 3rd solution
```

Then we see the following output on screen:

```
ans =
    ' + 1.3*x1**2 + 4.7*x2**2 + (-3.1+2.3*i)'
```

```
    ' + 2.1*x2**2 -1.9*x1'
ns =
     4
s3 =

            time: 1
    multiplicity: 1
             err: 4.0340e-16
             rco: 0.1243
             res: 2.7760e-16
              x1: -3.9180 + 0.3876i
              x2: 0.0930 + 1.8851i
```

We see the coordinates of the solution are in the last fields (displayed by default in short format, we may see more in format long) and extra diagnostics in the first five fields, briefly explained below.

time is the end value of the continuation parameter. If this value is not equal to one, then it means that the path tracker did not manage to reach the end of the path. This may happens with paths diverging to infinity or with highly singular solutions.

multiplicity is the multiplicity of the solution. A solution is regular when the multiplicity is one. When the approximation for a solution is not yet accurate enough, then the multiplicity might still be reported as one, although the value for `rco` might be close to the threshold.

err is the magnitude of the last update Newton's method made to the solution. At singular solutions, the polynomial functions exhibit a typical "flat" behavior. Although the residual may then be already very small, the value for this `err` can be still large.

rco is an estimate for the inverse of a condition number of the Jacobian matrix evaluated at the approximate solution. A solution is deemed singular when this number drops below the threshold value of 10^{-8}. Multiple solutions are singular. The condition number C of the Jacobian matrix measures the forward error, i.e.: if the coefficients are given with D digits precision, then the error on the approximate solution can be as large as $C \times 10^{-D}$.

res is the residual, or the magnitude of the polynomial system evaluated at the approximate solution. This residual measures the backward error: how much one should change the coefficients of the given system to have the computed approximation as the exact solution.

The values of the coordinates of the solutions are by default displayed in MATLAB's (or Octave's) `format short`. By `format long e` we can see the full length in scientific format. For the solution above, the values of `err`, `rco`, and `res` indicate an excellent quality of the computed solution.

4.2. Path tracking from a generic to a specific system. The four solutions of the system we solved are all very well conditioned, so we may use them as start solutions to solve a system with the same coefficient structure, but with more specific coefficients:

$$f(x_1, x_2) = \begin{cases} x_1^2 + 4x_2^2 - 4 = 0 \\ -2x_2^2 + x_1 - 2 = 0 \end{cases}, \quad \text{with } i = \sqrt{-1}. \tag{4.2}$$

Geometrically, the polynomials in the system $f(\mathbf{x}) = \mathbf{0}$ respectively define an ellipse and a parabola, positioned in such a way that their real intersection point is a double solution.

In the sequence of instructions below we use the function **track**, using the new system `double` (the system $f(\mathbf{x}) = \mathbf{0}$) as target system and the system `t` we solved as start system (we called it $g(\mathbf{x}) = \mathbf{0}$). Note that before calling **track**, we must set the value of time in every solution to zero, so `s` contains proper start solutions.

```
double = [1.0 2 0; 4.0 0 2; -4.0 0 0; 0 0 0;
          -2.0 0 2; +1.0 1 0; -2.0 0 0; 0 0 0];
make_system(double)              % shows system
s(1).time = 0; s(2).time = 0;    % initialize time for every
s(3).time = 0; s(4).time = 0;    %   start solution to zero
sols = track(double,t,s);        % call the path trackers
ns = size(sols,2)                % check number of solutions
s2 = sols(2)                     % look at the 2nd solution
```

The choice of the second solution was done on purpose because this solution needs extra processing. In general however, we have no control over the order in which the solutions are computed, i.e.: while every run should give the same four solutions back, the order of solutions could be permuted.

The output we see on screen of the sequence above is

```
ans =
    'x1**2 + 4*x2**2 -4'
    ' -2*x2**2 + x1 -2'
ns =
     4
s2 =
        time: 1
multiplicity: 1
         err: 4.373000000000000e-07
         rco: 3.147000000000000e-07
         res: 7.235000000000000e-13
          x1: 2.000000000000000e+00 - 5.048709793414480e-29i
          x2: -2.493339146012010e-07 - 1.879166705634450e-07i
```

Recall that we constructed the equations in our second system $f(\mathbf{x}) = \mathbf{0}$ so that there is a double solution at $(2, 0)$. However, since we are not yet close enough to the actual double solution $(2, 0)$, the magnitude of the condition

number is about 10^7, below the threshold of 10^8, so phc does not recognize the solution as a double root. We will next see how to get closer to the actual solution.

4.3. Refining and reconditioning singular solutions. To refine the solution we save in s2, we execute 10 addition Newton steps, applying **refine_sols** to the second solution s2:

```
r2 = refine_sols(double,s2,1.0e-16,1.0e-08,1.0e-16,10)
```

We allow 10 iterations (last parameter of **refine_sols**) of Newton's method, requiring that either the magnitude of the correction vector (err) or the residual (res) is less or equal than 10^{-16}, as specified respectively by the third and fifth parameter of **refine_sols**.

Below, on the output we see the estimate for the inverse condition number has decreased, along with the value for x2:

```
r2 =
        time: 1
multiplicity: 1
         err: 3.300000000000000e-09
         rco: 3.885000000000000e-09
         res: 6.427999999999999e-17
          x1: 2.000000000000000e+00 + 4.309100000000000e-41i
          x2: -2.999062183346541e-09 - 3.017695139191104e-10i
```

Now that the estimate for the inverse condition number has dropped from 10^{-7} to 10^{-9}, below the threshold of 10^{-8}, we expect this solution to be singular. To deflate the multiplicity [9] and recondition the solution, we execute

```
def_sols = deflation(double,sols);
def_sols{4,1}
```

and then we see on screen

```
ans =
        time: 1
multiplicity: 2
         err: 2.186000000000000e-07
         rco: 1.242000000000000e-01
         res: 1.003000000000000e-13
          x1: 2.000000000000010e+00 + 1.929286255918420e-14i
          x2: -1.742621478521780e-14 + 8.266179457715231e-15i
        lm_1_1: 3.077939801899640e-01 + 6.678691166401400e-01i
        lm_1_2: -6.737524546080300e-01 - 2.946929268111410e-01i
```

Notice the value rco which has increased dramatically from 3.885e-09 to 1.242e-01, as a clear indication that the solution returned by deflation is well conditioned. Yet the multiplicity is two as a solution of the original system. The deflation procedure has constructed an augmented system for which the double solution of the original system is a regular root. The

values for `lm_1_1` and `lm_1_2` are the values of the multipliers $\lambda_{1,1}$ and $\lambda_{1,2}$ used in the first deflation of the system. The number of multipliers used equals the one plus the numerical rank of the given approximate solution evaluated at the Jacobian matrix of the original system. The augmented system is returned in `def_sols{3,1}`.

4.4. Mixed volumes and random coefficient systems. In order to solve the system (4.2) we used the output of the blackbox solver on a more general system. The blackbox solver uses polyhedral homotopies [5] to solve a system with the same sparse structure but with random coefficients. Such random coefficient system has exactly as many isolated solutions as its mixed volume [10]. The function **mixed_volume** in PHClab gives access to the code MixedVol [3] as it is available as translated form in PHCpack.

If we continue our session with in `double` the tableau input for the system (4.2), then we can compute its mixed volume and solve a random coefficients start system via the following sequence of commands:

```
[v,g,s] = mixed_volume(double,1); % compute mixed volume
v                                 % check the mixed volume
ns = size(s,2)                    % check number of solutions
g                                 % random coefficient system
```

The output to these command is

```
v = 4
ns = 4
g =
{
  [1,1] =
  +( 9.51900029533701E-01 + 3.06408769087537E-01*i)*x1^2
  +( 9.94012861166580E-01 + 1.09263131180786E-01*i)
  [2,1] =
  +( 6.10442645118414E-01 - 7.92060462982993E-01*i)*x2^2
  +(-5.76175858933274E-01 - 8.17325748757804E-01*i)
}
```

5. Solving many systems. Using PHCpack from within a MATLAB or Octave session provides novel opportunities to solve polynomial systems. In this section we show how the scripting environments can help to control the quality of the developed software. The high level parallel programming capabilities of MPITB will speed up this process in a convenient manner.

5.1. Automatic testing and benchmarking. The scripting language of MATLAB and Octave lends itself very directly to automatically solving many polynomial systems, as one would do for benchmarking purposes.

We introduce another PHClab function: **read_system** which reads a polynomial system from file. The value returned by this function can be passed to the blackbox solver. The system on file must have the following

format. On the first line we have two numbers: the number of equations and variables. Thereafter follow the polynomials, each one is terminated by a semicolon. For example, the system $g(\mathbf{x}) = \mathbf{0}$ is represented as

```
2 2
 1.3*x1**2 + 4.7*x2**2 + (-3.1+2.3*i);
 2.1*x2**2 -1.9*x1;
```

Note that `i` (and `I`) may not be used to denote variables, as they both represent the imaginary unit $\sqrt{-1}$. Because `e` and `E` are used to denote floating-point numbers, `e` and `E` may not by used as the start of names of variables.

If `/tmp/Demo` contains the polynomial systems in the files with names `ku10`, `cyclic5`, `/tmp/Demo/fbrfive4`, `/tmp/Demo/game4two`, (taken from the demonstration database[1] at [21]), then the script with contents

```
f = {'/tmp/Demo/ku10'
     '/tmp/Demo/cyclic5'
     '/tmp/Demo/fbrfive4'
     '/tmp/Demo/game4two'};
for k= 1:size(f,1)
  p = read_system(f{k});
  t0 = clock;
  s = solve_system(p);
  et = etime(clock(),t0);
  n = size(s,2);
  fprintf('Found %d solutions for %s in %f sec.\n',
          n,f{k},et);
end;
```

will produce the following statistics:

```
Found 2 solutions for /tmp/Demo/ku10 in 1.819892 sec.
Found 70 solutions for /tmp/Demo/cyclic5 in 11.094403 sec.
Found 36 solutions for /tmp/Demo/fbrfive4 in 18.750158 sec.
Found 9 solutions for /tmp/Demo/game4two in 1.630962 sec.
```

5.2. Parallel scripting with MPITB. MPITB for Octave [2] extends Octave environment by using DLD functions. It allows Octave users in a computer cluster to build message-passing based parallel applications, by the means of installing the required packages and adding MPI calls to Octave scripts. To use MPITB for Octave, dynamically linked LAM/MPI libraries are required. All nodes in the cluster need to be able to access the custom-compiled Octave that supports DLD functions.

Our choice of MPITB for Octave was motivated primarily by its functionality and availability through open source. In our testing environment, the latest MPITB for Octave was compiled against LAM/MPI 7.1.2 and

[1] available at `http://www.math.uic.edu/~jan/demo.html`.

Octave 2.1.64. To illustrate conducting parallel computation with the combination of MPITB, PHClab and Octave, we used origami equations [1]. The main script is small enough to be included here:

```
function origami
%
% mpirun -c <nprocs> octave-2.1.64 -q --funcall origami
%
% The manager distributes the systems to the worker nodes
% using dynamic load balancing.  Every node writes the
% solutions to file when its job is done and sends a
% message to the manager asking % for the next job.
%
tic                              % start the timer
info = MPI_Init;                 % MPI startup
[info rank] = MPI_Comm_rank(MPI_COMM_WORLD);
[info nprc] = MPI_Comm_size(MPI_COMM_WORLD);
path(LOADPATH,'/huis/phcpack/PHClab');
set_phcpath('/huis/phcpack/PHCv2/bin/phc');
if rank == 0                     % code for the manager
   origamisys = extract_sys('alignmentequations.txt');
   distribute_tasks(nprc,origamisys);
   fprintf('elapsed time = %.3f s\n',toc);
else
   worker_solves_system();       % code for the workers
end
info = MPI_Finalize;
LAM_Clean;
quit
end
```

Each origami system described in [1] has 4 inhomogeneous equations in 4 variables and other free parameters. The mixed volume of Newton Polytopes serve as a sharp upper bound for the number of solutions of these origami systems because of the generic parameters. The output of a run on our Rocketcalc cluster configuration with 13 workers is below:

```
prompt$ mpirun -c 13 octave-2.1.64 -q --funcall origami
Task tallies:
n0   18 (local)
n01   14
n02   14
n03   14
n04   11
n05   12
n06   13
n07   13
```

```
n08    12
n09    15
n10    15
n11    17
n12    15
sum  183 (SIZE 183)
elapsed time = 371.603 s
```

6. A numerical irreducible decomposition. There is not (yet) a blackbox solver in PHCpack to compute a numerical irreducible decomposition. In the subsections below we describe the functions which call the tools of **phc**. We start by defining how we represent positive dimensional solution set.

6.1. Witness sets. To obtain a numerical representation of a positive dimensional solution set, we add as many random hyperplanes as the expected top dimension. Extra slack variables are introduced to turn the augmented system into a square system (i.e.: having as many equations as unknowns) for which we may then apply the blackbox solver.

We illustrate our methods on a special Stewart-Gough platform, which are "architecturally singular" like the so-called Griffis-Duffy platform [4], analyzed in [6]; also see [17]. Once a witness set has been computed, the numerical irreducible decomposition in PHCpack applies monodromy [15] and linear traces [16].

A witness set consists of a polynomial system and a set of solutions which satisfy this system. The polynomial system contains the original polynomial system augmented with hyperplanes whose coefficients are randomly chosen complex numbers. The number of hyperplanes added to the original system equals the dimension of the solution set. The number of solutions in the witness set equals the degree of the solution set.

There are two methods to compute witness sets. The (chronologically) first method is to work top down, starting at the top dimensional solution component and using a cascade [14] of homotopies to compute (super) witness sets as numerical representations of solution sets of all dimensions. The second method works top down, processing equation by equation [19].

6.2. Top down computation using a cascade. The input to **embed** is a system of 8 equations[2] and the number 1, which is the expected top dimension. We solve the embedded system with **solve_system** and then run **cascade** to look for isolated solutions.

```
S = read_system('gdplatB');   % read the system from file
E = embed(S,1);               % embed with 1 extra hyperplane
sols = solve_system(E);       % call the blackbox solver
size(sols,2)                  % see candidate witness #points
```

[2]Maple code to generate the equations is at http://www.math.uic.edu/~jan/FactorBench/grifdufAe1.html.

```
[sw,R] = cascade(E,sols)        % perform a cascade
```

The blackbox solver returns 40 solutions of the embedded system, which turns out the degree of the one dimension curve, because **cascade** finds no other isolated solutions. This can be read from the output shown on screen:

```
ans =

    40

sw =

                []
    [1x40 struct]

R =

            []
    {9x1 cell}
```

The function **cascade** returns two arrays. The first array contains the solutions, while the second one contains the embedded systems. A witness set for a k-dimensional solution is defined by the $(k+1)$-th entries of the arrays returned by **cascade**.

The top down approach has the disadvantage that it requires the user to enter an expected top dimension. While in many practical applications one can guess this top dimension from the context in which the application arises, the default value – taking it as high as the number of variables minus one – is often too expensive.

6.3. Bottom up computation: Equation-by-equation. The new equation-by-equation solver [19] relieves the user from submitting a top dimension and seems more flexible. A disadvantage of the solver is that its performance depends on the order of equations. For the equation describing our Griffis-Duffy platform, we move the simplest equations first.

```
p = read_system('gdplatBa')
[sw,R] = eqnbyeqn(p)

p =

    ' g0*h0+g1*h1+g2*h2+g3*h3'
    ' g0^2+g1^2+g2^2+g3^2-h0^2-h1^2-h2^2-h3^2'
    [1x102 char]
    [1x308 char]
    [1x333 char]
    [1x333 char]
    [1x308 char]
```

```
    [1x308 char]

sw =

                []
    [1x40 struct]

R =

           []
    {9x1 cell}
```

6.4. Factoring into irreducible components. We continue with the output (sw,R), computed either with **cascade** or **eqnbyeqn**.

```
dc = decompose(R{2},sw{2,1})

ans =

    40

irreducible factor 1:

ans =

1x28 struct array with fields:
    time
    multiplicity
    err
    rco
    res
    h0
    h1
    h2
    h3
    g3
    g1
    g2
    g0
    zz1

irreducible factor 2:

ans =

        time: 1
multiplicity: 1
```

```
      err: 5.103000000000000e-15
      rco: 1
      res: 3.598000000000000e-15
       h0: -3.091000000000000e-01 - 2.563000000000000e-01i
       h1: -4.439300000000000e-01 + 5.353800000000000e-01i
       h2: 2.563000000000000e-01 - 3.091000000000000e-01i
       h3: -5.353800000000000e-01 - 4.439300000000000e-01i
       g3: 4.766100000000000e-01 + 8.732700000000000e-01i
       g1: 1.164500000000000e+00 - 2.351500000000000e-01i
       g2: -3.360600000000000e-01 + 4.145700000000000e-01i
       g0: -3.643000000000000e-03 + 8.404300000000000e-01i
      zz1: 8.171700000000000e-16 + 5.026400000000000e-16i

.. % 12 more similar linear factors not shown to save space

dc =

    [1x28 struct]
    [1x1  struct]

.. % 12 more similar structs not shown to save space
```

The output of **decompose** shows one irreducible component of degree 28 and 12 lines.

Acknowledgments. This paper was written while the second author was a long term visitor at the IMA in the Fall of 2006, participating to the annual thematic program on Applications of Algebraic Geometry. The first author is grateful for financial support to present this work at the IMA software workshop, held at the IMA, 23–27 October, 2006. The first author also thanks Javier Fernández Baldomero, the author of MPITB, and Michael Creel for helpful communication. We thank Bernd Sturmfels for forwarding origami equations [1] to the IMA workshop "Software for Algebraic Geometry" as a challenge problem.

APPENDIX

A. Alphabetic List of PHClab Functions. Below is an alphabetic list of the functions offered by PHClab.

cascade executes a sequence of homotopies, starting at the top dimensional solution set to find super witness sets. The input consists of an embedded system (the output of **embed**) and its solutions (typically obtained via **solve_system**. The output of this function is a sequence of super witness sets. A witness set is a numerical

representation for a positive dimensional solution set. The "super" means that the k-th super witness set may have junk points on solutions sets of dimension higher than k.

decompose takes a witness set on input and decomposes it into irreducible factors. The witness set is represented by two input parameters: an embedded system and solutions which satisfy it. The number of solutions equal the degree of the pure dimensional solution set represented by the witness set. On return is a sequence of witness sets, each witness set in the sequence corresponds to one irreducible component.

deflation reconditions isolated singular solutions. The input consists of two parameters: a polynomial system and a sequence of approximate solutions to the system. Typically these solutions are obtained via the blackbox solver or as the output of the function **track**. On return is a list of reconditioned solutions, along with the augmented systems which have as regular solution the multiple solution of the original system.

embed adds extra hyperplanes and slack variables to a system, as many as the expected top dimension of the solution set. There are two input parameters: a polynomial system and the number of hyperplanes which have to be added. Typically, this number is the top dimension of the solution set. If nothing is known about this top dimension, a default value for this number is the number of variables minus one.

eqnbyeqn solves polynomial systems equation by equation. For the polynomial system on input, this function returns a sequence of witness sets. The kth witness set in the sequence is a numerical representation of the solution set of dimension k.

make_system converts the matrix format of a system into a symbolic format acceptable to `phc`. A polynomial system of N equations in n variables, with a total of m terms, is represented by a matrix with $N+m$ rows and $n+1$ columns. Each polynomial is terminated by a zero row in the matrix. Each row represents one term in a polynomial, starting with its (complex) coefficient and continuing with the values of the exponents for each variable.

mixed_volume computes the mixed volume for a system of n equations in n variables. There are two input parameters: the system and a flag to indicate whether a random coefficient system must be created and solved. If the flag on input is one, then on return is a start system which has as many solutions as the mixed volume.

phc_filter removes from a super witness set those junk points while lie on a higher dimensional solution set. The third and last input parameter is a set of points to be filtered. The first two parameters represent a witness set, given by an embedded system and a sequence of solutions which satisfy the embedded system. On

return are those points of the third input that do not lie on the component represented by the witness set.

read_system reads a polynomial system from file. There is only one input parameter: a file name. The format of the polynomial system on file must follow the input format of PHCpack. The function returns an array of strings, each string in the array is a multivariate polynomial in symbolic format.

refine_sols applies Newton's method to refine a solution. There are six input parameters: a polynomial system, an approximate solution, a tolerance for the magnitude of the correction vector `err`, a threshold for to decide whether a solution is singular (relative to `rco`), a tolerance for the residual `res`, and finally a natural number with the maximal number of Newton iterations that are allowed. On return is an array of refined approximate solutions.

set_phcpath defines the directory where the executable version of phc is. For example, if the program `phc` is in the directory `/tmp`, then **set_phcpath**('/tmp/phc') must be executed at the start of a PHClab session. On Windows, `'/tmp/phc'` could be replaced by `'C:/Downloads/phc'` if `phc.exe` is in the directory `Downloads` on the C drive.

solve_system calls the blackbox solver of phc. On input is a polynomial system in matrix format, see the input description for the command **make_system**. An alternative input format is the cell array returned by **read_system**. The output is an array of structures. Every element in the array contains one solution at the end of a solution path. In addition to the values for the coordinates of the solution, an estimate for the condition number of the solution which leads to a measure for the forward error, while the residual measures the backward error.

track applies numerical continuation methods for a homotopy between start and target system, for a specified set of start solutions. The three arguments for **track** are respectively the target system, the start system and the solutions of the start system. The target and start system must be given in matrix format. If the start solutions are singular, then the path tracker will fail to start. The output of **track** is an array of the same length as the array of start solutions, containing the values at the end of the solution paths.

B. Exercises.

1. Use the blackbox solver to solve (the `phc` input format is on the right):

$$\begin{cases} x^2 + y^2 - 1 = 0 \\ x^3 + y^3 - 1 = 0 \end{cases} \qquad \begin{array}{l} 2 \\ \texttt{x\^{}2 + y\^{}2 - 1;} \\ \texttt{x\^{}3 + y\^{}3 - 1;} \end{array} \tag{B.1}$$

The exact solutions are (1,0), (0,1), $(-1+i\frac{\sqrt{2}}{2}, -1-i\frac{\sqrt{2}}{2})$ and $(-1-i\frac{\sqrt{2}}{2}, -1+i\frac{\sqrt{2}}{2})$.
How many solutions does **solve_system** return? Verify whether the output matches the exact solutions. Use **refine_sols** to discover what the multiplicities of the solutions (0,1) and (1,0) are.

2. If we have to solve repeatedly a polynomial system with the same structure, we may want to save a start system. To solve systems with the same monomials as in (B.1), we could use

$$g(x,y) = \begin{cases} x^2 + 1.232y^2 + 1.1211i = 0 \\ y^3 - 0.872y^2 - 0.6231 + 1.032i = 0 \end{cases} \tag{B.2}$$

Since the coefficients are random complex numbers (feel free to make other *random* choices) all solutions of the system $g(x,y) = \mathbf{0}$ will be regular.

   (a) Solve the system $g(x,y) = \mathbf{0}$, using **solve_system**. Verify that all solutions are regular.
   (b) Use **track** to solve the system in (B.1).
   Check whether you find the same solutions, eventually computed in a different order.

3. The following system has multiple roots:

$$\begin{cases} x^2 + y - 3 = 0 \\ x + 0.125y^2 - 1.5 = 0 \end{cases} \tag{B.3}$$

   (a) Use **solve_system** to find approximate roots. Can you see which roots are multiple?
   (b) Apply **deflation** to the approximate roots.
   Observe the values of the field `rco` of the solutions before and after the deflation.
   (c) What is the multiplicity of each solution?

4. All adjacent minors of an indeterminate 2-by-4 matrix for a system of 3 equations in 8 variables:

$$\begin{cases} x_{11}x_{22} - x_{21}x_{12} = 0 \\ x_{12}x_{23} - x_{22}x_{13} = 0 \\ x_{13}x_{24} - x_{23}x_{14} = 0. \end{cases} \tag{B.4}$$

   (a) Use **embed** to add 5 random hyperplanes.
   (b) Solve the embedded system. What is the degree of this 5-dimensional solution set?
   (c) Apply **decompose** to factor the solution set. How many irreducible factors do you find?
   (d) Repeat the process for larger instances of this problem, for $n = 5, 6, \ldots$.

5. Consider the system

$$\begin{cases} (x_1^2 - x_2)(x_1 - 0.5) = 0 \\ (x_1^3 - x_3)(x_2 - 0.5) = 0 \\ (x_1 x_2 - x_3)(x_3 - 0.5) = 0. \end{cases} \quad \text{(B.5)}$$

Solving this system means to compute witness sets for all irreducible factors.

(a) Use **embed** to add 1 random hyperplane.
(b) Solve the embedded system with **solve_system.**
Among the solutions, can you see the three witness points on the twisted cubic?
Look for solutions with a slack variable close to zero.
(c) Apply **cascade** to find candidate isolated solutions.
(d) Use **phc_filter** to filter the candidate isolated solutions.
How many isolated solutions does the system have?

REFERENCES

[1] R.C. Alperin and R.J. Lang. One and Two-Fold Origami Axioms *2006 4OSME Proceedings*, Pasadena, CA, A.K.Peters, 2007.

[2] J. Fernández, M. Anguita, E. Ros, and J.L. Bernier. SCE Toolboxes for the development of high-level parallel applications. *Proceedings of ICCS 2006*, Volume **3992** of Lecture Notes in Computer Science, pages 518–525, Springer-Verlag, 2006

[3] T. Gao, T.Y. Li, and M. Wu. Algorithm 846: MixedVol: a software package for mixed-volume computation. *ACM Trans. Math. Softw.*, **31**(4):555–560, 2005.

[4] M. Griffis and J. Duffy. Method and apparatus for controlling geometrically simple parallel mechanisms with distinctive connections. US Patent 5,179,525, 1993.

[5] B. Huber and B. Sturmfels. A polyhedral method for solving sparse polynomial systems. *Math. Comp.*, **64**(212):1541–1555, 1995.

[6] M.L. Husty and A. Karger. Self-motions of Griffis-Duffy type parallel manipulators. *Proc. 2000 IEEE Int. Conf. Robotics and Automation*, CDROM, San Francisco, CA, April 24–28, 2000.

[7] A. Leykin and J. Verschelde. PHCmaple: A Maple interface to the numerical homotopy algorithms in PHCpack. In Quoc-Nam Tran, editor, *Proceedings of the Tenth International Conference on Applications of Computer Algebra (ACA'2004)*, pages 139–147, 2004.

[8] A. Leykin and J. Verschelde. Interfacing with the numerical homotopy algorithms in PHCpack. In Nobuki Takayama and Andres Iglesias, editors, *Proceedings of ICMS 2006.* Volume **4151** of Lecture Notes in Computer Science, pages 354–360, Springer-Verlag, 2006.

[9] A. Leykin, J. Verschelde, and A. Zhao. Newton's method with deflation for isolated singularities of polynomial systems. *Theoretical Computer Science* **359**(1–3):111–122, 2006.

[10] T.Y. Li. Numerical solution of polynomial systems by homotopy continuation methods. In F. Cucker, editor, *Handbook of Numerical Analysis. Volume XI. Special Volume: Foundations of Computational Mathematics*, pages 209–304. North-Holland, 2003.

[11] M. Maekawa, M. Noro, K. Ohara, Y. Okutani, N. Takayama, and Y. Tamura. OpenXM – an open system to integrate mathematical softwares. Available at `http://www.OpenXM.org/`.

[12] M. MAEKAWA, M. NORO, K. OHARA, N. TAKAYAMA, AND Y. TAMURA. The design and implementation of OpenXM-RFC 100 and 101. In K. Shirayanagi and K. Yokoyama, editors, *Computer mathematics. Proceedings of the Fifth Asian Symposium (ASCM 2001) Matsuyama, Japan 26–28, September 2001*, Volume **9** of *Lecture Notes Series on Computing*, pages 102–111. World Scientific, 2001.

[13] J.M. MCCARTHY. *Geometric Design of Linkages.* Volume 11 of Interdisciplinary Applied Mathematics, Springer-Verlag, 2000.

[14] A.J. SOMMESE AND J. VERSCHELDE. Numerical homotopies to compute generic points on positive dimensional algebraic sets. *J. of Complexity*, **16**(3):572–602, 2000.

[15] A.J. SOMMESE, J. VERSCHELDE, AND C.W. WAMPLER. Using monodromy to decompose solution sets of polynomial systems into irreducible components. In C. Ciliberto, F. Hirzebruch, R. Miranda, and M. Teicher, editors, *Application of Algebraic Geometry to Coding Theory, Physics and Computation*, pages 297–315. Kluwer Academic Publishers, 2001. Proceedings of a NATO Conference, February 25–March 1, 2001, Eilat, Israel.

[16] A.J. SOMMESE, J. VERSCHELDE, AND C.W. WAMPLER. Symmetric functions applied to decomposing solution sets of polynomial systems. *SIAM J. Numer. Anal.*, **40**(6):2026–2046, 2002.

[17] A.J. SOMMESE, J. VERSCHELDE, AND C.W. WAMPLER. Advances in polynomial continuation for solving problems in kinematics. *ASME Journal of Mechanical Design* **126**(2):262–268, 2004.

[18] A.J. SOMMESE, J. VERSCHELDE, AND C.W. WAMPLER. Numerical irreducible decomposition using PHCpack. In M. Joswig and N. Takayama, editors, *Algebra, Geometry, and Software Systems*, pages 109–130. Springer-Verlag, 2003.

[19] A.J. SOMMESE, J. VERSCHELDE, AND C.W. WAMPLER. Solving polynomial systems equation by equation. Accepted for publication in the IMA volume on Algorithms for Algebraic Geometry.

[20] A.J. SOMMESE AND C.W. WAMPLER. *The Numerical solution of systems of polynomials arising in engineering and science.* World Scientific Press, Singapore, 2005.

[21] J. VERSCHELDE. Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation. *ACM Trans. Math. Softw.*, **25**(2):251–276, 1999. Software available at `http://www.math.uic.edu/~jan`.

COMPUTING GRÖBNER FANS AND TROPICAL VARIETIES IN GFAN*

ANDERS NEDERGAARD JENSEN†

Abstract. The Gröbner fan of an ideal in the polynomial ring in n variables is an n-dimensional polyhedral complex and the tropical variety of the ideal is a certain subcomplex. In this paper we describe the software Gfan for computing these fans. Computing the Gröbner fan is equivalent to computing all the reduced Gröbner bases of the ideal.

Key words. Gröbner fans, tropical geometry, Gfan, software.

1. Introduction. Gfan [13] is a software package for computing Gröbner fans and tropical varieties of polynomial ideals. The Gröbner fan of an ideal $I \subseteq \mathbb{Q}[x_1, \ldots, x_n]$ is a polyhedral complex defined in [15]. For a homogeneous ideal the Gröbner fan is a complete fan and the normal fan of a polytope. Its cones are in bijection with the various initial ideals of I. In particular, the full-dimensional cones are in bijection with the monomial initial ideals and thereby also in bijection with the reduced Gröbner bases of I. In [4] the local basis change of Gröbner bases was introduced. This method allows us to go from one Gröbner basis in the fan to a neighboring one, giving an effective algorithm for computing the Gröbner fan by traversing its maximal cones. This algorithm is the foundation of Gfan. A detailed description of the algorithm is given in [6].

A computation of the Gröbner fan of I is useful when searching for an initial ideal with a particular property. The computer program TiGERS [11] is an earlier implementation of the Gröbner fan traversal for toric ideals. Gfan is more general as it can compute the Gröbner fan of any polynomial ideal $I \subseteq \mathbb{Q}[x_1, \ldots, x_n]$. Taking the union of all reduced Gröbner bases of I we get a *universal* Gröbner basis, that is, a set which is a Gröbner basis with respect to any term order.

Recently the field of tropical mathematics has received much attention. In tropical mathematics the semi-ring $(\mathbb{R}, \max, +)$ is considered. Here maximum takes the role of addition and plus the role of multiplication. Hence tropical polynomials define piecewise linear functions. In tropical mathematics classical objects have tropical analogs. For example the tropical variety $\mathcal{T}(I)$ of a polynomial ideal $I \subseteq \mathbb{Q}[x_1, \ldots, x_n]$ is the analog of the usual variety of the ideal. A tropical variety is a piecewise linear object. There are many equivalent ways of defining the tropical variety of I. We prefer to state the definition in terms of initial ideals:

*Research partially supported by the Faculty of Science, University of Aarhus, Danish Research Training Council (Forskeruddannelsesrådet, FUR), Institute for Operations Research ETH, the Swiss National Science Foundation Project 200021-105202, and Institute for Mathematics and its Applications, University of Minnesota.

†Institat für Mathematik, Technische Universität Berlin, D-10623, Germany.

$$\mathcal{T}(I) := \{\omega \in \mathbb{R}^n : \text{in}_\omega(I) \text{ contains no monomials}\}$$

where $\text{in}_\omega(I)$ denotes the initial ideal of I with respect to the vector ω. This definition appeared in [17]. The fact that the tropical variety is a union of Gröbner cones ensures that the tropical variety can be given the structure of a polyhedral complex making it a subfan of the Gröbner fan. An exhaustive traversal algorithm for computing tropical varieties of prime ideals was developed in [3]. The algorithm relies on the relation to Gröbner fans, a connectivity result, constructions of tropical bases, polyhedral computations and lifting results. Gfan contains the only existing implementation of this algorithm.

Gfan is a collection of command line tools written in C++. The libraries GMP [8] and cddlib [5] are used for exact arithmetic and polyhedral computations. Gfan contains a simple implementation of Buchberger's algorithm which is sufficient for the kind of enumerations Gfan does but which is not competitive to implementations found in other algebra software. Gfan is released under the GNU GPL license, runs on Linux and Mac OS X, and can be compiled with newer versions of gcc.

This paper is organized as follows. The first section gives an introduction to Gröbner fans while illustrating how these may be computed in Gfan. After this follows a similar section about tropical varieties. We then discuss some of the algorithms in Gfan, give a few benchmark examples and discuss how Gfan can interface other mathematical programs. This paper is an extended version of [14].

2. Computing Gröbner fans. The Gfan package consists of several command line programs. The most important one is `gfan` which takes generators for an ideal $I \subseteq \mathbb{Q}[x_1, \ldots, x_n]$ and computes the complete set of reduced Gröbner bases for I.

EXAMPLE 1. *Running the command* `gfan` *and typing in the input* `{a-b-ab, a^2+ab}` *produces the list*

```
{{b^3-2*b^2,
a-b+b^2}

,
{b^2-b+a,
a*b+b-a,
a^2-b+a}

,
{b-a-a^2,
a^3+2*a^2}}
```

of all marked reduced Gröbner bases for the ideal generated by the input polynomials.

By a *marked* Gröbner basis we mean a basis where each polynomial has a distinguished, marked term. In Gfan the marked term of a polynomial is the first term listed. Furthermore, the marked term should be the initial term of the polynomial with respect to the term order.

Typically a user will ask Gfan to manipulate files instead of reading from the keyboard and writing to the screen. This is achieved in the shell by redirecting the standard input and output as follows.

```
gfan <generators.txt >listofbases.txt
```

Here the ideal is read in from the file `generators.txt` and the output is written to the file `listofbases.txt`.

We will explain the connection to polyhedral geometry in the following. Recall that given a term order $\prec$ on $\mathbb{Q}[x_1, \ldots, x_n]$ and a vector $\omega \in \mathbb{R}^n_{\geq 0}$ we may define the term order $\prec_\omega$:

$$x^u \prec_\omega x^v \Leftrightarrow u \cdot \omega < v \cdot \omega \vee (u \cdot \omega = v \cdot \omega \wedge x^u \prec x^v) \quad \text{for } u, v \in \mathbb{N}^n.$$

Here we use the notation $x^w = x_1^{w_1} \cdots x_n^{w_n}$ for $w \in \mathbb{N}^n$. The term order $\prec_\omega$ is called the *refinement* of ω with respect to $\prec$. Furthermore, recall that Buchberger's algorithm for computing a reduced Gröbner basis takes as input an ideal and a term order. If the term order is of the above form $\prec_\omega$, then different ω may lead to different reduced Gröbner bases. The Gröbner fan of I encodes which vectors give the same marked reduced Gröbner basis. For each marked reduced Gröbner basis $\mathcal{G}$ we consider the region of vectors in $\mathbb{R}^n_{\geq 0}$ that will produce $\mathcal{G}$ with Buchberger's algorithm. To avoid ambiguities we take the closure of this region and call it the *Gröbner cone* of $\mathcal{G}$. The collection of all Gröbner cones is called the (restricted) *Gröbner fan* of I. We shall give a more precise definition in the case of homogeneous ideals later.

It is well-known (see [11]) that we have bijections between

- the marked reduced Gröbner bases of I,
- the full-dimensional cones in the Gröbner fan of I, and
- the monomial initial ideals of I (with respect to term orders).

Here the bijection between the marked Gröbner bases and the initial ideals follows from uniqueness of reduced Gröbner bases while the bijection between the two first sets follows from the definition of the Gröbner fan and an argument that the Gröbner cones for term orders are full-dimensional; see [19, Proposition 1.13]. The bijections are illustrated in Figure 1.

In Example 1 we computed the Gröbner fan of I, although, it was given to us in a rather inconvenient way as a collection of marked Gröbner bases. In the next example we show how to compute defining equations for the Gröbner cones. Notice that in this paper we describe the behavior of Gfan version 0.2.2. Future versions will be slightly different.

EXAMPLE 2. *Consider the first marked reduced Gröbner basis in Example 1:*

$$\mathcal{G} = \{\underline{b^3} - 2b^2, \underline{a} - b + b^2\}.$$

We may ask ourselves for which term orders $\prec_\omega$ this is a Gröbner basis. An important observation is that Buchberger's S-pair criterion only depends

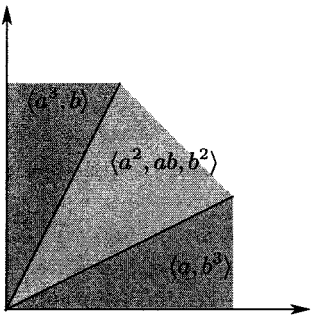


FIGURE 1. *The Gröbner fan in $\mathbb{R}^2_{\geq 0}$ of the ideal in Example 1. For each Gröbner cone its monomial initial ideal has been listed.*

on the marked terms - not on the term order. Hence, any $\omega \in \mathbb{R}^2_{\geq 0}$ that picks out the marked terms as initial monomials will define a term order for which $\mathcal{G}$ is a Gröbner basis. To pick out the marked terms, ω must satisfy:

$$\begin{array}{rcl} 3\omega_2 & > & 2\omega_2 \quad , \\ \omega_1 & > & \omega_2 \quad , \text{ and} \\ \omega_1 & > & 2\omega_2 \quad . \end{array}$$

These inequalities, of which $\omega_1 > \omega_2$ is redundant, define the open Gröbner cone of $\mathcal{G}$. The command `gfan_groebnercone` *will take the marked reduced Gröbner basis as input and produce the defining inequalities while* `gfan_facets` *will cut the set of inequalities down to the ones defining facets. The following command line combines the programs in the shell and produces the list* `{(0,1),(1,-2)}` *of inner facet normals:*

```
gfan_groebnercone | gfan_facets
```

We now show how to use Gfan for making drawings of Gröbner fans.

EXAMPLE 3. *In the case of three variables Gfan can draw a picture of the positive orthant for us. For example:*

```
gfan <input.txt | gfan_render > picture1.fig
```

produces the Xfig drawing of the Gröbner fan of the ideal in `input.txt`*; see Figure 2.*

Sometimes the ideals we consider have symmetry. This symmetry can be exploited by Gfan. The advantage is that the computation is faster and that the output may be more readable.

EXAMPLE 4. *Consider the ideal $I = \langle a^2b - c, b^2c - a, c^2a - b \rangle$. The symmetric group S_3 acts on the polynomial ring $\mathbb{Q}[a, b, c]$ by permuting variables. It also acts on $\mathbb{R}^3_{\geq 0}$ by permuting coordinates. The alternating subgroup $A_3 \subset S_3$ keeps I fixed. As a result A_3 acts on the set of Gröbner cones of I. We wish to exploit the $A_3 \subset S_3$ symmetry. We will use the command line* `gfan --symmetry | gfan_render > output.fig` *and the*

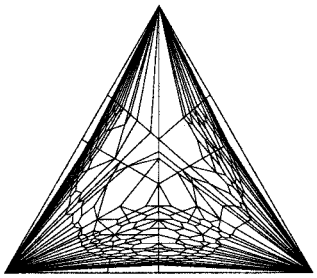


FIGURE 2. *The Gröbner fan of the ideal* $\langle x_1^5 + x_2^3 + x_3^2 - 1, x_1^2 + x_2^2 + x_3 - 1, x_1^6 + x_2^5 + x_3^3 - 1\rangle$ *intersected with the standard 2-simplex in* $\mathbb{R}^3$. *See [19, Example 3.9].*

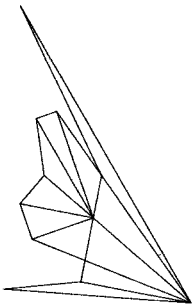


FIGURE 3. *In Example 4 Gfan only produces one Gröbner cone in each orbit with respect to the action of* A_3. *For this reason cones are missing in the picture.*

input will be a list of generators for the ideal plus a list of permutations generating A_3. *In general we will need several generators to describe a subgroup of* S_n, *but in the case of* A_3 *one generator suffices.*

```
{a^2b-c,b^2c-a,c^2a-b}
{(1,2,0)}
```

The output is the Xfig-file shown in Figure 3.

Gfan outputs only one Gröbner cone in each orbit. When a new Gröbner basis is computed it is checked if it is in the orbit of a previously computed basis. In this way Gfan avoids computing the entire Gröbner fan. Checking if two Gröbner bases are in the same orbit is time consuming put cheap compared to doing the symmetric Gröbner basis computations needed for a complete Gröbner fan traversal. The symmetry techniques used in Gfan are similar to those used in TOPCOM [16] for computing the regular triangulations of a point configuration up to symmetry.

Before we move on to discuss tropical varieties, we will need the formal definition of the Gröbner fan of a homogeneous ideal as it was defined by Mora and Robbiano [15]. Recall that for $\omega \in \mathbb{R}^n$, $\mathrm{in}_\omega(I)$ is the initial ideal

generated by the ω-initial forms of elements in I and that this initial ideal does not have to be a monomial ideal.

DEFINITION 2.1. *Let $I \subseteq \mathbb{Q}[x_1, \ldots, x_n]$ be a homogeneous ideal. We define the equivalence relation $\sim$ on $\mathbb{R}^n$ as follows:*

$$u \sim v \Leftrightarrow \mathrm{in}_u(I) = \mathrm{in}_v(I),$$

where $u, v \in \mathbb{R}^n$. The equivalence classes are relatively open polyhedral cones in $\mathbb{R}^n$. Their usual topological closures are called the Gröbner cones *of I. The set of all Gröbner cones form the* Gröbner fan *of I.*

A Gröbner fan is indeed a polyhedral fan with nice intersection properties of its cones. See [19] or [6] for a proof. In particular it contains cones of different dimensions and not just the full-dimensional cones we discussed earlier. For a homogeneous ideal, Gfan lets you compute the f-vector of the fan with the line.

```
gfan | gfan_fvector
```

The intersection of all cones in the Gröbner fan of an ideal I is again a Gröbner cone. We call this Gröbner cone the *homogeneity space* of I. It may be computed with the command `gfan_homogeneityspace` which takes a reduced Gröbner basis of the ideal I as input. The homogeneity space of I consists of all vectors which induce gradings on $\mathbb{Q}[x_1, \ldots, x_n]$ for which I is homogeneous. The homogeneity space is a subspace and a face of every cone in the Gröbner fan.

A note on the different definitions of Gröbner fans related to Gfan and the connection between them is in place. For a homogeneous ideal the restricted Gröbner fan which we started out by defining is gotten by taking the common refinement of the Gröbner fan defined above and the fan consisting of the faces of the non-negative orthant. In [6] a third uniform Gröbner fan definition is given which for homogeneous ideals coincides with the Gröbner fan definition above and for non-homogeneous ideal has the property that any cone is the closure of an equivalence class in the sense above. Again taking the common refinement of this fan with the non-negative orthant we get the restricted Gröbner fan. While the last uniform definition has the nicest properties, from a practical point of view it is less important which definition we use since for all three the maximal cones are in bijection.

As a last feature for computing Gröbner fans we should mention the program `gfan_interactive`. This program lets the user walk around in the full-dimensional cones of the Gröbner fan. At each cone the user gets to choose a facet to walk through. This is useful when studying the Gröbner fan locally or when searching for an initial ideal with a special property.

3. Computing tropical varieties. The tropical variety of an ideal $I \subseteq \mathbb{Q}[x_1, \ldots, x_n]$ is a certain subset of $\mathbb{R}^n$, namely, it consists of all $\omega \in \mathbb{R}^n$ such that $\mathrm{in}_\omega(I)$ does not contain a monomial. Besides considering the

tropical variety as a set we may also consider it as a subfan of the Gröbner fan by considering all Gröbner cones whose initial ideal is monomial-free. Since the tropical variety is a closed set, this actually defines a subfan. There is nothing special about the fan structure that is induced by the Gröbner fan — there may be other coverings of the tropical variety with a fewer number of cones. An example of this is the ideal in [3, Example 20] which is generated by the 3×3-minors of a generic 4×4 Hankel matrix. Gfan uses the structure induced by the Gröbner fan for its tropical computations.

More generally one may be interested in ideals in polynomial rings over different fields, for example over a field with a valuation being taken into account when defining initial ideals. Most examples in practice, however, reduce to the situation of ideals in $\mathbb{Q}[x_1,\dots,x_n]$.

Gfan can handle four cases of tropical varieties:

- tropical hypersurfaces,
- intersections of tropical hypersurfaces,
- tropical curves, and
- varieties defined by prime ideals.

We shall discuss these cases below.

In the special case of a principal ideal $I=\langle f\rangle$ computing the tropical variety is straight forward. In this case the variety is called a *tropical hypersurface.* It is the codimension 1 skeleton of the normal fan of the Newton polytope of f. We may compute it in the following way with Gfan.

EXAMPLE 5. *If we run* `gfan_tropicalintersection --incidence` *on the input* `{a+b+c+d}` *we get the following as a part of the output:*

```
Rays:
{
0: (-1,0,0,0),
1: (0,-1,0,0),
2: (1,1,1,0),
3: (0,0,-1,0)}
Printing index list for dimension 3 cones:
{
{0,1},
{0,2},
{0,3},
{2,3},
{1,2},
{1,3}}
F-vector:
(4,6)
```

The first list is the set of rays in the tropical hypersurface. The second list shows how these rays are combined to form the maximal cones in the hypersurface. Notice that the homogeneity space of the ideal is 1-dimensional. Hence, the rays are really 2-dimensional and the maximal

cones have dimension 3. For example the first ray $(-1,0,0,0)$ *is the cone* $\{s(-1,0,0,0)+t(1,1,1,1)$ *where* $(s,t) \in \mathbb{R}_{\geq 0} \times \mathbb{R}\}$. *If the hypersurface consists of cones of other dimensions, then these will also be listed.*

The command `gfan_tropicalintersection` is actually more general than illustrated above. It takes a list of polynomials, each defining a tropical hypersurface and computes the intersection of these hypersurfaces. The result is a polyhedral fan which is the common refinement of all the input tropical hypersurfaces. This intersection is not a tropical variety in general. Take for example the two tropical hypersurfaces defined by the polynomials $a+b+ab^2$ and $b+1$. Their intersection is the non-negative a-axis which is not a tropical variety since it does not satisfy the tropical zero-tension condition at 0.

An alternative way of defining the tropical variety $\mathcal{T}(I)$ of an ideal I is as the intersection of *all* tropical hypersurfaces of polynomials in I. A finite set of polynomials generating an ideal I is called a *tropical basis* for I if the intersection of the hypersurfaces the polynomials define equals $\mathcal{T}(I)$. Gfan contains the program `gfan_tropicalbasis` which will compute a tropical basis for an ideal defining a tropical curve. Here an ideal is said to define a *tropical curve* if it has dimension one larger than the dimension of its homogeneity space. We should point out that it is important that the ideal defines a curve. The reason for this can be found in the proof of [3, Algorithm 5]. Basically, what is needed is that it suffices to check if a single relative interior vector of a cone is in a tropical variety to know that the cone is contained in the tropical variety. This is not true in general.

EXAMPLE 6. *Consider the ideal* $I \subseteq \mathbb{Q}[a,b,c,d]$ *generated by* $\{ab+aa+cc+dd, ba+ca+db\}$. *The homogeneity space of* I *has dimension* 1 *and the Krull dimension of* $\mathbb{Q}[a,b,c,d]/I$ *is* 2. *Hence the ideal defines a tropical curve and we may use* `gfan_tropicalbasis` *to compute a tropical basis of* I. *This gives the following output:*

```
{
a*b+d^2+c^2+a^2,
a*b+b*d+a*c,
a^2*b-1/2*c*d^2-1/2*c^3+b*c*d+1/2*b*c^2-1/2*b^2*d-1/2*a*c*d+a*c^2}
```

With `gfan_tropicalintersection` *one can verify that the intersection of the tropical hypersurfaces defined by the output polynomials is actually smaller than the intersection of the hypersurfaces defined by the input polynomials.*

A theorem by Bieri and Groves [2] says that the tropical variety of a d-dimensional prime ideal is *pure of dimension* d, which means that all maximal cones have dimension d. Furthermore, Speyer's theorem [3, Theorem 14] says that this pure d-dimensional complex is connected in codimension one. Being connected in codimension 1 means that any two d-dimensional cones in the complex can be connected by a sequence of d-dimensional cones with any two consecutive cones intersecting in a $(d-1)$-dimensional ridge.

These theorems form the basis for Gfan's methods for computing tropical varieties of prime ideals. The following example shows how to combine the programs `gfan_tropicalstartingcone` and `gfan_tropicaltraverse` to compute the tropical variety of a homogeneous prime ideal.

EXAMPLE 7. *Let $a, b, \ldots, j$ be the 10 2x2 minors of the 2x5 matrix*

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \end{pmatrix}.$$

For a suitable ordering they satisfy the Grassmann-Plücker relations:

$$0 = bf - ah - ce = bg - ai - de = cg - aj - df = ci - bj - dh = fi - ej - gh.$$

The ideal generated by these relations is called the Grassmann-Plücker ideal $I \subseteq \mathbb{Q}[a, \ldots, j]$. The ideal is prime and has dimension 7. Hence the tropical variety of I is pure of dimension 7. To exploit the connectivity of the tropical variety we first need to compute a single 7-dimensional cone of $\mathcal{T}(I)$. In the worst case this may be as difficult as computing $\mathcal{T}(I)$ itself. The program `gfan_tropicalstartingcone` *has a heuristic method for guessing a starting cone. The idea is to build up the cone one dimension at a time as explained in [3, Algorithm 9]. In each step a ray in the tropical variety outside the homogeneity space is needed. To get this ray Gfan computes extreme rays of the Gröbner cone of the input Gröbner basis and checks if the rays are in the tropical variety. The heuristics work reasonable well but may fail. In that case it is usually worthwhile to try a different input Gröbner basis for the program. For our example the program produces the following output:*

```
{f*i-e*j,
d*h-c*i,
d*f+a*j,
d*e+a*i,
c*e+a*h}
{f*i-g*h-e*j,
d*h-c*i+b*j,
d*f-c*g+a*j,
d*e-b*g+a*i,
c*e-b*f+a*h}
```

This is a pair of reduced Gröbner bases that Gfan uses for representing the d-dimensional starting Gröbner cone. For n-dimensional Gröbner cones only a single Gröbner basis is needed as we saw in Section 2. For lower-dimensional cones the pair consists of a Gröbner basis for the initial ideal $\mathrm{in}_\omega(I)$ and a Gröbner basis for I itself with respect to a term order $\prec_\omega$ where ω is in the relative interior of the Gröbner cone to be represented.

Having computed the starting cone we may run `gfan_tropicaltraverse` *which traverses the 7-dimensional cones and produces an output in a format similar to the one in Example 5. The*

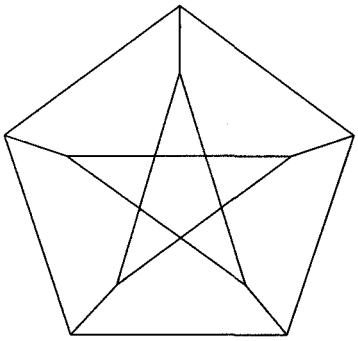


FIGURE 4. *The tropical variety of the ideal in Example 7 drawn projectively.*

homogeneity space of I has dimension 5 *and all cones in the tropical variety contains this subspace. Besides the homogeneity space the tropical variety consists of 10 6-dimensional cones and 15 7-dimensional cones. Modulo the homogeneity space the tropical variety is* 2*-dimensional. Projectively we may draw its combinatorics as the Petersen graph; see Figure 4. The homogeneity space is the center of the projection.*

If the ideal is equi-dimensional, but not prime it may be that the tropical variety is not connected in codimension 1. In that case `gfan_tropicaltraverse` computes a connected component. Just as it is the case for Gröbner fans Gfan is able to enumerate the cones of a tropical variety up to symmetry.

For `gfan_tropicaltraverse` to compute the tropical variety of an ideal the ideal must be homogeneous and defined over the rationals. However most tropical papers consider the case where I is defined in the polynomial ring over the Puiseux series field $\mathbb{C}\{\{t\}\}$. In this case the valuation of the coefficients is taken into account when defining initial forms and the tropical variety is a polyhedral complex which need not consist of cones. Thus, a priori, Gfan seems too restrictive. Notice however,

- the tropical variety of the homogenization of an ideal I equals, as a set, the tropical variety of I except that it lives in a vector space of dimension one higher; see [3, Lemma 4].
- the tropical variety of an ideal $I \subseteq \mathbb{C}\{\{t\}\}[x_1, \ldots, x_n]$ generated by elements in the polynomial ring over the rational functions in t, $\mathbb{C}(t)[x_1, \ldots, x_n]$, may be computed by considering the ideal $I \cap \mathbb{C}[t, x_1, \ldots, x_n]$ in $\mathbb{C}[t, x_1, \ldots, x_n]$ and intersecting its tropical variety with the $t = 1$ hyperplane, see [3, Lemma 1].
- if an ideal I is generated by elements in $K[x_1, \ldots, x_n]$ where K is an algebraic field extension of $\mathbb{Q}$ with an algebraic root α then $\mathcal{T}(I)$ can be computed as the tropical variety of an ideal in $\mathbb{Q}[\alpha, x_1, \ldots, x_n]$ where the minimal polynomial of α has been added as a generator, see [12, Lemma 3.12].

In addition Gfan can do computations in the polynomial ring $\mathbb{Z}/p\mathbb{Z}[x_1, \ldots, x_n]$ where p is a prime. Hence, Gfan has the theoretic ability to compute the tropical variety of most ideals we encounter in practice.

4. Algorithms. We briefly summarize the most important algorithms implemented in Gfan. For the Gröbner fan part we have the following algorithms:

- a simple implementation of Buchberger's algorithm,
- an algorithm for finding facets of maximal Gröbner cones,
- an implementation of the local basis change procedure of [4], and
- two algorithms for doing global enumerations of the maximal cones in the Gröbner fan.

Finding facets of Gröbner cones amounts to solving linear programming problems. For this Gfan uses cddlib [5] which does its computations in exact arithmetic.

The local basis change algorithm takes as input a Gröbner basis $\mathcal{G}$ and a facet F of its Gröbner cone. The output is the Gröbner basis of the cone on the other side of F. The algorithm consists of two steps. Step 1 is to compute a certain Gröbner basis of the initial ideal $\mathrm{in}_\omega(I)$ where ω is in the relative interior of F. This Gröbner basis computation turns out to be relatively cheap. The reason for this is that $\mathrm{in}_\omega(I)$ is homogeneous with respect to any grading given by a vector in F. In fact, the Newton polytopes of the polynomials that need to be manipulated are all parallel line segments. The second step is to lift the Gröbner basis of $\mathrm{in}_\omega(I)$ to the right Gröbner basis of I. In Gfan turning the lifted basis into a reduced basis is often the most time consuming part of the whole local basis change process.

For the global enumeration of the Gröbner fan two different approaches are implemented. The straight forward approach is to make an exhaustive enumeration of all maximal Gröbner cones. This method has the advantage that it is easy to exploit symmetry. The other approach is to apply the reverse search technique [1]. The advantage is that reverse search is memory-less in the sense that it does not have to store the set of already computed cones. Saving memory is important if Gröbner fans with millions of cones need to be enumerated. In [6] a reverse search rule which works even in the non-homogeneous case was presented. Unfortunately, it seems that reverse search is not compatible with enumeration up to symmetry. The algorithms for computing Gröbner fans are described in detail in [6].

The tropical part of Gfan has implementations of the following algorithms

- algorithms for handling polyhedral cones and turning them into unique form,
- an algorithm for constructing the codimension one skeleton of the normal fan of a polytope,

- an algorithm for computing common refinements (intersections) of tropical varieties,
- an algorithm for computing tropical bases of curves, and
- an algorithm for traversing a connected component of a tropical variety.

Again, the basic polyhedral computations are handled by cddlib. Computing refinements of tropical hypersurfaces is equivalent to computing the mixed faces of a Minkowski sum. Computation of mixed faces is a well studied problem as it can be used for mixed volume computations and for the first step in the polyhedral homotopy method for solving polynomial system numerically. In these cases one is usually only interested in the mixed facets and has the option of applying a generic lifting to make things generic. However, for Gfan the exact combinatorics are needed. This means that Gfan computes mixed faces of every dimension.

The algorithm for computing tropical bases of curves was described in [3, Algorithm 5]. It relies on the intersection of tropical hypersurfaces and a Gröbner basis computation with respect to an order that depends on the polyhedral computations. This Gröbner basis step can be difficult and thereby it contrasts the easy Gröbner basis computations needed in the traversal of the Gröbner fan.

The global traversal is an exhaustive search where each local step amounts to a tropical curve computation; see [3, Algorithm 7]. Here the enumeration can again be performed up to symmetry. We should mention that in Gfan symmetry is never exploited in Gröbner basis computations but only in the global enumeration of fans.

5. A few benchmarks. While Gröbner basis computations are doubly exponential in the worst case the Buchberger step of the local Gröbner basis change procedure is relatively easy in practice due to the homogeneity and other properties of the ideals in question. However, sometimes the Gröbner fan is simply too big to be computed or the conversion steps take too much time. Here is a few examples of what can be computed.

In Figure 2 the 360 maximal cones were computed in 58 seconds using 5 megabytes of memory. It is easy to find four variable ideals where the Gröbner fan is too big to be traversed or the local steps are too time consuming. For some classes of ideals the Gröbner fan enumeration algorithm can handle much larger examples.

EXAMPLE 8. *Let I be the ideal generated by the 3x3 minors of the 4x4 matrix*

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{pmatrix}$$

in the polynomial ring of 16 variables.

The Gröbner fan consists of 163032 full-dimensional cones. The group $S_4 \times S_4$ of order 576 acts on the polynomial ring by permuting rows and columns of the matrix. Under this action I is fixed as an ideal. The 163032 marked reduced Gröbner bases can be computed up to symmetry in 7 minutes using 9 megabytes of memory. There are only 289 orbits to consider. Without exploiting symmetry the computation takes 14 hours.

It is not only the Gröbner fan computation which has bad complexity. In [22] several decision problems related to tropical varieties are shown to be NP-hard. The tropical variety $\mathcal{T}(I)$ in the example above is a 12-dimensional sub-complex of the Gröbner fan with 936 maximal cones. Its f-vector is $(1, 50, 360, 1128, 1680, 936)$, meaning that there is one cone of dimension 7 which is the homogeneity space. Traversing the maximal cones of $\mathcal{T}(I)$ while exploiting symmetry takes 2 minutes.

EXAMPLE 9. *Consider the 20 3x3 minors of a 3x6 matrix*

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & x_{26} \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & x_{36} \end{pmatrix}.$$

The relations on these generate the Grassmann-Plücker ideal $I_{3,6}$ which is an ideal in a polynomial ring with 20 variables. Its tropical variety is pure of dimension 10 and has a 6-dimensional homogeneity space. It consists of 1035 maximal cones which were computed in 3-4 hours exploiting symmetry. The cones come in 102 orbits.

6. Interfacing other programs and future plans. Interfacing different pieces of mathematical software is a non-trivial task. For Gfan one difficulty is that on one hand it is a software system for commutative algebra and on the other hand a software system for polyhedral geometry. Thus no single interface suffices.

Thanks to the developers of Macaulay 2 [9] and SAGE [18] it is now possible to invoke some basic Gfan functions from these programs. For SAGE one additional advantage of this is that Gfan is part of the SAGE distribution and therefore does not need a separate installation. Due to the command line interface of Gfan it is possible, but cumbersome, to invoke it from other mathematical software such as Singular [10] through a `system()` call. This approach was chosen in the Singular implementation of the main algorithm in [12].

For the polyhedral part the upcoming Gfan version 0.3 will output the computed tropical varieties in a Polymake [7] compatible format. Since Polymake does not handle polyhedral fans in its current version this has limited use at the moment. However, it is our hope that the Polymake format will serve as a standard format to be used by other tropical variety software such as TrIm [20].

Internally Gfan is linked to GMP [8] and cddlib [5] for exact arithmetic and polyhedral computations. As pointed out in the discussion about performance for tropical varieties the Gröbner basis computation is one weak

point of Gfan. One solution is to link Gfan to a commutative algebra software library. CoCoALib [21] is currently under consideration.

REFERENCES

[1] D. Avis and K. Fukuda, *A basis enumeration algorithm for convex hulls and vertex enumeration of arrangements and polyhedra.*, Discrete Computational Geometry, **8** (1992), pp. 295–313.

[2] R. Bieri and J. Groves, *The geometry of the set of characters induced by valuations*, J. reine und angewandte Mathematik, **347** (1984), pp. 168–195.

[3] T. Bogart, A. Jensen, R. Thomas, D. Speyer, and B. Sturmfels, *Computing tropical varieties.*, J. Symb. Comput., **42** (2007), pp. 54–73.

[4] S. Collart, M. Kalkbrener, and D. Mall, *Converting bases with the Gröbner walk.*, J. Symb. Comput., **24** (1997), pp. 465–469.

[5] K. Fukuda, *cddlib reference manual, cddlib Version 094b*, Swiss Federal Institute of Technology, Lausanne and Zürich, Switzerland, 2005. (`http://www.ifor.math.ethz.ch/~fukuda/cdd_home/cdd.html`.)

[6] K. Fukuda, A. Jensen, and R. Thomas, *Computing Gröbner fans.*, Mathematics of Computation, **76** (2007), pp. 2189–2212.

[7] E. Gawrilow and M. Joswig, *polymake: a framework for analyzing convex polytopes*, in Polytopes — Combinatorics and Computation, G. Kalai and G.M. Ziegler, eds., Birkhäuser, 2000, pp. 43–74.

[8] T. Granlund and et al., *GNU multiple precision arithmetic library 4.1.2*, December 2002. (`http://swox.com/gmp/`.)

[9] D.R. Grayson and M.E. Stillman, *Macaulay 2, a software system for research in algebraic geometry*, 2007. (`http://www.math.uiuc.edu/Macaulay2/`.)

[10] G.-M. Greuel, G. Pfister, and H. Schönemann, Singular *2.0.5*, A Computer Algebra System for Polynomial Computations, Centre for Computer Algebra, University of Kaiserslautern, 2004. (`http://www.singular.uni-kl.de`.)

[11] B. Huber and R.R. Thomas, *Computing Gröbner fans of toric ideals.*, Experimental Mathematics, **9** (2000), pp. 321–331.

[12] A. Jensen, H. Markwig, and T. Markwig, *An algorithm for lifting points in a tropical variety*, 2007. Preprint, math.AG/0705.2441.

[13] A.N. Jensen, *Gfan, a software system for Gröbner fans*, 2006. (http://www.math.tu-berlin.de/~jensen/software/gfan/gfan.html.)

[14] A.N. Jensen, *A presentation of the Gfan software.*, in Mathematical Software - ICMS 2006, Iglesias and Takayama, eds., Springer, 2006, pp. 222–224.

[15] T. Mora and L. Robbiano, *The Gröbner fan of an ideal.*, J. Symb. Comput., **6** (1988), pp. 183–208.

[16] J. Rambau, *TOPCOM: Triangulations of point configurations and oriented matroids*, ZIB report, 02-17 (2002).

[17] D. Speyer and B. Sturmfels, *The tropical Grassmannian*, Adv. Geom., **4** (2004), pp. 389–411.

[18] W. Stein, *Sage: Software for algebra and geometry experimentation.*, 2007. (`http://www.sagemath.org/,http://sage.scipy.org/`.)

[19] B. Sturmfels, *Gröbner bases and Convex Polytopes*, Vol. **8** of University Lecture Series, American Mathematical Society, 1996.

[20] B. Sturmfels and J. Yu, *Tropical implicitization and mixed fiber polytopes*, IMA Volume **148**: Software for Algebraic Geometry; Editors: Michael E. Stillman, Nobuki Takayama, and Jan Verschelde; Publisher: Springer Science+Business Media, 2008.

[21] The CoCoA Team, *The CoCoA Project*, 2007. (`http://cocoa.dima.unige.it/`.)

[22] T. Theobald, *On the frontiers of polynomial computations in tropical geometry*, J. Symbolic Comput., **41** (2006), pp. 1360–1375.

ON A CONJECTURE FOR THE DIMENSION OF THE SPACE OF THE MULTIPLE ZETA VALUES

MASANOBU KANEKO*, MASAYUKI NORO†, AND KEN'ICHI TSURUMAKI‡

Abstract. Since Euler, values of various zeta functions have long attracted a lot of mathematicians. In computer algebra community, Apéry's proof of the irrationality of $\zeta(3)$ is well known. In this paper, we are concerned with the "multiple zeta value (MZV)". More than fifteen years ago, D. Zagier gave a conjecture on MZVs based on numerical computations on PARI. Since then there have been various derived conjectures and two kinds of efforts for attacking them: one is a mathematical proof and another one is a computational experiment to get more confidence to verify a conjecture. We have checked one of these conjectures up to weight $k = 20$, which will be explained later, with Risa/Asir function for non-commutative polynomials and special parallel programs of linear algebra designed for this purpose.

Key words. Multiple zeta value, double shuffle relation, symbolic computation, parallel computation.

AMS(MOS) subject classifications. Primary 14G10, 11M06, 11Y99, 68W30.

1. Introduction. The multiple zeta value (MZV) is a real number defined by the convergent series

$$\zeta(\mathbf{k}) = \zeta(k_1, k_2, \ldots, k_n) = \sum_{m_1 > m_2 > \cdots > m_n > 0} \frac{1}{m_1^{k_1} m_2^{k_2} \cdots m_n^{k_n}}, \qquad (1.1)$$

where $\mathbf{k} = (k_1, k_2, \ldots, k_n)$ is an index set of positive integers with $k_1 > 1$ (which ensures the convergence). In recent years, the MZVs have been appeared in various areas of mathematics and physics and aroused stimulating interest among researchers. In particular, it has become apparent that the structures of (linear or algebraic) relations over the rationals $\mathbf{Q}$ among MZVs reflect properties or structures of various, seemingly unrelated mathematical objects. We refer the readers to Zagier's pioneer work [13] and [2], [12], [8] together with the references therein for more on the subjeet. In the present paper, we discuss experiments concerning a certain conjecture on the linear relations among MZVs.

For each integer $k \geq 2$, let $\mathcal{Z}_k$ be the $\mathbf{Q}$-vector space spanned by the MZVs $\zeta(k_1, \ldots, k_n)$ whose weight $= k_1 + \cdots + k_n$ is equal to k. In [13], Don Zagier gave a remarkable conjectural formula for $\dim_{\mathbf{Q}} \mathcal{Z}_k$:

$$\dim_{\mathbf{Q}} \mathcal{Z}_k \stackrel{?}{=} d_k,$$

*Graduate School of Mathematics, Kyushu University, 33, Fukuoka 812-8581, Japan.

†Department of Mathematics, Graduate School of Science, Kobe University, 1-1, Rokkodai, Nada-ku, Kobe 657-8501, Japan.

‡Oracle Corporation, Japan.

where the number d_k is determined by the Fibonacci-like recurrence

$$d_2 = d_3 = d_4 = 1, \quad d_k = d_{k-2} + d_{k-3} \quad (k \geq 5).$$

The total number of index sets of weight k is 2^{k-2}, which is much bigger than $d_k \approx 0.4115\cdots \times (1.3247\cdots)^k$. For instance, $2^{20-2} = 262144$ whereas $d_{20} = 114$. Hence we expect many linear relations among MZVs of given weight (note that any relations known so far are the relations among MZVs of *same* weight). One of the recent major progress in the theory is that Goncharov [5] and Terasoma [11] independently proved that the number d_k gives an upper bound of $\dim_{\mathbf{Q}} \mathcal{Z}_k$:

THEOREM 1.1 ([5],[11]). *The inequality* $\dim_{\mathbf{Q}} \mathcal{Z}_k \leq d_k$ *holds for all* k.

However, their proofs (both relying on the theory of mixed Tate motif) do not give any explicit set of linear relations to reduce the number of generators to the upper bound, and the question as to what sorts of relations are needed for that is still unanswered. Concerning to this question, there is a conjecture in which we are mainly interested:

CONJECTURE 1.1 ([7]). *The extended double shuffle relations suffice to give all linear relations among MZVs.*

A stronger version of this conjecture was proposed by H.N. Minh, M. Petitot et al. in [9] and they verified it (in the sense that their proposed relations suffice to reduce $\dim_{\mathbf{Q}} \mathcal{Z}_k$ to d_k) up to weight 16 (private communication). Also, Espie, Novelli and Racinet [3] verified the conjecture up to weight 19 (but modulo powers of π^2 at even weights), in a different context of certain Lie algebra closely related to the "Drinfel'd associator". Our main objective is to verify (a still stronger version of) this conjecture up to weight $k = 20$. In the next section we explain what this conjecture exactly means and introduce an algebraic setup to study the conjecture. Using this setup, we can implement tools for generating relations among MZVs systematically. This will be carried out in Section 3. In Section 4, we will verify the conjecture up to $k = 20$ by using the algebraic tools and special parallel programs of linear algebra. This work follows an experimental computation by Minh and Petitot. We also give a new conjecture in Section 4, which is a refined version of Conjecture 1.1.

2. The algebraic formulation. The MZV can be given not only as a sum (1.1) but also as an integral

$$\zeta(k_1, k_2, \ldots, k_n) = \int\cdots\int_{1>t_1>t_2>\cdots>t_k>0} \omega_1(t_1)\omega_2(t_2)\cdots\omega_k(t_k), \tag{2.1}$$

where $k = k_1 + k_2 + \cdots + k_n$ is the weight and $\omega_i(t) = dt/(1-t)$ if $i \in \{k_1, k_1+k_2, \ldots, k_1+k_2+\cdots+k_n\}$ and $\omega_i(t) = dt/t$ otherwise. From each of these representations one finds that the product of two MZVs is a $\mathbf{Z}$-linear combination of MZVs. The first example is

$$\begin{aligned}\zeta(2)^2 &= \left(\sum_{m>0}\frac{1}{m^2}\right)\left(\sum_{n>0}\frac{1}{n^2}\right) = \sum_{m,n>0}\frac{1}{m^2n^2}\\ &= \left(\sum_{m>n>0}+\sum_{n>m>0}+\sum_{m=n>0}\right)\frac{1}{m^2n^2}\\ &= 2\zeta(2,2)+\zeta(4)\end{aligned}$$

and

$$\begin{aligned}\zeta(2)^2 &= \left(\iint_{1>t_1>t_2>0}\frac{dt_1dt_2}{t_1(1-t_2)}\right)\left(\iint_{1>t_1'>t_2'>0}\frac{dt_1'dt_2'}{t_1'(1-t_2')}\right)\\ &= \iiiint_{\substack{1>t_1>t_2>0\\1>t_1'>t_2'>0}}\frac{dt_1dt_2dt_1'dt_2'}{t_1(1-t_2)t_1'(1-t_2')}\\ &= 4\iiiint_{1>s_1>s_2>s_3>s_4>0}\frac{ds_1ds_2ds_3ds_4}{s_1s_2(1-s_3)(1-s_4)}\\ &\quad +2\iiiint_{1>s_1>s_2>s_3>s_4>0}\frac{ds_1ds_2ds_3ds_4}{s_1(1-s_2)s_3(1-s_4)}\\ &= 4\zeta(3,1)+2\zeta(2,2),\end{aligned}$$

and this gives $\zeta(4)=4\zeta(3,1)$. The point here is that the two expressions obtained are always different, and thus their equality gives a collection of linear relations among MZVs which we call the *finite double shuffle relations* (FDS). Moreover, one can extend the finite double shuffle relations by taking divergent sums and integrals into account together with a certain regularization procedure. We call these generalized relations the *extended double shuffle relations* (EDS), and the conjecture is that the EDS suffices to give all linear relations among MZVs (Conjecture 1.1).

The two multiplication rules mentioned above are described in a purely algebraic manner, as given in Hoffman [6]. Let $\mathfrak{H}=\mathbf{Q}\langle x,y\rangle$ be the non-commutative polynomial algebra over the rationals in two indeterminates x and y, and $\mathfrak{H}^1$ and $\mathfrak{H}^0$ its subalgebras $\mathbf{Q}+\mathfrak{H}y$ and $\mathbf{Q}+x\mathfrak{H}y$, respectively. Let $Z:\mathfrak{H}^0\to\mathbf{R}$ be the $\mathbf{Q}$-linear map ("evaluation map") which assigns to each word (monomial) $u_1u_2\cdots u_k$ $(u_i\in\{x,y\})$ in $\mathfrak{H}^0$ the multiple integral

$$\int_{1>t_1>t_2>\cdots>t_k>0}\cdots\int\omega_{u_1}(t_1)\omega_{u_2}(t_2)\cdots\omega_{u_k}(t_k)$$

where $\omega_x(t)=dt/t$, $\omega_y(t)=dt/(1-t)$. We set $Z(1)=1$. Since the word $u_1u_2\cdots u_k$ is in $\mathfrak{H}^0$, we always have $\omega_{u_1}(t)=dt/t$ and $\omega_{u_k}(t)=dt/(1-t)$, so the integral converges. By the integral representation (2.1), we have

$$Z(x^{k_1-1}yx^{k_2-1}y\cdots x^{k_n-1}y)=\zeta(k_1,k_2,\ldots,k_n).$$

The weight $k = k_1 + k_2 + \cdots + k_n$ of $\zeta(k_1, k_2, \ldots, k_n)$ is the total degree of the corresponding monomial $x^{k_1-1}yx^{k_2-1}y \cdots x^{k_n-1}y$.

Let $z_k := x^{k-1}y$, which corresponds under Z to the Riemann zeta value $\zeta(k)$. Then $\mathfrak{H}^1$ is freely generated by z_k $(k = 1, 2, 3, \ldots)$. We define the *harmonic product* $*$ on $\mathfrak{H}^1$ inductively by

$$1 * w = w * 1 = w,\ z_k w_1 * z_l w_2 = z_k(w_1 * z_l w_2) + z_l(z_k w_1 * w_2) + z_{k+l}(w_1 * w_2),$$

for all $k, l \geq 1$ and any words $w, w_1, w_2 \in \mathfrak{H}^1$, and then extending by $\mathbf{Q}$-bilinearity. Equipped with this product, $\mathfrak{H}^1$ becomes a commutative algebra ([6]) and $\mathfrak{H}^0$ a subalgebra. The first multiplication law of MZVs asserts that the evaluation map $Z : \mathfrak{H}^0 \to \mathbf{R}$ is an algebra homomorphism with respect to the multiplication $*$, i.e.,

$$Z(w_1 * w_2) = Z(w_1)Z(w_2) \quad (w_1, w_2 \in \mathfrak{H}^0). \tag{2.2}$$

For instance, the harmonic product $z_k * z_l = z_k z_l + z_l z_k + z_{k+l}$ corresponds to the identity $\zeta(k)\zeta(l) = \zeta(k, l) + \zeta(l, k) + \zeta(k + l)$.

The other commutative product ш, referred to as the *shuffle product*, corresponding to the product of two integrals in (2.1), is defined on all of $\mathfrak{H}$ inductively by setting

$$1 \text{ш} w = w \text{ш} 1 = w,\ uw_1 \text{ш} vw_2 = u(w_1 \text{ш} vw_2) + v(uw_1 \text{ш} w_2),$$

for any words $w, w_1, w_2 \in \mathfrak{H}$ and $u, v \in \{x, y\}$, and again extending by $\mathbf{Q}$-bilinearity. The character 'ш' is the Cyrillic *sha*, standing for *shuffle*. This product gives $\mathfrak{H}$ the structure of a commutative $\mathbf{Q}$-algebra ([10]) which we denote by $\mathfrak{H}_{\text{ш}}$. Obviously the subspaces $\mathfrak{H}^1$ and $\mathfrak{H}^0$ become subalgebras of $\mathfrak{H}_{\text{ш}}$, denoted by $\mathfrak{H}^1_{\text{ш}}$ and $\mathfrak{H}^0_{\text{ш}}$ respectively. By the standard shuffle product identity of iterated integrals, the evaluation map Z is again an algebra homomorphism for the multiplication ш:

$$Z(w_1 \text{ш} w_2) = Z(w_1)Z(w_2) \quad (w_1, w_2 \in \mathfrak{H}^0). \tag{2.3}$$

By equating (2.2) and (2.3), we get the *finite double shuffle relations* (FDS) of MZV:

$$Z(w_1 \text{ш} w_2 - w_1 * w_2) = 0 \quad (w_1, w_2 \in \mathfrak{H}^0). \tag{2.4}$$

These relations, however, do not suffice to obtain all relations. For instance, there are two MZVs of weight 3, $\zeta(3)$ and $\zeta(2, 1)$. As Euler already showed in [4], these two values are actually equal, and therefore $\dim_{\mathbf{Q}} \mathcal{Z}_3 = 1$. But the least weight of FDS is 4 so that we cannot deduce the relation $\zeta(3) = \zeta(2, 1)$ by the FDS. In order to obtain more relations, we introduce a "regularization" procedure which amounts to incorporate the divergent MZVs into the picture. In the following we only give a minimum of what we need to formulate the extended double shuffle relations, and the interested readers are invited to consult the paper [7].

It is known (cf. [10]) that the commutative algebra $\mathfrak{H}^1_{ш}$ is isomorphic to the polynomial algebra over $\mathfrak{H}^0_{ш}$ in y:

$$\mathfrak{H}^1_{ш} \simeq \mathfrak{H}^0_{ш}[y]. \tag{2.5}$$

Let $\mathrm{reg}_{ш}$ ("regularization map") be the map from $\mathfrak{H}^1_{ш}$ to $\mathfrak{H}^0_{ш}$ obtained by taking the "constant term" with respect to y under the isomorphism (2.5):

$$\mathrm{reg}_{ш} : \mathfrak{H}^1_{ш} \ni w = \sum_{i=0}^{n} w_i ш y^{ш i} \mapsto w_0 \in \mathfrak{H}^0_{ш}.$$

Note that the map $\mathrm{reg}_{ш}$ is the identity if restricted to the subspace $\mathfrak{H}^0_{ш}$. We then have the following theorem. For a proof, we refer the reader to [7].

THEOREM 2.1 (extended double shuffle relations (EDS), [7]). *For any $w_1 \in \mathfrak{H}^1$ and $w_0 \in \mathfrak{H}^0$, we have*

$$Z\big(\mathrm{reg}_{ш}(w_1 ш w_0 - w_1 * w_0)\big) = 0.$$

When $w_1 \in \mathfrak{H}^0$, the above relation reduces to the finite double shuffle relation and so the EDS contains the FDS. If we take $w_1 = y$ as a particular case, it can be shown that the element $y ш w_0 - y * w_0$ is always in $\mathfrak{H}^0$ and hence we obtain the relation (without the regularization)

$$Z(y ш w_0 - y * w_0) = 0 \quad (w_0 \in \mathfrak{H}^0). \tag{2.6}$$

If we substitute $x^{k_1-1}yx^{k_2-1}y\cdots x^{k_n-1}y$ for w_0 in this relation and expand it by using the definitions of ш and $*$, we readily obtain the relation known as Hoffman's relation:

$$\begin{aligned}\sum_{i=1}^{n} &\zeta(k_1,\ldots,k_{i-1},k_i+1,k_{i+1},\ldots,k_n)\\ &= \sum_{1\le l\le n, k_l\ge 2}\ \sum_{j=0}^{k_l-2} \zeta(k_1,\ldots,k_{l-1},k_l-j,j+1,k_{l+1},\ldots,k_n).\end{aligned} \tag{2.7}$$

3. Implementation of the ring of bivariate non-commutative polynomials.

3.1. Prototyping. The algebra $\mathbf{Q}\langle x,y\rangle$ is a non-commutative polynomial ring. There are several computer algebra systems supporting non-commutative polynomial ring and we have tried some of them. For example Mathematica can calculate non-commutative polynomials, however it did not behave as we hoped and it was not efficient. Next we tried representing a non-commutative monomial as a list and wrote a short program to implement it. But it was not efficient because a list representation needs many links by pointers and even a simple monomial operation requires a considerable number of memory operations. Then we tried representing a monomial by a character string. It may seem more naive than the list

representation, but in fact it can be efficiently realized because the product of two monomials can be computed by the concatenation of two character strings, and the comparison can be done by comparing the character strings according to a specific ordering, say, the lexicographic ordering. By using this method, we implemented the operations in $\mathbf{Q}\langle x, y\rangle$ by the user language of Risa/Asir. The program amounts to only 160 lines.

3.2. Implementation as a built-in data type. Based on the prototyping, we decided to implement $\mathbf{Q}\langle x, y\rangle$ as a built-in data type in Risa/Asir for our large scale experiments. In this implementation, a monomial is represented as a bit sequence. All the fundamental operations in $\mathbf{Q}\langle x, y\rangle$, including the shuffle and harmonic products are implemented as built-in functions written in C. Preliminary experiments show that they are more efficient than the prototype, nevertheless the speed-up is about a factor of 10 and the prototype by general facilities is proved to be efficient enough for small experiments.

An element in $\mathbf{Q}\langle x, y\rangle$ is converted from a `QUOTE` object, which is also a built-in data type for expressing general non-commutative objects. A `QUOTE` object holds a tree structure converted from an input expression. It preserves the order of products and we can convert it to an element in $\mathbf{Q}\langle x, y\rangle$ by calling a built-in function `qt_to_nbp()`. After computing polynomials giving EDS relations of a fixed weight k, we convert them to integer vectors via the monomial basis consisting of all weight k monomials in $\mathfrak{H}^0$. In order to make the conversion efficient, we provide a function to compute the index of a monomial in the monomial basis. Note that the index can be computed easily if we apply the lexicographic ordering. `http://www.math.kobe-u.ac.jp/OpenXM/Math/MZV/mzv_dsr.rr` is a simple Risa/Asir program to compute a set of generators of $\mathcal{Z}_k$ and to represent remaining MZVs of weight k as linear combinations of the generators. The function `dsr_matrix(k)` returns a matrix constructed from a specific subset of EDS relations of weight k, which will be explained in Section 4.1. Each row of the matrix gives the coefficients of MZVs in an EDS relation, where MZVs are indexed according to the lexicographic ordering. FIG. 1 is the output of `dsr_matrix(7)`. For example, the first row in FIG. 1 shows a relation

$$\zeta(7) - \zeta(6,1) - \zeta(5,2) - \zeta(4,3) - \zeta(3,4) - \zeta(2,5) = 0.$$

The function `dsr_basis(k)` returns a list $[[M_1, P_1], \ldots, [M_i, P_i], \ldots]$. In this output M_i and P_i are given as elements of $\mathbf{Q}\langle x, y\rangle$. Each pair $[M_i, P_i]$ means $Z(M_i) = Z(P_i)$, which represents an MZV $Z(M_i)$ as a linear combination of the generators of $\mathcal{Z}_k$ computed from the input matrix given by `dsr_matrix(k)`. The function `nbp_to_mzv(F)` converts a polynomial F in $\mathbf{Q}\langle x, y\rangle$ to a linear combination of MZVs. In the output of this function, an MZV is given as a list $[k_1, k_2, \ldots]$ which represents $\zeta(k_1, k_2, \ldots)$.

$$\left(\begin{array}{cccccccccccccccccccccccccccccc}
1&-1&-1&0&-1&0&0&0&-1&0&0&0&0&0&0&0&-1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&1&-1&0&-1&0&0&0&-1&0&0&0&0&0&0&0&-1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&1&-1&-1&0&0&0&-1&0&0&0&0&0&0&0&-1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&1&1&-1&0&0&0&-1&0&0&0&0&0&0&0&-1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&1&-1&-1&0&-1&0&0&0&0&0&0&0&-1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&1&1&-1&0&-1&0&0&0&0&0&0&0&-1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&1&0&1&-1&-1&0&0&0&0&0&0&0&-1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&1&1&-1&0&0&0&0&0&0&0&-1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&-1&-1&0&-1&0&0&0&-1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&1&-1&0&-1&0&0&0&-1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&1&1&-1&0&0&0&-1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&1&1&-1&0&0&0&-1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&1&-1&-1&0&-1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&1&1&-1&0&-1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&1&0&1&1\\
\hline
-1&10&4&0&3&0&0&0&2&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&-1&0&12&-1&4&0&0&0&2&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&-1&0&0&6&6&0&-1&4&2&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&-1&0&0&0&12&0&-1&0&4&-1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&-1&0&0&0&0&4&2&0&4&0&0&0&-1&6&2&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&-1&0&0&0&0&6&0&4&0&0&0&-1&0&6&-1&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&-1&0&0&0&0&0&0&4&6&0&0&0&-1&0&0&2&1&0&-1&4&0&0&0&0&0\\
0&0&0&0&0&0&0&-1&0&0&0&0&0&0&10&0&0&0&-1&0&0&0&3&0&-1&0&2&-1&1&0&0\\
-1&20&10&0&3&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&-1&0&18&0&9&3&0&-1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&-1&0&0&6&6&0&0&8&4&0&0&0&0&0&-1&6&2&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&-1&0&0&0&12&0&0&0&6&0&4&2&0&0&-1&0&2&0&2&1&0&-1&1&1&0&0&0&0\\
0&-1&0&12&0&6&3&0&0&4&2&0&2&0&0&0&-1&1&1&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&-1&-2&0&0&0&18&0&-1&-1&7&0&2&0&0&0&-1&-1&1&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&-1&-1&-1&0&0&0&0&6&0&8&6&0&0&-1&-1&6&-2&2&1&0&0&0&0&0&0&0\\
0&0&0&0&0&-1&-1&-3&0&0&0&0&0&0&0&20&0&0&-1&-3&0&-2&-2&4&0&-1&-1&1&0&0&0
\end{array}\right)$$

FIG. 1. *The output of* `dsr_matrix(7)`.

The following example shows that $\mathcal{Z}_6$ is generated by

$$\{Z(xyyyxy), Z(xyyyyy)\} = \{\zeta(2,1,1,2), \zeta(2,1,1,1,1)\}.$$

For example, the first element of the output means

$$\zeta(2,1,2,1) = -\frac{1}{2}\zeta(2,1,1,2) + \frac{13}{24}\zeta(2,1,1,1,1).$$

Note that the number of generators coincides with $d_6 = 2$.

```
[0] load("./mzv_dsr.rr")$
[7] map(print,dsr_basis(6))$
[(1)*xyyxyy,(-1/2)*xyyyxy+(13/24)*xyyyyy]
[(1)*xyyxxy,(-1)*xyyyxy+(61/48)*xyyyyy]
[(1)*xyxyyy,(-1)*xyyyxy+(3/4)*xyyyyy]
[(1)*xyxyxy,(3/16)*xyyyyy]
[(1)*xyxxyy,(3/2)*xyyyxy+(-11/12)*xyyyyy]
[(1)*xyxxxy,(1)*xyyyxy]
[(1)*xxyyyy,(1/2)*xyyyxy+(-7/24)*xyyyyy]
[(1)*xxyyxy,(3/2)*xyyyxy+(-11/12)*xyyyyy]
[(1)*xxyxyy,(-3)*xyyyxy+(97/48)*xyyyyy]
[(1)*xxyxxy,(-1/2)*xyyyxy+(13/24)*xyyyyy]
[(1)*xxxyyy,(1)*xyyyxy+(-31/48)*xyyyyy]
[(1)*xxxyxy,(-1)*xyyyxy+(3/4)*xyyyyy]
[(1)*xxxxyy,(1/2)*xyyyxy+(-7/24)*xyyyyy]
[(1)*xxxxxy,(1)*xyyyyy]
```

4. Experimental results and a new conjecture. In this section, we numerically verify Conjecture 1.1 up to $k = 20$, which seems to be a world record. Exactly speaking we show that the set of EDS relations reduces the upper bound of $\dim_{\mathbf{Q}} \mathcal{Z}_k$ to d_k. That is, the verification is to check $\dim_{\mathbf{Q}} DS^{(k)} \geq 2^{k-2} - d_k$, where $DS^{(k)}$ denotes the $\mathbf{Q}$-vector space spanned by all EDS relations of weight k. There are several conjectures on the set of relations sufficient for reducing the dimension to d_k. Minh, Jacob, Oussous and Petitot [9] conjecture that Hoffman's relations and the FDS suffice and they checked it up to $k = 16$. We shall check (a stronger version of) their conjecture up to $k = 20$. We choose the weight $k = 20$ as our final target because the verification of the case is hard with respect to both the time and space complexity but it is expected that it is executable on an ordinary computing environment, if we apply well-known techniques such as elimination by sparse rows to reduce the size of a matrix, computation over a finite field and parallel Gaussian elimination. Detailed explanation of these techniques will be given in Section 4.2, 4.3 and 4.4 respectively.

4.1. Generation of the double shuffle relations. We denote the sets of polynomials giving all FDS and Hoffman's relations of weight k by $F^{(k)}$ and $E^{(k)}$ respectively. $F^{(k)}$ is given by $F^{(k)} = \bigcup_{i=2}^{\lfloor k/2 \rfloor} F_i^{(k)}$, where

$$F_i^{(k)} = \{w_1 \text{Ш} w_2 - w_1 * w_2 \mid w_1 \in \mathfrak{M}_i^0, w_2 \in \mathfrak{M}_{k-i}^0\}$$

and $\mathfrak{M}_i^0$ denotes the set of all monomials of weight i in $\mathfrak{H}^0$. Denoting the cardinality of a set S by $|S|$, $|\mathfrak{M}_i^0| = 2^{i-2}$ implies $|F_i^{(k)}| = 2^{k-4}$ for $i < k/2$. If k is even, $|F_{k/2}^{(k)}| = \frac{2^{k/2-2}(2^{k/2-2}+1)}{2}$, and $|E^{(k)}| = 2^{k-3}$. Thus we have

$$|F^{(k)}| + |E^{(k)}| > (\lfloor \frac{k}{2} \rfloor - 2)2^{k-4} + 2^{k-3}. \tag{4.1}$$

In particular we have $|F^{(k)}|+|E^{(k)}| > 2^{k-2}$ for $k \geq 8$. The verification of the conjecture is reduced to the rank computation of a matrix $M^{(k)}$ constructed from the coefficients of the relations. The inequality (4.1) means that the number of rows of $M^{(k)}$ is greater than that of columns and their ratio increases if k becomes large. Our purpose is to show $\text{rank}(M^{(k)}) \geq 2^{k-2} - d_k$ and it is sufficient to show this inequality for a sub-matrix of $M^{(k)}$. Our experiments for small k indicate that

$$R^{(k)} = E^{(k)} \cup F_2^{(k)} \cup F_3^{(k)}$$

suffices to attain the lower bound of the rank and we are led to a new conjecture:

CONJECTURE 4.1. *$R^{(k)}$ suffices to reduce the upper bound of $\dim_{\mathbf{Q}} \mathcal{Z}_k$ to d_k.*

If $k \geq 7$, $|R^{(k)}|$ is equal to 2^{k-2} and the matrix constructed from $R^{(k)}$ is a square matrix. As d_k is much smaller than 2^{k-2}, $R^{(k)}$ is practically optimal for our experimental verification. The FDS relations are generated by shuffle and harmonic products implemented in Risa/Asir. Hoffman's relations can be generated either by (2.6) or by the explicit representation (2.7). FIG. 1 actually shows the matrix constructed from $R^{(7)}$. Its upper-half part comes from Hoffman's relations and the lower-half part comes from the FDS relations. The matrix data for $k \leq 20$ are available from `http://www.math.kobe-u.ac.jp/OpenXM/Math/MZV/matdata`. For each k, $E^{(k)}$, $F_2^{(k)}$ and $F_3^{(k)}$ are converted to matrices and stored as files `mh`k, `mfds`k`_2` and `mfds`k`_3` respectively. These files are written according to the following format:

$$\boxed{(r\ c)\ (l_1 \cdots l_r)\ (j_{1,1}a_1) \cdots (j_{1,l_1}a_{l_1}) \cdots (j_{r,1}a_{s+1}) \cdots (j_{r,l_r}a_{s+l_r})}$$

where (r, c) denotes the size of the matrix, l_i denotes the number of non-zero elements in the i-th row and $(j_{i,k}, a_t)$ denotes a non-zero element a_t at $(i, j_{i,k})$. That is, non-zero elements are stored in row-major order. In the file, all numbers are four-byte integers and they are represented according to the network byte order.

4.2. Preprocessing by Hoffman's relations. If we convert $R^{(k)}$ itself to a dense matrix, then we need huge memory for a large k. Fortunately the matrix contains a large number of sparse rows coming from $E^{(k)}$ and we can apply preprocessing to eliminate many matrix entries with a small cost. The coefficients of $f \in E^{(k)}$ are 1 or -1 and the number of terms in f is $k-1$. Furthermore, under the lexicographic ordering we have the following (cf. FIG. 1):

PROPOSITION 4.1. *For any $xxw \in \mathfrak{M}_k^0$ there uniquely exists $f \in E^{(k)}$ such that the leading term of f is xxw.*

That is, the left-half of the $2^{k-3} \times 2^{k-2}$ sub-matrix obtained from $E^{(k)}$ is already upper triangular with unit diagonals. Thus we can easily eliminate the left-half of the sub-matrix obtained from $F_2^{(k)}$ and $F_3^{(k)}$ by using $E^{(k)}$. Note that this is a sparse elimination and it can be done efficiently. After this operation, we have the lower-right sub-matrix which is denoted by $S^{(k)}$. We set $N_k = 2^{k-3}$. $S^{(k)}$ is an $N_k \times N_k$ matrix and what we have to show is $\operatorname{rank}(S^{(k)}) \geq N_k - d_k$.

4.3. Rank computation over finite fields. The coefficients of the polynomials in $F_2^{(k)}$ and $F_3^{(k)}$ are within one machine word for $k \leq 20$. However, the result of the whole Gaussian elimination will have larger entries and it will be hard to execute the rank computation over the rationals. For any prime p we have $\operatorname{rank}(S^{(k)}) \geq \operatorname{rank}(S^{(k)} \bmod p)$. Therefore it is sufficient for our purpose to show $\operatorname{rank}(S^{(k)} \bmod p) \geq N_k - d_k$ for some prime p. In our experiments we use a two-byte prime $p = 31991$ for computing $\operatorname{rank}(S^{(k)} \bmod p)$. TABLE 1 shows the sizes of memory required for

TABLE 1
Required size of memory for storing $S^{(k)} \bmod p$.

k	16	17	18	19	20
size	128MB	512MB	2GB	8GB	32GB

storing $S^{(k)} \bmod p$. If a machine is not equipped with sufficient amount of memory, then a further preprocessing may be necessary. By examining $S^{(k)}$ for $k \leq 20$, we find that there are $N_k/4 \times N_k$ sparse sub-matrix whose left-quarter part is upper triangular. We can apply a sparse elimination by this sub-matrix, which results in a $3/4 \cdot N_k \times 3/4 \cdot N_k$ matrix. For example, 32GB of memory is required to store $S^{(20)} \bmod p$. If the second preprocessing is not applied then it is practically hard to execute the computation on a machine with just 32GB of memory, which is one of our computing environments. If we apply the preprocessing, then the required size is reduced to 18GB and we can efficiently compute the rank on that machine. We note that there are probabilistic Wiedemann-Krylov type algorithms to compute the rank of a matrix over finite fields. In general these are more efficient than Gaussian elimination in sparse cases and it is interesting to compare their performances in our cases.

4.4. Parallel computation by MPI. Even if we apply the preprocessing to reduce the size of the matrices, they are still huge and it is necessary to apply parallel computation from a practical point of view. We wrote two parallel programs by MPI to execute Gaussian elimination over finite fields. The implemented method is a simplification of the one used in ScaLAPACK [1]. That is, the whole matrix is decomposed according to 1- or 2-dimensional cyclic distribution algorithm and a non-blocking Gaussian elimination is executed in parallel.
`http://www.math.kobe-u.ac.jp/OpenXM/Math/MZV/gauss.c` is a C code for computing the rank of a matrix over a small finite field by using MPI. Each processor element reads the same input files containing fragments of the whole matrix, and stores a subset of rows of the input matrix into its local memory. At each step of eliminating a column, informations of the rows in the local matrices are shared among all processor elements to determine the pivot row. Then the selected pivot row is broadcasted to all processor elements and the elimination is done locally in each processor element. Both the algorithm and the implementation are not so optimized but the performance is satisfactory. We used several computing environments: (1) a cluster of heterogeneous linux PCs, (2) a cluster of large SMP Sparc/Solaris machines, and (3) an SMP linux PC. The last one is an 8 CPU SMP machine with two quad-core Intel X5355/2.66GHz CPUs and 32GB of memory. TABLE 2 shows various statistics in the last environment up to $k = 20$. In the table, "Generation", "Preprocessing" and "Elimination" show the elapsed time for generating all relations, preprocessing by

TABLE 2
Statistics of the rank computation in the environment (3).

k	16	17	18	19	20
d_k	37	49	65	86	114
N_k	8192	16384	32768	65536	131072
$\mathrm{rank}(S^{(k)} \bmod p)$	8155	16335	32703	65450	130958
$N_k - \mathrm{rank}(S^{(k)} \bmod p)$	37	49	65	86	114
Generation (1CPU)	22sec	85sec	4.5min	11min	30min
Preprocessing (1CPU)	5sec	19sec	1.5min	9min	57min
Elimination (8CPU)	2min	13min	1.3hour	9hour	67hour
Total memory	72MB	288MB	1.2GB	4.6GB	18GB

the sparse elimination and Gaussian elimination respectively. "Total memory" shows the size to store the $3/4 \cdot N_k \times 3/4 \cdot N_k$ matrix. The table shows that $\mathrm{rank}(S^{(k)} \bmod 31991) = N_k - d_k$ up to $k = 20$, thus the conjecture is verified up to $k = 20$. The table also tells us that the preprocessing is almost negligible compared with the final Gaussian elimination. We note that we also tried the same computations in the second environment with another parallel program and $p = 16381$, and that we obtained the same results of the ranks.

Acknowledgements. Prof. Nobuki Takayama gave us sincere encouragements and valuable comments on this work. Discussion with Dr. Naoyuki Shinohara was useful to implement an efficient representation of a non-commutative monomial in Risa/Asir. Prof. Kinji Kimura and Prof. Kazuhiro Yokoyama were interested in our work and provided several computing environments for our experiments.

REFERENCES

[1] L.S. BLACKFORD et al., *ScaLAPACK Users' Guide*, SIAM (1997), `http://www.netlib.org/scalapack/slug/`.

[2] P. CARTIER, *Fonctions polylogarithmes, nombres polyzetas et groupes pro-unipotents*, Séminaire Bourbaki, 53 éme année, 2000–2001, **885** (2001).

[3] M. ESPIE, J-C. NOVELLI, AND G. RACINET, *Formal computations about multiple zeta values*, in "From Combinatorics to Dynamical Systems" (Strasbourg, 2002), IRMA Lect. Math. Theor. Phys. 3, F. Fauvet and C. Mitschi (eds.), de Gruyter, Berlin (2003), 1–16.

[4] L. EULER, *Meditationes circa singulare serierum genus*, Novi Comm. Acad. Sci. Petropol, **20** (1775), 140–186, reprinted in Opera Omnia ser. I, Vol. 15, B.G. Teubner, Berlin (1927), 217–267.

[5] A.B. GONCHAROV, *Periods and mixed motives*, preprint (2002).

[6] M. HOFFMAN, *The algebra of multiple harmonic series*, J. Algebra, **194** (1997), 477–495.

[7] K. IHARA, M. KANEKO AND D. ZAGIER, *Derivation and double shuffle relations for multiple zeta values*, Compositio Math., **142-02** (2006), 307–338.

[8] M. KANEKO, *Multiple zeta values* (translation of Sugaku, **54** (2002), no. 4, 404–415), Sugaku Expositions, **18** (2005), no. 2, 221–232.
[9] H.N. MINH, G. JACOB, N.E. OUSSOUS AND M. PETITOT, *Aspects combinatoires des polylogarithmes et des sommes d'Euler-Zagier*, J. Électr. Sém. Lothar. Combin., **43** (2000), Art. B43e, pp. 29.
[10] C. REUTENAUER, *Free Lie Algebras*, Oxford Science Publications, 1993.
[11] T .TERASOMA, *Mixed Tate motives and multiple zeta values*, Invent. Math., **149** (2002), 339–369.
[12] M. WALDSCHMIDT, *Valeurs zeta multiples: une introduction*, J. Theor. Nombres Bordeaux, **12** (2000), 581–595.
[13] D. ZAGIER, *Values of zeta functions and their applications*, in ECM volume, Progress in Math., **120** (1994), 497–512.

DEMICS: A SOFTWARE PACKAGE FOR COMPUTING THE MIXED VOLUME VIA DYNAMIC ENUMERATION OF ALL MIXED CELLS

TOMOHIKO MIZUTANI* AND AKIKO TAKEDA*†

Abstract. DEMiCs is a software package written in C++ for computing the mixed volume of the Newton polytopes of a general semi-mixed polynomial system through dynamic enumeration of all mixed cells. The underlying mixed cells play an essential role for computing all isolated zeros of a polynomial system by polyhedral homotopy continuation method. A notable feature of DEMiCs is in the construction of a dynamic enumeration tree for finding all mixed cells. The dynamic enumeration method, proposed by Mizutani, Takeda and Kojima for fully mixed polynomial systems, is extended to semi-mixed systems and incorporated in the package. Numerical results show that DEMiCs significantly is faster than existing software packages for semi-mixed polynomial systems with many distinct supports. The software package DEMiCs is available at `http://www.is.titech.ac.jp/~mizutan8/DEMiCs/`.

Key words. mixed volume, mixed cell, polyhedral homotopy, polynomial system, semi-mixed structure, dynamic enumeration.

AMS(MOS) subject classifications. Primary 68N30, 52A39, 90C05.

1. Introduction. The polyhedral homotopy continuation method, proposed by Huber and Sturmfels [15] and Verschelde et al. [24], is a powerful and reliable numerical method [7, 13, 14, 16, 17, 25] for computing all isolated zeros of a polynomial system $\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x}))$ in the variables $\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{C}^n$. In order to build a family of homotopy functions, this method needs all mixed cells in a fine mixed subdivision of the Newton polytopes of a polynomial system $\boldsymbol{f}(\boldsymbol{x})$ (See [15] and [17]). It uses the mixed cells to construct homotopy functions between start systems, which are polynomial systems whose zeros can be computed easily, and target system $\boldsymbol{f}(\boldsymbol{x})$. The mixed volume, which can be obtained from the volumes of all mixed cells, is also the total number of solutions for the start systems. It is also an upper bound for the total number of isolated zeros in $(\mathbb{C} \setminus \{0\})^n$ of $\boldsymbol{f}(\boldsymbol{x})$, as guaranteed by Bernshtein's theorem [1].

The aim of this paper is to introduce the software package DEMiCs. The package uses dynamic enumeration to compute all mixed cells and the mixed volume of the support of a general semi-mixed polynomial system, including fully mixed and unmixed systems as special cases. The dynamic enumeration method was originally developed for a fully mixed system in [19], and it is significantly faster than other software packages [8, 10, 12, 18, 25, 22] for enumerating all mixed cells for large-scale semi-mixed systems

*Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552, Japan (mizutan8@is.titech.ac.jp).
†(takeda@is.titech.ac.jp).

with many distinct support sets. It also opens the door to computing all zeros of larger polynomial systems by polyhedral homotopy.

The mixed cells are enumerated by using a tree on which each node is provided with a linear inequality system. An enumeration tree has the following properties:

(i) A leaf node corresponds to a mixed cell if and only if the linear inequality system attached to the leaf node is feasible.

(ii) Each node shares a common system with the child nodes, so that if the node is infeasible, so are all of its descendant nodes.

There are various ways of constructing such enumeration trees. The static enumeration method [9, 10, 18, 25] fixes the structure of a tree before finding mixed cells. However, it is ideal that most of the child nodes are infeasible and pruned when branching at each node is carried out. To pursue this idea, the paper [19] proposed the dynamic enumeration method for a fully mixed system. This method chooses a suitable child node set among all the candidates generated from a parent node, so that the total number of nodes of a tree is expected to be small. In this situation, it is essential for computational efficiency to utilize information from each node to be generated during the construction of a tree. In this paper, we explain how the dynamic enumeration method is extended to a semi-mixed polynomial system. Recently, we noticed that Emiris and Canny [8] also enumerate all mixed cells dynamically for fully mixed systems, and build the dynamic enumeration tree implicitly for this purpose. Both methods use a linear programming (LP) problem as a means to enumeration mixed cells. The main differences between the two methods are as follows.

(i) Formulation for finding mixed cells: Our formulation is well suited for utilizing rich information which can be obtained by solving the LP problems at upper level of a tree.

(ii) Choice of a child node set at each node: Our method efficiently chooses a suitable child node set in all the candidates generated from a parent node, using information obtained from the LP problems to be solved during the execution of enumeration. Emiris and Canny's method select the best child node set in all the candidates without taking account of such information, and hence, it is more computationally expensive.

In fact, the numerical results in [10, 18, 22] show that Emiris and Canny's dynamic enumeration strategy has a speed disadvantage. In Subsection 3.3, we precisely describe how the two methods construct a dynamic enumeration tree.

There are several related software packages for computing the mixed volume through enumeration of all mixed cells: HOM4PS [11], MixedVol

[10, 12], MVLP [8], PHCpack [25], PHoM [22], mvol [18], etc. HOM4PS, PHCpack and PHoM, written in FORTRAN, Ada and C++ respectively, are software packages for computing all isolated zeros of a polynomial system by polyhedral homotopy. These packages each contain a module for enumeration of mixed cells. In particular, PHCpack is the most popular of these software packages. The C package MVLP uses a dynamic enumeration for finding all mixed cells. The C++ package MixedVol, which employs the static enumeration method, specializes in the mixed volume computation. MixedVol performs better than all other software packages in terms of computational efficiency and memory usage, as reported in the papers [10, 12]. A feature of the package is that all mixed cells can be generated efficiently for a semi-mixed system by taking account of the special structure of a semi-mixed support.

Numerical results show that DEMiCs can find all mixed cells much faster than the other software packages [8, 10, 12, 18, 25, 22] not only for fully mixed polynomial systems but also for semi-mixed polynomial systems. Although DEMiCs has a speed advantage over the other packages for large-scale fully mixed systems and semi-mixed systems with many distinct support sets, it does not always excel. Indeed, for unmixed systems and semi-mixed systems with a few distinct supports, DEMiCs is slower than MixedVol [12] because of its computation overhead associated with the dynamic branching rule.

This paper is organized as follows. Section 2 and 3 describe the technical details of our method. Section 2 outlines the dynamic enumeration method for a general semi-mixed polynomial system, and Section 3 explains how to check the feasibility of each node and how to construct an enumeration tree when a polynomial system is a semi-mixed type. Section 4 describes the usage of DEMiCs. Section 5 compares the performance of DEMiCs with those of the other software packages on artificial semi-mixed polynomial systems and well-known large-scale benchmark systems. Section 6 is devoted to concluding remarks.

2. Dynamic enumeration algorithm for a semi-mixed system.

2.1. Preliminaries. In this paper, we represent each component polynomial $f_i(\boldsymbol{x})$ in a polynomial system $\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x}))$ in $\boldsymbol{x} \in \mathbb{C}^n$ as

$$f_i(\boldsymbol{x}) = \sum_{\boldsymbol{a} \in \mathcal{A}_i} c_i(\boldsymbol{a})\, \boldsymbol{x}^{\boldsymbol{a}},$$

using a nonempty finite subset $\mathcal{A}_i$ of $\mathbb{Z}_+^n$ and nonzero $c_i(\boldsymbol{a}) \in \mathbb{C}$ for $\boldsymbol{a} \in \mathcal{A}_i$. Here $\mathbb{Z}_+^n$ denotes the set of nonnegative integer vectors in $\mathbb{R}^n$, $\mathbb{R}$ and $\mathbb{C}$ are the sets of real and complex numbers, respectively, and $\boldsymbol{x}^{\boldsymbol{a}} = x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}$ for $\boldsymbol{a} = (a_1, a_2 \ldots, a_n) \in \mathbb{Z}_+^n$. We call the set $\mathcal{A}_i$ the support of $f_i(\boldsymbol{x})$, and $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n)$ the support of $\boldsymbol{f}(\boldsymbol{x})$. The convex hull $\mathcal{P}_i := \mathrm{conv}(\mathcal{A}_i)$ is called the *Newton polytope* of f_i. Define $N = \{1, 2, \ldots, n\}$.

The following describes the definition of the mixed volume of the n-tuple Newton polytopes $\mathcal{P}_i$ in $\mathbb{R}^n$. For all positive number $\lambda_1, \lambda_2, \ldots, \lambda_n$, it is known that the n-dimensional volume of the Minkowski sum

$$\lambda_1\mathcal{P}_1+\lambda_2\mathcal{P}_2+\cdots+\lambda_n\mathcal{P}_n = \{\lambda_1 p_1+\lambda_2 p_2+\cdots+\lambda_n p_n : p_i \in \mathcal{P}_i,\ i \in N\}$$

is given by a homogeneous polynomial of degree n in λ_i $(i \in N)$. The mixed volume for $\mathcal{A}$ is defined to be the coefficient of $\lambda_1\lambda_2\cdots\lambda_n$ in the polynomial.

Some support sets in $\mathcal{A} = (\mathcal{A}_i : i \in N)$ of $\boldsymbol{f}(\boldsymbol{x})$ may be equal to each other. We suppose that the polynomial system has exactly m $(\leq n)$ distinct support sets $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m$ among $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$ such that

$$\mathcal{S}_i := \mathcal{A}_{j_1} = \mathcal{A}_{j_2} \text{ for every } j_1, j_2 \in I_i \quad (i \in M), \tag{2.1}$$

where we define $M = \{1, 2, \ldots, m\}$. Obviously, the subset I_i of N satisfies $\cup_{i\in M} I_i = N$ and $I_{i_1} \cap I_{i_2} = \emptyset$ for every $i_1, i_2 \in M$. A polynomial system with the support $\mathcal{S} = (\mathcal{S}_i : i \in M)$ is called *semi-mixed.* The system is called *unmixed* when $m = 1$, and *fully mixed* when $m = n$, and it is called *semi-mixed of type* $(k_1, k_2, \ldots, k_m)$ if and only if $\mathcal{S}_i$ occurs exactly k_i times in $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$, *i.e.*, $\#I_i = k_i$. In this paper, we treat a semi-mixed polynomial system $\boldsymbol{f}(\boldsymbol{x})$ of type $(k_1, k_2, \ldots, k_m)$ with support $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m)$, and assume that each support set $\mathcal{S}_i$ consists of r_i elements.

For a support $\mathcal{S} = (\mathcal{S}_i : i \in M)$ of a semi-mixed system described above, we use the notation $\mathcal{Q}_i = \mathrm{conv}(\mathcal{S}_i)$ for the Newton polytope of each polynomial. The mixed volume for $\mathcal{S} = (\mathcal{S}_i : i \in M)$ of a semi-mixed type can be computed by finding all *mixed cells* in *a fine mixed subdivision* of the Minkowski sum $\mathcal{Q} = \mathcal{Q}_1 + \mathcal{Q}_2 + \cdots + \mathcal{Q}_m$. These are essentially piece polytopes, called *cells*, in a polyhedral subdivision of $\mathcal{Q}$. The reader may want to refer to the definition of a fine mixed subdivision in [6, 15]. It is known that each cell in a fine mixed subdivision can be represented as the Minkowski sum of simplices $R_j := \mathrm{conv}(C_1^j) + \mathrm{conv}(C_2^j) + \cdots + \mathrm{conv}(C_m^j)$ for $\boldsymbol{C} = (C_1^j, C_2^j, \ldots, C_m^j)$ where each C_i^j is the subset of $\mathcal{S}_i$ and its convex hull $\mathrm{conv}(C_i^j)$ is a simplex of dimension $\#C_i^j - 1$. In particular, when a polynomial system is a semi-mixed system of type $(k_1, k_2, \ldots, k_m)$, we call a cell R_j that is described as a Minkowski sum of each $\mathrm{conv}(C_i^j)$, a *mixed cell* if $\dim(\mathrm{conv}(C_i^j)) = k_i$ for every $i \in M$. It is shown in [15, Theorem 2.4.] that the mixed volume for $\mathcal{S}$ is obtained from the volumes of all mixed cells in a fine mixed subdivision of $\mathcal{Q}$. This means that only distinct support sets contribute the mixed volume computation. A semi-mixed system certainly can be treated as a fully mixed type, and the mixed volume can be obtained by summing the volumes of all mixed cells in a fine mixed subdivision of $\mathcal{P} = \mathcal{P}_1 + \mathcal{P}_2 + \cdots + \mathcal{P}_n$. However, the numerical results in [10] show that the computational time for finding all mixed cells can be reduced if we focus

on the only distinct support sets for a semi-mixed system. Therefore, it is important for computational efficiency to utilize a semi-mixed structure.

The papers [2, 15] describe how to construct a fine mixed subdivision of $\mathcal{Q}$ by using a real-valued function $\omega_i : \mathcal{S}_i \to \mathbb{R}$. The function ω_i lifts $\mathcal{S}_i$ to

$$\hat{\mathcal{S}}_i = \left\{ \begin{pmatrix} \boldsymbol{a} \\ \omega_i(\boldsymbol{a}) \end{pmatrix} : \boldsymbol{a} \in \mathcal{S}_i \right\}.$$

Let $\hat{\mathcal{Q}}$ denote the Minkowski sum $\hat{\mathcal{Q}} = \hat{\mathcal{Q}}_1 + \hat{\mathcal{Q}}_2 + \cdots + \hat{\mathcal{Q}}_m$ for $\hat{\mathcal{Q}}_i = \mathrm{conv}(\hat{\mathcal{S}}_i)$. For the subset $\hat{C}_i$ of $\hat{\mathcal{S}}_i$, we will use the notation $\hat{\boldsymbol{C}} = (\hat{C}_1, \hat{C}_2, \ldots, \hat{C}_m)$. Suppose that the function ω_i gives a random number so that the value $\omega_i(\boldsymbol{a})$ is sufficiently generic for every $i \in M$ and every $\boldsymbol{a} \in \mathcal{S}_i$. Then, the projection $\mathbb{R}^{n+1} \to \mathbb{R}^n$ of the set of lower facets of $\hat{\mathcal{Q}}$ gives a fine mixed subdivision of $\mathcal{Q}$. In this paper, we call such a function ω_i a *generic lifting.*

Li and Li [18] proposed an efficient algorithm for finding lower facets of $\hat{\mathcal{Q}}$ via an enumeration tree. Recently, for a fully mixed polynomial system, the paper [19] improved their algorithm by replacing a static enumeration tree of [9, 10, 18, 22] with a dynamic enumeration tree. In this paper, a dynamic enumeration method is applied to find all mixed cells in a fine mixed subdivision for a semi-mixed system, including a fully mixed and unmixed type.

2.2. Algorithm. We briefly describe the *dynamic enumeration algorithm* of [19] and apply it to a semi-mixed polynomial system. For every $L \subseteq M = \{1, 2, \ldots, m\}$, we define

$$\Omega(L) = \left\{ \boldsymbol{C} = (C_1, C_2, \ldots, C_m) : \begin{array}{l} C_i \subseteq \mathcal{S}_i,\ \#C_i = k_i + 1\ (i \in L) \\ C_j = \emptyset\ (j \notin L) \end{array} \right\}$$

$$\Omega = \cup_{L \subseteq M} \Omega(L).$$

The set Ω represents the collection of all nodes in an enumeration tree. The tree has a root node $\emptyset^m \in \Omega(\emptyset) := \{\emptyset^m\}$ and the leaf nodes $\Omega(M) \subset \Omega$. A node at the ℓth level corresponds to the element in $\cup_{L \subseteq M, \#L = \ell} \Omega(L)$. Let $L(\boldsymbol{C}) = \{i \in M : C_i \neq \emptyset\}$ for any $\boldsymbol{C} \in \Omega(L)$ $(L \subseteq M)$. This definition is used for extracting every index of nonempty sets C_i from $\boldsymbol{C} = (C_1, C_2, \ldots, C_m) \in \Omega(L)$, *i.e.* $L(\boldsymbol{C}) = L$ for $\boldsymbol{C} \in \Omega(L)$ $(L \subseteq M)$. Each node $\boldsymbol{C} = (C_i : i \in L) \in \Omega(L)$ with $L \subseteq M$ has a linear inequality system $\mathcal{I}(\boldsymbol{C})$:

$$\mathcal{I}(\boldsymbol{C}) := \left\{ \begin{array}{ll} \langle \hat{\boldsymbol{a}}_i, \hat{\boldsymbol{\alpha}} \rangle = \langle \hat{\boldsymbol{a}}_i', \hat{\boldsymbol{\alpha}} \rangle, & \forall \hat{\boldsymbol{a}}_i, \hat{\boldsymbol{a}}_i{}' \in \hat{C}_i \\ \langle \hat{\boldsymbol{a}}_i, \hat{\boldsymbol{\alpha}} \rangle \leq \langle \hat{\boldsymbol{a}}, \hat{\boldsymbol{\alpha}} \rangle, & \forall \hat{\boldsymbol{a}} \in \hat{\mathcal{S}}_i \setminus \hat{C}_i \end{array} \right. \quad (i \in L(\boldsymbol{C})), \tag{2.2}$$

where

$$\hat{\boldsymbol{\alpha}} = \begin{pmatrix} \boldsymbol{\alpha} \\ 1 \end{pmatrix} \in \mathbb{R}^{n+1}.$$

$\langle \cdot, \cdot \rangle$ stands for the usual inner product in Euclidean space. Note that $\hat{\mathcal{S}}_i$ is obtained by applying a generic lifting ω_i to $\mathcal{S}_i$, and $\hat{C}_i$ is the subset of $\hat{\mathcal{S}}_i$. Li and Li showed in [18] that any mixed cell in a fine mixed subdivision of $\mathcal{S} = (\mathcal{S}_i : i \in M)$ is in one-to-one correspondence to $\boldsymbol{C} = (C_1, C_2, \ldots, C_m)$ with $C_i \subseteq \mathcal{S}_i$ and $\#C_i = k_i + 1$ for each $i \in M$ such that the linear inequality system $\mathcal{I}(\boldsymbol{C})$ is feasible. We say that $\boldsymbol{C} \in \Omega$ is a feasible node when $\mathcal{I}(\boldsymbol{C})$ is feasible. Let

$$\Omega^* = \{\boldsymbol{C} \in \Omega(M) : \boldsymbol{C} \text{ is feasible}\}.$$

Then we can easily see from (2.2) that Ω^* consists of every mixed cell in a fine mixed subdivision.

For $\boldsymbol{C} \in \Omega$ and $L \subseteq M$, we use the notation $\boldsymbol{C}_L$ for $\boldsymbol{C}_L = (C_i : i \in L)$. Regarding the root node $\emptyset^m \in \Omega$ as a feasible node, we construct an enumeration tree according to Algorithm 2.1 of [19]. Namely, for a node $\boldsymbol{C} \in \Omega(L)$ with the proper subset $L \subsetneq M$ and $t \in M \setminus L(\boldsymbol{C})$ we generate the child node set $W(\boldsymbol{C}, t)$ of $\boldsymbol{C}$

$$W(\boldsymbol{C}, t) = \left\{\bar{\boldsymbol{C}} \in \Omega(L(\boldsymbol{C}) \cup \{t\}) : \bar{\boldsymbol{C}}_{L(\boldsymbol{C})} = \boldsymbol{C}_{L(\boldsymbol{C})}\right\}.$$

Starting from the root node $\emptyset^m \in \Omega$, we choose t from $M \setminus L(\boldsymbol{C})$ at each node $\boldsymbol{C} \in \Omega(L)$ with $L \subsetneq M$ and create child nodes of $\boldsymbol{C}$ until $\#L = m - 1$ based on the algorithm. Thus, Ω^* coincides with the set of the feasible leaf nodes $\boldsymbol{C} \in \Omega(M)$. If we check the feasibility of all leaf nodes, all mixed cells in a fine mixed subdivision can be obtained. Note that this algorithm produces various types of trees depending on a choice of an index $t \in M \setminus L(\boldsymbol{C})$ at each node $\boldsymbol{C} \in \Omega(L)$ with $L \subsetneq M$. *A static enumeration tree* is constructed in the previous works [9, 10, 18, 22], which specify how to choose an index $t \in M \setminus L(\boldsymbol{C})$ at each node $\boldsymbol{C} \in \Omega(L)$ with $L \subsetneq M$ before the building of a tree. In contrast to this, the paper [19] develops a *dynamic enumeration tree* by choosing a suitable index t from $M \setminus L(\boldsymbol{C})$ at each node $\boldsymbol{C} \in \Omega(L)$ with $L \subsetneq M$.

We can enumerate feasible leaf nodes of a tree efficiently if the following property is taken into account. The feasible region of the linear inequality system $\mathcal{I}(\boldsymbol{C})$ attached to a node $\boldsymbol{C}$ contains that of $\mathcal{I}(\bar{\boldsymbol{C}})$, which corresponds to a child node $\bar{\boldsymbol{C}}$ of $\boldsymbol{C}$. That is, we can say that if a parent node $\boldsymbol{C}$ is infeasible, then all child nodes $\bar{\boldsymbol{C}} \in W(\boldsymbol{C}, t)$ $(t \in M \setminus L(\boldsymbol{C}))$ are infeasible. If a node is detected to be infeasible, we can prune a subtree having the node as the root node because there are no mixed cells in the subtree. Therefore, by replacing $W(\boldsymbol{C}, t)$ in Algorithm 2.1 of [19] with

$$W^*(\boldsymbol{C}, t) = \{\bar{\boldsymbol{C}} \in W(\boldsymbol{C}, t) : \bar{\boldsymbol{C}} \text{ is feasible}\} \subseteq W(\boldsymbol{C}, t),$$

every mixed cell can be found as the feasible leaf nodes in the tree. Taking account of this property, we search for all nodes in Ω^* according to the dynamic enumeration algorithm stated below. We will use the notation A_ℓ $(\ell \in \{0\} \cup M)$ for the set of feasible nodes.

Algorithm 1 Dynamic enumeration algorithm.

Input: A support $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m)$.
Output: All mixed cells in a fine mixed subdivision.

$A_i \leftarrow \emptyset$ for all $i \in M$.
$A_0 \leftarrow \emptyset^m$ and $\ell \leftarrow 0$.

while $\ell < m$ **do**
  for all $\boldsymbol{C} \in A_\ell$ **do**
    Choose t from $M \setminus L(\boldsymbol{C})$: (A)
    $A_{\ell+1} \leftarrow A_{\ell+1} \cup W^*(\boldsymbol{C}, t)$: (B)
  end for
  $\ell \leftarrow \ell + 1$.
end while

Once this algorithm terminates, A_m contains all nodes in Ω^*, *i.e,* every mixed cell can be stored in A_m. We introduce this algorithm, which constructs a tree in breadth-first order, for simplicity of description; it is essentially the same as Algorithm 2.2 described in [19], which is in depth-first order. Hence, we can obtain the same result from these two algorithms though there is the difference in the search order for feasible leaf nodes. The software package DEMiCs employs the depth-first order, which is similar to Algorithm 2.2 in [19], to save memory.

The following two issues have a major effect on the computational efficiency of the dynamic enumeration algorithm for semi-mixed systems.

(i) How we choose an index t from $M \setminus L(\boldsymbol{C})$ in (A).

(ii) How we construct $W^*(\boldsymbol{C}, t)$ in (B).

As for (i), in the *static enumeration method* proposed in the previous works [9, 10, 18, 22], we set up a permutation π of M before starting the algorithm, and choose the index t such as $t = \pi(\ell+1) \in M \setminus L(\boldsymbol{C})$. Hence, a choice of an index t at (A) is determined by a permutation π. Here we employ a refinement of the *dynamic enumeration method* from [19]. Suppose that $\boldsymbol{C}$ is a feasible node in A_ℓ with $\ell < m$. Then, we consider a choice of an index t from $M \setminus L(\boldsymbol{C})$ so that many child nodes of $\boldsymbol{C}$ are expected to be infeasible. Ideally, we would like to choose the index t such that the size of $W^*(\boldsymbol{C}, t)$ is the smallest among $t \in M \setminus L(\boldsymbol{C})$. Although this strategy is employed by Emiris and Canny's method [8], it is costly because we need to construct $W^*(\boldsymbol{C}, t)$ for each $t \in M \setminus L(\boldsymbol{C})$. Therefore, instead of $W^*(\boldsymbol{C}, t)$, we consider another set which can be constructed easily. In Subsection 3.3, we explain how to construct this set.

As for (ii), it may not be an easy task to find all feasible nodes in $W(\boldsymbol{C},t)$ because the size of $W(\boldsymbol{C},t)$ is not small. So we embed the construction process for $W^*(\boldsymbol{C},t)$ in a tree, and prune worthless subtrees in order to reduce computational effort. In Subsection 3.1 we discuss the details of this procedure, and show the formulation of the feasibility check of a node in Subsection 3.2.

3. Feasibility check for a semi-mixed system.

3.1. Tree structure for construction of $W^*(\boldsymbol{C},t)$ in (B). We now explain a tree structure for finding all elements in $W^*(\boldsymbol{C},t)$ efficiently. Suppose that an index t is chosen from $M \setminus L(\boldsymbol{C})$ for a feasible node $\boldsymbol{C} \in A_\ell$ with $\ell < m$ in (B) of Algorithm 1. Then, we would like to construct $W^*(\boldsymbol{C},t) \subseteq W(\boldsymbol{C},t)$. For a nonnegative integer k, let

$$\Gamma(k;\boldsymbol{C},t) = \left\{ U = (U_1, U_2, \ldots, U_m) : \begin{array}{l} \boldsymbol{U}_{L(\boldsymbol{C})} = \boldsymbol{C}_{L(\boldsymbol{C})} \\ U_t \subseteq \mathcal{S}_t,\ \#U_t = k \\ U_i = \emptyset\ (i \notin L(\boldsymbol{C}) \cup \{t\}) \end{array} \right\}.$$

Note that $\Gamma(k_t+1;\boldsymbol{C},t) = W(\boldsymbol{C},t)$.

For a feasible node $\boldsymbol{C} \in A_\ell$ with $\ell < m$ in the dynamic enumeration algorithm, we build a tree T for constructing $W^*(\boldsymbol{C},t)$. The tree structure is outlined as follows. Let $K_t = \{0,1,\ldots,k_t+1\}$. The set

$$\Gamma := \cup_{k \in K_t} \Gamma(k;\boldsymbol{C},t)$$

serves as the collection of all nodes in the tree. The tree has $\boldsymbol{C} \in \Gamma(0;\boldsymbol{C},t) = \{\boldsymbol{C}\}$ as the root node, and $\boldsymbol{U} \in \Gamma(k;\boldsymbol{C},t)$ with $\#U_t = k$ as a node at the kth level. Each node $\boldsymbol{U} \in \Gamma(k;\boldsymbol{C},t)$ has a linear inequality system $\mathcal{I}(\boldsymbol{U})$ in a variable vector $\hat{\boldsymbol{\alpha}} \in \mathbb{R}^{n+1}$. Note that each system $\mathcal{I}(\boldsymbol{U})$ with $\boldsymbol{U} \in \Gamma(k_t+1;\boldsymbol{C},t)$ is identical to the linear inequality system $\mathcal{I}(\bar{\boldsymbol{C}})$ at a node $\bar{\boldsymbol{C}} \in W(\boldsymbol{C},t)$. Therefore, $W^*(\boldsymbol{C},t)$ can be obtained by checking the feasibility of any node $\boldsymbol{U} \in \Gamma(k_t+1;\boldsymbol{C},t)$ which corresponds to each leaf node at the (k_t+1)th level.

We now describe more precisely the tree structure for building $W^*(\boldsymbol{C},t)$. For $U_t \subseteq \mathcal{S}_t$, we choose a function $m_t : \mathcal{S}_t \to \mathbb{Z}$ which gives the maximum number i among the indices of $\boldsymbol{a}_i \in U_t$. Namely, $m_t(U_t) = \max\{i \in \mathbb{Z} : \boldsymbol{a}_i \in U_t\}$ for $U_t \subseteq \mathcal{S}_t$. Let $T = (V,E)$ be a rooted tree, which describes the relation among elements of a node set Γ. Recall that $\mathcal{S}_t$ has r_t elements. For a node $\boldsymbol{U} \in \Gamma(k;\boldsymbol{C},t)$, we generate the child node set

$$Z(\boldsymbol{U};\boldsymbol{C},t) = \left\{ \bar{\boldsymbol{U}} \in \Gamma(\#U_t+1;\boldsymbol{C},t) : \begin{array}{c} \bar{U}_t = U_t \cup \{\boldsymbol{a}_i\}, \\ \text{for every } i,\ m_t(U_t) < i \le r_t \end{array} \right\}$$

and construct $T = (V,E)$ with $V = \bigcup_{k \in K_t} V_k$ and $E = \bigcup_{k \in K_t} E_k$, based on Algorithm 2. The root node of T, constructed by the algorithm, corresponds to $V_0 = \{\boldsymbol{C}\}$, where $\boldsymbol{C}$ is one of the elements in A_ℓ generated by

Algorithm 2 Construction of the tree $T = (V, E)$.

Input: A feasible node $\boldsymbol{C} \in A_\ell$ and an index $t \in M \setminus L(\boldsymbol{C})$.
Output: $T = (V, E)$ with $V = \bigcup_{k \in K_t} V_k$ and $E = \bigcup_{k \in K_t} E_k$.

$V_0 \leftarrow \boldsymbol{C}$, $E_0 \leftarrow \emptyset$ and $k \leftarrow 0$.
$V_i \leftarrow \emptyset$ and $E_i \leftarrow \emptyset$ for all $i \in K_t \setminus \{0\}$.

while $k < k_t + 1$ **do**
  for all $\boldsymbol{U} \in V_k$: (♮) **do**
    if $m_t(U_t) < r_t$ **then**

$$V_{k+1} \leftarrow V_{k+1} \cup Z(\boldsymbol{U}; \boldsymbol{C}, t)$$
$$E_{k+1} \leftarrow E_{k+1} \cup \{(\boldsymbol{U}, \bar{\boldsymbol{U}})) \in V_k \times V_{k+1} : \bar{\boldsymbol{U}} \in Z(\boldsymbol{U}; \boldsymbol{C}, t)\}$$

    end if
  end for
  $k \leftarrow k + 1$.
end while

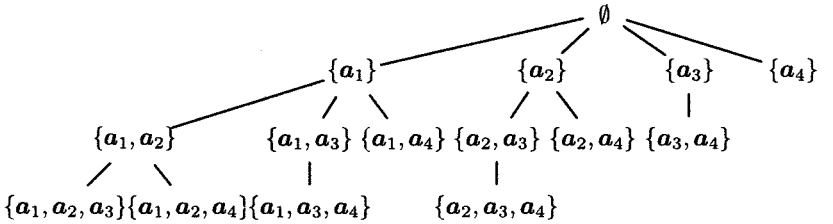


FIG. 1. *Tree structure generated by Algorithm 2.*

Algorithm 1. When this algorithm terminates, V_k stores every element in $\Gamma(k; \boldsymbol{C}, t)$ for all $k \in K_t$. Hence, $W(\boldsymbol{C}, t)$, which is generated by a feasible $\boldsymbol{C} \in A_\ell$ and an index $t \in M \setminus L(\boldsymbol{C})$ in Algorithm 1, is equal to V_{k_t+1}, which has each leaf node at the $(k_t + 1)$th level of the tree T.

EXAMPLE 1. *We consider the distinct support sets* $\mathcal{S}_1, \mathcal{S}_2$ $(\mathcal{S}_i \subseteq \mathbb{Z}_+^3)$ *of the semi-mixed system* $\boldsymbol{f}(\boldsymbol{x})$ $(\boldsymbol{x} \in \mathbb{C}^3)$ *of type* $(1, 2)$ *such that* $|\mathcal{S}_1| = 3$ *and* $|\mathcal{S}_2| = 4$. *We execute Algorithm 1 for the support* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$. *Suppose that the algorithm produces a nonempty set* A_1 *when* $t = 1$ *is chosen in (A) at the first iteration. Figure 1 shows the shape of the tree generated by Algorithm 2 for input data* $\boldsymbol{C} \in A_1$. *Here, each label of nodes represents* $U_2 \subseteq \mathcal{S}_2$ *of* $\boldsymbol{U} = (U_1, U_2) \in \Gamma(k; \boldsymbol{C}, 2)$ $(0 \leq k \leq 3)$.

For a node $\boldsymbol{U} \in V_k$ and child node $\bar{\boldsymbol{U}} \in V_{k+1}$ of $\boldsymbol{U}$, the feasible region of $\mathcal{I}(\boldsymbol{U})$ contains that of $\mathcal{I}(\bar{\boldsymbol{U}})$. Hence, if the node $\boldsymbol{U} \in V_k$ is infeasible, every child node $\bar{\boldsymbol{U}} \in V_{k+1}$ of $\boldsymbol{U}$ is infeasible. Therefore, the subtree with root node $\boldsymbol{U}$ can be pruned, since it does not contain any feasible leaf nodes at the (k_t+1)th level. Accordingly, even if we replace V_k at ($\natural$) of Algorithm 2 by

$$V_k^* = \{\boldsymbol{U} \in V_k : \boldsymbol{U} \text{ is feasible}\},$$

we can find every feasible leaf node at the (k_t+1)th level in $V_{k_t+1}^*$, and thus obtain $W^*(\boldsymbol{C}, t)$.

After building $T = (V, E)$, we enumerate all feasible nodes at the (k_t+1)th level in T and construct $V_{k_t+1}^*$. First, we initialize V_0^* as $V_0^* = V_0$. Next, we repeat the following procedure until $k = k_t$. Suppose that V_k^* is constructed. Then, we check the feasibility of every child node $\bar{\boldsymbol{U}}$ of $\boldsymbol{U} \in V_k^*$ and store the feasible nodes in V_{k+1}^*. As a result, $W^*(\boldsymbol{C}, t)$ is obtained as $V_{k_t+1}^*$. Note that here we use Algorithm 2 having a breadth-first order for the simplicity of description and that DEMiCs has a depth-first order to save memory in constructing $W^*(\boldsymbol{C}, t)$.

3.2. Formulation of the feasibility check. Suppose that a tree $T = (V, E)$ with $V = \bigcup_{k \in K_t} V_k$ and $E = \bigcup_{k \in K_t} E_k$ is generated by Algorithm 2 for a feasible node $\boldsymbol{C} \in A_\ell$ and an index $t \in M \setminus L(\boldsymbol{C})$. Then, we need to check the feasibility of each node $\boldsymbol{U} \in V_k$ in order to construct V_k^*. An LP problem can be formulated for the feasibility check of a node $\boldsymbol{U} \in V_k$. Let $\boldsymbol{\gamma}$ denote a specific n-dimensional real vector, and furthermore let $\hat{\boldsymbol{\gamma}}^T = (\boldsymbol{\gamma}^T, 0)$. To determine whether a node $\boldsymbol{U} \in V_k$ is feasible or not, we solve the following problem in a variable vector $\hat{\boldsymbol{\alpha}} \in \mathbb{R}^{n+1}$:

$$\mathrm{P}(\boldsymbol{U}): \ \text{maximize} \quad \langle \hat{\boldsymbol{\gamma}}, \hat{\boldsymbol{\alpha}} \rangle \quad \text{subject to} \quad \mathcal{I}(\boldsymbol{U}).$$

Let $\boldsymbol{a}_i$ be an element in U_i for any $i \in L(\boldsymbol{U})$. The dual problem in a variable vector $\boldsymbol{x} \in \mathbb{R}^d$, where $d = \sum_{i \in L(\boldsymbol{U})} (r_i - 1)$, is written as

$$\mathrm{D}(\boldsymbol{U}): \left| \begin{array}{lll} \text{minimize} & \Phi(\boldsymbol{x}; \boldsymbol{U}) & \\ \text{subject to} & \Psi(\boldsymbol{x}; \boldsymbol{U}) = \boldsymbol{\gamma}, & \\ & -\infty < x_{\boldsymbol{b}} < +\infty & \boldsymbol{b} \in U_i \setminus \{\boldsymbol{a}_i\} \\ & x_{\boldsymbol{b}'} \geq 0 & \boldsymbol{b}' \in \mathcal{S}_i \setminus U_i \quad (i \in L(\boldsymbol{U})). \end{array} \right.$$

Here, the linear functions $\Phi(\boldsymbol{x}; \boldsymbol{U})$ and $\Psi(\boldsymbol{x}; \boldsymbol{U})$ in $\boldsymbol{x} \in \mathbb{R}^d$ are defined as follows:

$$\Phi(\boldsymbol{x}; \boldsymbol{U}) = \sum_{i \in L(\boldsymbol{U})} \sum_{\boldsymbol{a} \in \mathcal{S}_i \setminus \{\boldsymbol{a}_i\}} (\omega_i(\boldsymbol{a}) - \omega_i(\boldsymbol{a}_i))\, x_{\boldsymbol{a}}$$

$$\text{and} \quad \Psi(\boldsymbol{x}; \boldsymbol{U}) = \sum_{i \in L(\boldsymbol{U})} \sum_{\boldsymbol{a} \in \mathcal{S}_i \setminus \{\boldsymbol{a}_i\}} (\boldsymbol{a}_i - \boldsymbol{a})\, x_{\boldsymbol{a}},$$

where

$$\boldsymbol{x} = (x_{\boldsymbol{a}} : \boldsymbol{a} \in \mathcal{S}_i \setminus \{\boldsymbol{a}_i\},\ i \in L(\boldsymbol{U})) \in \mathbb{R}^d.$$

Any real vector $\boldsymbol{\gamma} \in \mathbb{R}^n$ can be chosen. If $\boldsymbol{\gamma}$ is fixed so that $\mathrm{D}(\boldsymbol{U})$ is feasible, the duality theorem holds for this primal-dual pair. $\mathrm{P}(\boldsymbol{U})$ is feasible if and only if $\mathrm{D}(\boldsymbol{U})$ is bounded below, and $\mathrm{P}(\boldsymbol{U})$ is infeasible if and only if $\mathrm{D}(\boldsymbol{U})$ is unbounded. Hence, the feasibility of $\mathrm{P}(\boldsymbol{U})$ can be revealed if the boundedness of $\mathrm{D}(\boldsymbol{U})$ is detected. The following describes how we set up $\boldsymbol{\gamma}$. Recall that $\boldsymbol{U} \in V_k$ is a node at the kth level in $T = (V, E)$ generated by Algorithm 2, and $\boldsymbol{C} \in V_0$ is the root node of T. We note that the feasible region of $\mathrm{D}(\boldsymbol{C})$ is included in that of $\mathrm{D}(\boldsymbol{U})$ for any $\boldsymbol{U} \in V_k$. Consider a right-hand vector $\boldsymbol{\gamma}$ of $\mathrm{D}(\boldsymbol{U})$ for any $\boldsymbol{U} \in V_k$ as

$$\tilde{\boldsymbol{\gamma}} = \Psi(\tilde{\boldsymbol{x}}; \boldsymbol{C})$$

using an arbitrary nonnegative vector $\tilde{\boldsymbol{x}} \in \mathbb{R}^d$. Then, $\mathrm{D}(\boldsymbol{U})$ becomes feasible for each $\boldsymbol{U} \in V_k$.

Furthermore, we can easily find an initial feasible solution for the dual problem $\mathrm{D}(\boldsymbol{U})$ for any $\boldsymbol{U} \in V_k$ by using an optimal solution of $\mathrm{D}(\boldsymbol{C})$. Recall that the root node $\boldsymbol{C} \in V_0$ was revealed to be feasible. That is, we solved $\mathrm{D}(\boldsymbol{C})$ and obtained an optimal solution $\boldsymbol{x}^* \in \mathbb{R}^d$ of $\mathrm{D}(\boldsymbol{C})$, where $d = \sum_{i \in L(\boldsymbol{C})}(r_i - 1)$. For any $\boldsymbol{U} \in V_k$, the vector

$$\begin{pmatrix} \boldsymbol{x}_* \\ \boldsymbol{0} \end{pmatrix} \in \mathbb{R}^{\bar{d}}, \text{ where } \bar{d} = \sum_{i \in L(\boldsymbol{U})} (r_i - 1). \tag{3.1}$$

becomes a feasible solution of $\mathrm{D}(\boldsymbol{U})$. In addition, if $\mathrm{D}(\boldsymbol{U})$ is bounded below for $\boldsymbol{U} \in V_k$ with $k < k_t + 1$, an optimal solution of $\mathrm{D}(\boldsymbol{U})$ is a feasible solution of $\mathrm{D}(\bar{\boldsymbol{U}})$ for any child node $\bar{\boldsymbol{U}} \in V_{k+1}$ of $\boldsymbol{U}$ since the feasible region of $\mathrm{D}(\boldsymbol{U})$ is included in that of $\mathrm{D}(\bar{\boldsymbol{U}})$. The simplex method is suitable for solving these dual problems with the common structure. Assume that a node $\boldsymbol{U} \in V_k$ $(k < k_t + 1)$ is feasible and generates the child nodes $\bar{\boldsymbol{U}} \in V_{k+1}$ of $\boldsymbol{U}$. The simplex method usually does not require many iterations for solving $\mathrm{D}(\bar{\boldsymbol{U}})$ when we reuse an optimal solution of $\mathrm{D}(\boldsymbol{U})$ as an initial solution. In terms of problem size, the dual problems are superior to the primal ones as stated in [19]. Therefore, we employ the dual problems to check the feasibility of a node and solve these problems using the simplex method.

3.3. Choice of $t \in M \setminus L(\boldsymbol{C})$ in (A). In Algorithm 1, we want to detect an index t from $M \setminus L(\boldsymbol{C})$ for a node $\boldsymbol{C} \in A_\ell$ with $\ell < m$, so that a large portion of the child nodes are infeasible. The best way to achieve this is to find an index t among $t \in M \setminus L(\boldsymbol{C})$ such that the size of $W^*(\boldsymbol{C}, t)$ attains the minimum size. Emiris and Canny's method [8] employs this strategy and selects the best support set from all candidates.

However, this approach might be unrealistic because it requires a large computation to construct $W^*(\boldsymbol{C},t)$ for every $t \in M \setminus L(\boldsymbol{C})$. In fact, the numerical results in [10, 18, 22] show that MVLP, in which Emiris and Canny's method is implemented, is slower than MixedVol [10], PHoM [22] and mvol [18]. Remember that each element in $W(\boldsymbol{C},t)$ is represented as the leaf nodes $\boldsymbol{U} \in V_{k_t}$ of a tree produced by Algorithm 2, and the leaf nodes are infeasible if the ancestor node of these leaf nodes is infeasible. To restrict the number of feasibility checks, we therefore focus on finding feasible nodes in V_1, which is the set of every node at the first level of the tree and is identical to $\Gamma(1;\boldsymbol{C},t)$. Obviously, the set

$$W_1^*(\boldsymbol{C},t) = \{\bar{\boldsymbol{C}} \in \Gamma(1;\boldsymbol{C},t) : \bar{\boldsymbol{C}} \text{ is feasible}\}$$

satisfies

$$W^*(\boldsymbol{C},t) \subseteq W_1^*(\boldsymbol{C},t) \subseteq W(\boldsymbol{C},t).$$

Using a feasible solution $\boldsymbol{x}_{init}$ of $\mathrm{D}(\boldsymbol{U})$ $(\boldsymbol{U} \in \Gamma(1;\boldsymbol{C},t))$ which can be easily obtained from an optimal solution of $\mathrm{D}(\boldsymbol{C})$, our method estimates the feasibility of each node in $\Gamma(1;\boldsymbol{C},t)$, and constructs $\hat{W}_1^*(\boldsymbol{C},t,\boldsymbol{x}_{init})$ satisfying

$$W_1^*(\boldsymbol{C},t) \subseteq \hat{W}_1^*(\boldsymbol{C},t,\boldsymbol{x}_{init}) \subseteq W(\boldsymbol{C},t).$$

The definition of $\hat{W}_1^*(\boldsymbol{C},t,\boldsymbol{x}_{init})$ is described as follows. To find feasible nodes $\boldsymbol{U}$ in $\Gamma(1;\boldsymbol{C},t)$, we deal with the dual problem $\mathrm{D}(\boldsymbol{U})$ instead of $\mathrm{P}(\boldsymbol{U})$, and check the boundedness of this problem. As stated in the previous subsection, we can obtain a feasible solution $\boldsymbol{x}_{init}$ of $\mathrm{D}(\boldsymbol{U})$ for any $\boldsymbol{U} \in \Gamma(1;\boldsymbol{C},t)$ by (3.1) with the use of an optimal solution $\boldsymbol{x}_*$ of $\mathrm{D}(\boldsymbol{C})$. From this initial solution, we start the pivoting process of the simplex method to check the boundedness of $\mathrm{D}(\boldsymbol{U})$. The simplex method easily solves $\mathrm{D}(\boldsymbol{U})$ with $\boldsymbol{x}_{init}$ as the initial solution, because the structure of these problems $\mathrm{D}(\boldsymbol{C})$ and $\mathrm{D}(\boldsymbol{U})$ are similar to each other. Indeed, in numerical experiments, we see that the simplex method requires only a few iterations for $\mathrm{D}(\boldsymbol{U})$ starting from $\boldsymbol{x}_{init}$. Accordingly, we can expect that a feasible solution $\boldsymbol{x}_{init}$ of $\mathrm{D}(\boldsymbol{U})$ has an *unbounded direction* when this problem is unbounded. Thereby, we test whether the feasible solution $\boldsymbol{x}_{init}$ of $\mathrm{D}(\boldsymbol{U})$ has an unbounded direction instead of solving this problem by the simplex method. At (A) of Algorithm 1, we construct

$$\hat{W}_1^*(\boldsymbol{C},t,\boldsymbol{x}_{init}) = \left\{ \bar{\boldsymbol{C}} \in \Gamma(1;\boldsymbol{C},t) : \begin{array}{l} \text{there is no unbounded direction} \\ \text{emanating from } \boldsymbol{x}_{init} \text{ in } \mathrm{D}(\bar{\boldsymbol{C}}) \end{array} \right\}$$

We need to explain the phrase "unbounded direction emanating from $\boldsymbol{x}_{init}$ in $\mathrm{D}(\bar{\boldsymbol{C}})$" used in the definition of $\hat{W}_1^*(\boldsymbol{C},t,\boldsymbol{x}_{init})$. Because a free variable x_b satisfying $-\infty < x_b < +\infty$ on $\mathrm{D}(\boldsymbol{U})$ can be represented by

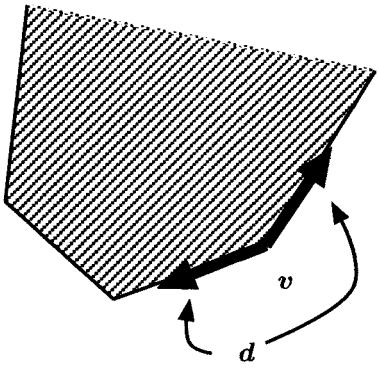


FIG. 2. *Direction vectors emanating from* $\boldsymbol{v}$.

$x_b = x'_b - x''_b$ using nonnegative variables x'_b, x''_b, for simplicity, we consider the standard LP problem

$$\begin{array}{ll} \text{minimize} & \langle \boldsymbol{c}, \boldsymbol{x} \rangle \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \quad \boldsymbol{x} \geq \boldsymbol{0} \end{array}$$

where a coefficient matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, cost vector $\boldsymbol{c} \in \mathbb{R}^n$ and constant vector $\boldsymbol{b} \in \mathbb{R}^m$ are given, and $\boldsymbol{x} \in \mathbb{R}^n$ is a variable vector. Suppose that this problem is feasible, *i.e.* the feasible region, which forms a polytope, is nonempty. Then, for a vertex $\boldsymbol{v}$ of the polytope, the simplex method computes an adjacent vertex $\bar{\boldsymbol{v}} = \boldsymbol{v} + \theta \boldsymbol{d}$ $(\theta \geq 0)$ of $\boldsymbol{v}$ using a direction vector $\boldsymbol{d} \in \mathbb{R}^n$, which emanates from $\boldsymbol{v}$. Fig. 2 illustrates the direction vectors incident to $\boldsymbol{v}$. A direction vector $\boldsymbol{d}$ emanating from $\boldsymbol{v}$ is said to be unbounded if we can increase the value of θ up to $+\infty$ while satisfying the following: (i) $\bar{\boldsymbol{v}} = \boldsymbol{v} + \theta \boldsymbol{d}$ satisfies all constraints, and (ii) the cost $\langle \boldsymbol{c}, \bar{\boldsymbol{v}} \rangle$ decreases from $\langle \boldsymbol{c}, \boldsymbol{v} \rangle$. If the LP problem is unbounded, it has an unbounded direction emanating from some vertex $\boldsymbol{v}$. In (A) of Algorithm 1, our dynamic enumeration method chooses an index t such that the size of $\hat{W}_1^*(\boldsymbol{C}, t, \boldsymbol{x}_{init})$ is the smallest among $t \in M \setminus L(\boldsymbol{C})$. Although we need a practical procedure to test whether $\boldsymbol{x}_{init}$ of $\mathrm{D}(\boldsymbol{U})$ has an unbounded direction, a detailed description can be found in [19, Section 3.2].

In (A) of Algorithm 1, our method uses $\hat{W}_1^*(\boldsymbol{C}, t, \boldsymbol{x}_{init})$ whereas Emiris and Canny's method uses $W^*(\boldsymbol{C}, t)$. There may be a difference between the sizes of $\hat{W}_1^*(\boldsymbol{C}, t, \boldsymbol{x}_{init})$ and $W^*(\boldsymbol{C}, t)$ for some $t \in M \setminus L(\boldsymbol{C})$. However, the numerical results in Section 5 show that our method sufficiently reduces the computational effort for finding all mixed cells in a fine mixed subdivision.

4. Usage of DEMiCs. After unpacking the software package DEMiCs, the user will see SRC and polySys, which include source and sample files, in the main directory. The make file exists in the directory SRC. The executable file "demics" is generated by executing the following command in the directory.

```
make all
```

The input file requires information regarding the support of a polynomial system: the dimension of the system, the number of the distinct support sets, the cardinality, multiplicity and elements of each support set. For example, consider the support sets for a semi-mixed system of type $(2,1,1)$ as follows:

$$\begin{aligned}
\mathcal{S}_1 := \quad \mathcal{A}_1 = \mathcal{A}_2 &= \left\{ \begin{array}{l} (1,0,0,0),\ (0,1,0,0),\ (0,0,1,0),\ (0,0,0,1), \\ (0,0,0,0),\ (1,1,1,1) \end{array} \right\} \\
\mathcal{S}_2 := \qquad \mathcal{A}_3 &= \left\{ \begin{array}{l} (2,0,0,0),\ (0,2,0,0),\ (0,0,2,0),\ (0,0,0,2), \\ (0,0,0,0),\ (2,2,2,2) \end{array} \right\} \\
\mathcal{S}_3 := \qquad \mathcal{A}_4 &= \left\{ \begin{array}{l} (3,0,0,0),\ (0,3,0,0),\ (0,0,3,0),\ (0,0,0,3), \\ (0,0,0,0),\ (3,3,3,3) \end{array} \right\}.
\end{aligned}$$

The input file for $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ is written in the following format:

```
# The dimension or the number of variables
Dim = 4

# The number of the distinct support sets
Support = 3

# The number of elements in each support set
Elem = 6 6 6

# The multiplicity of each support set
Type = 2 1 1


# The elements of the 1st support set
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
0 0 0 0
1 1 1 1

# The elements of the 2nd support set
2 0 0 0
0 2 0 0
0 0 2 0
0 0 0 2
0 0 0 0
2 2 2 2
```

```
# The elements of the 3rd support set
3 0 0 0
0 3 0 0
0 0 3 0
0 0 0 3
0 0 0 0
3 3 3 3
```

The directory polySys also contains some sample files describing the support sets of several benchmark polynomial systems.

The above input file is placed in SRC as "poly.dat". To compute the mixed volume via a fine mixed subdivision, we simply execute

```
demics  poly.dat
```

in SRC, in which the executable file "demics" and the input file "poly.dat" exist. The software package then displays the total number of mixed cells, the value of the mixed volume and cpu time on the screen.

```
# Mixed Cells: 4
Mixed Volume: 24

CPU time: 0 s
```

Furthermore, we can select three options "-c", "-s" and "-cs" when running the program. The option "-c" offers information about each mixed cell $\boldsymbol{C} = (C_i : i \in M) \in \Omega(M)$. After executing the command

```
demics -c poly.dat
```

the following information is displayed on the screen

```
# 1 : 1 : ( 1 2 6 ) 2 : ( 1 5 ) 3 : ( 5 3 )
Volume: 6

# 2 : 1 : ( 4 1 6 ) 2 : ( 1 5 ) 3 : ( 3 5 )
Volume: 6

# 3 : 1 : ( 4 2 6 ) 3 : ( 3 4 ) 2 : ( 6 5 )
Volume: 6

# 4 : 1 : ( 4 2 6 ) 3 : ( 4 5 ) 2 : ( 5 1 )
Volume: 6

# Mixed Cells: 4
```

```
Mixed Volume: 24

CPU time: 0.01 s
```

On the first line with "# 1", "1 : (1 2 6)" means the subset $C_1 = \{\boldsymbol{a}_{11}, \boldsymbol{a}_{12}, \boldsymbol{a}_{16}\}$ of $\mathcal{S}_1$. That is, the number in front of a colon corresponds to the index of the support set. We thus know that one of mixed cells $\boldsymbol{C} = (C_1, C_2, C_3) \in \mathcal{S}_1 \times \mathcal{S}_2 \times \mathcal{S}_3$ consists of

$$C_1 = \{\boldsymbol{a}_{11}, \boldsymbol{a}_{12}, \boldsymbol{a}_{16}\}, \quad C_2 = \{\boldsymbol{a}_{21}, \boldsymbol{a}_{25}\} \quad \text{and} \quad C_3 = \{\boldsymbol{a}_{35}, \boldsymbol{a}_{33}\}$$

where $\boldsymbol{a}_{ij}$ is an element of $\mathcal{S}_i$. "Volume" on the next line represents the volume of the mixed cell $\boldsymbol{C} = (C_1, C_2, C_3)$. Note that the mixed volume is obtained from the summation of volumes of four mixed cells for the specific lifting values $\omega_i(\boldsymbol{a})$ $(\boldsymbol{a} \in \mathcal{S}_i)$. On a line with "#", the sequence of indices i for the subset C_i of each support set $\mathcal{S}_i$ indicates the order of an index t chosen from $M \setminus L(\boldsymbol{C})$ in Algorithm 1. For example, the line with "# 3" shows that the support sets $\mathcal{S}_1, \mathcal{S}_3$ and $\mathcal{S}_2$ are chosen in this order.

The software package needs a seed number to generate a random number for each lifting value $\omega_i(\boldsymbol{a})$ $(\boldsymbol{a} \in \mathcal{S}_i)$. If no option is selected as stated in the above, the seed number is automatically set to "1". The option "-s" is useful in the case where we change the seed number to generate different lifting values for each execution. As an example, when "6" is chosen as the seed number for the input data "poly.dat", we type the command

```
demics -s poly.dat 6
```

Given the seed number "2", the "-cs" option is used as follows to get detailed information about mixed cells.

```
demics -cs poly.dat 2
```

5. Numerical results. This software package has been tested on a large variety of polynomial systems including unmixed, semi-mixed and fully mixed types. The papers [10, 12] report the superiority of MixedVol in computational time for these three types of the systems over the existing software packages: HOM4PS [11], MVLP [8], PHCpack [25], PHoM [22] and mvol [18]. Therefore, we compare DEMiCs with MixedVol for each type of polynomial systems in terms of the computational time. Note that MixedVol employs the static enumeration method, while DEMiCs adopts the dynamic enumeration method. All numerical experiments were executed on a 2.4GHz Opteron 850 with 8GB memory, running Linux.

First, we observe how the computational time of DEMiCs and MixedVol varies depending on m, which is the number of distinct support sets $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m$ of the semi-mixed polynomial systems $\boldsymbol{f}(\boldsymbol{x})$ in $\boldsymbol{x} \in \mathbb{R}^n$. In

numerical experiments, we deal with artificial semi-mixed systems, which are created to investigate the feature of DEMiCs. Each support set $\mathcal{S}_i$ of the semi-mixed systems is given as follows. We choose the subset $\mathcal{T}$ of $\mathbb{Z}_+^n$ with $\#\mathcal{T} = 2n$ such as

$$\mathcal{T} = \left\{ \begin{array}{l} (1,0,\ldots,0,0),\ (0,1,\ldots,0,0),\cdots,(0,0,\ldots,1,0),(0,0,\ldots,0,0) \\ (1,0,\ldots,0,1),\ (0,1,\ldots,0,1),\cdots,(0,0,\ldots,1,1),(0,0,\ldots,0,1) \end{array} \right\}.$$

Note that the convex hull of $\mathcal{T}$ is the n-dimensional prism with a simplex basis, and the n-dimensional volume is $\frac{1}{(n-1)!}$. Let $\boldsymbol{e}_i$ denote the n-dimensional ith unit vector. For $\mathcal{T} \subseteq \mathbb{Z}_+^n$, we consider the transition of $\mathcal{T}$ by the direction vector $\boldsymbol{e}_i$ as $\mathcal{S}_i$. Namely,

$$\mathcal{S}_i := \boldsymbol{e}_i + \mathcal{T} = \{\boldsymbol{a} + \boldsymbol{e}_i : \boldsymbol{a} \in \mathcal{T}\} \quad \text{for each } i \in M.$$

Assume that each support set $\mathcal{S}_i$ has the multiplicity $n/m \in \mathbb{Z}$ for the dimension n and the number of distinct support sets m. That is, we deal with semi-mixed polynomial systems $\boldsymbol{f}(\boldsymbol{x})$ of type $(n/m, n/m, \ldots, n/m)$. Here, the mixed volume for the support $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m)$ of $\boldsymbol{f}(\boldsymbol{x})$ is calculated as $n! \times \frac{1}{(n-1)!} = n$ because the n-dimensional volume of $\mathrm{conv}(\mathcal{S}_i)$ and $\mathrm{conv}(\mathcal{T})$ is equal to each other.

To demonstrate the performance of DEMiCs and MixedVol, we choose two different values $n = 18, 24$ for the dimension of elements in $\mathcal{S}_i \subseteq \mathbb{Z}_+^n$, and change the number of distinct supports sets m in response to each dimension n. Table 1 and 2, which are in the case of $n = 18, 24$ respectively, summarize the cpu time of DEMiCs and MixedVol for each system. We performed 10 times numerical experiments for each system by choosing the different lifting values in DEMiCs and MixedVol. The cpu time listed in Table 1 and 2 is the average for each trial. The column "#Supp." means the number of distinct support sets m, and "Ratio" indicates the ratio between the cpu time of DEMiCs and MixedVol. The symbol "-" means that the software package has not been applied to the corresponding system. From these tables, we see that DEMiCs is superior to MixedVol in the computational time if the number of the distinct support sets is large. To the contrary, if the number of the distinct support sets is small, we may not expect the advantage of a dynamic enumeration method. One of the main reasons is that there is not great difference between the structure of the dynamic and static enumeration trees. Moreover, DEMiCs needs more computational tasks involved in choosing an index t from $M \setminus L(\boldsymbol{C})$ at Algorithm 1. Therefore, DEMiCs takes more computational time than MixedVol for semi-mixed systems with a few distinct supports.

Second, we consider the following benchmark polynomial systems, including unmixed, semi-mixed and fully mixed systems. The PRS-10 and RRS-12 systems, which are arising from kinematic problems in [21], have 12 polynomials with 12 variables and 11 polynomials with 11 variables, respectively. The PRS-10 system is a semi-mixed system of type $(1, 1, 1, 9)$, and

TABLE 1
$n = 18$ *(The mixed volume of all systems is 18).*

# Supp. (m)	DEMiCs	MixedVol	Ratio
$m = 1$	0.052s	0.045s	0.87
$m = 3$	8.893s	4.969s	0.56
$m = 6$	8.715s	52.482s	6.02
$m = 9$	15.453s	3m40.422s	14.26
$m = 18$	1m7.927s	1h13m36.590s	65.02

TABLE 2
$n = 24$ *(The mixed volume of all systems is 24).*

# Supp. (m)	DEMiCs	MixedVol	Ratio
$m = 1$	0.599s	0.341s	0.57
$m = 3$	11m42.896s	5m32.361s	0.47
$m = 6$	4m20.044s	1h21m13.110s	18.74
$m = 8$	4m31.044s	4h3m41.700s	53.95
$m = 12$	9m9.827s	23h43m40.700s	155.36
$m = 24$	1h40m11.080s	-	

the first three support sets have 4 elements, the last 100 elements. Also, the RRS-12 system is an unmixed system of type (11), and the support set has 224 elements. The cyclic-n[3], chandra-n[5] and katsura-n[4] systems are fully mixed systems, and size-expandable systems by the number n. The katsura-n systems consist of $(n+1)$ polynomials with $(n+1)$ variables, and the others n polynomials with n variables. The detailed description of the systems can be found in the web site [23]. We changed the lifting values 10 times for each system, and executed numerical experiments. In Table 3, we list the comparison of the average cpu time of DEMiCs and MixedVol. The column "$\mathcal{MV}$" presents the mixed volume for the support of corresponding systems, and "Ratio" is the ratio between the cpu time of these software packages. The numerical results for the cyclic-n, chandra-n and katsura-n systems in Table 3 show that DEMiCs improves the cpu time for finding all mixed cells dramatically when we address the polynomial systems with many distinct support sets. However, the numerical results on the PRS-10 and RRS-12 systems imply that it may be difficult for DEMiCs to deal with the unmixed and semi-mixed system which has only a few distinct support sets, compared with MixedVol.

Finally, we consider the large-scale cyclic-n, chandra-n and katsura-n polynomial systems. Numerical experiments were carried out 5 times for each system associated with the different lifting values. Table 4 exhibits the

TABLE 3
CPU time for the benchmark systems.

System	Size (n)	$\mathcal{MV}$	DEMiCs	MixedVol	Ratio
PRS-10		142,814	14.3s	4.0s	0.28
RRS-12		226,512	29.9s	0.7s	0.02
Cyclic-n	$n = 12$	500,352	1m11.6s	4m43.0s	3.95
	$n = 13$	2,704,156	11m0.3s	49m57.4s	4.54
	$n = 14$	8,795,976	1h27m27.3s	7h14m24.1s	4.97
Chandra-n	$n = 17$	65,536	1m9.4s	33m13.4s	28.70
	$n = 18$	131,072	3m10.5s	2h14m15.3s	42.29
	$n = 19$	262,144	9m21.1s	8h19m6.3s	53.38
Katsura-n	$n = 12$	4,096	46.9s	14m3.5s	17.98
	$n = 13$	8,192	5m8.1s	1h21m19.4s	15.84
	$n = 14$	16,384	24m17.2s	7h54m29.4s	19.54

TABLE 4
A large size of cyclic-n, chandra-n and katsura-n systems.

System	Size (n)	$\mathcal{MV}$	CPU time
Cyclic-n	$n = 15$	35,243,520	13h33m53.8s
	$n = 16$	135,555,072	110h21m40.5s
Chandra-n	$n = 20$	524,288	29m50.8s
	$n = 21$	1,048,576	1h15m19.7s
	$n = 22$	2,097,152	3h5m48.2s
	$n = 23$	4,194,304	11h24m4.6s
	$n = 24$	8,388,608	30h1m33.6s
Katsura-n	$n = 15$	32,768	1h30m54.7s
	$n = 16$	65,536	9h49m20.8s
	$n = 17$	131,072	46h37m35.8s

average cpu time of DEMiCs for each system. As compared with numerical results in Table 2 of [19], the computational time of DEMiCs is less than that of the program developed in [19] as the size of the systems becomes larger. It could be due to improvement on how to use memory space in DEMiCs.

6. Concluding remarks. In this paper, we introduced the software package DEMiCs. Using the dynamic enumeration method, this package computes the mixed volume, which can be obtained from the volumes of all mixed cells, for the support of a semi-mixed polynomial system. The dynamic enumeration method, which was developed for a fully-mixed type in [19], can be extended to a semi-mixed type naturally. Numerical results show that this package significantly is faster than existing ones for large-scale semi-mixed systems with many distinct support sets. We confirm that it is important to take account of how we construct an enumeration tree for finding mixed cells. From numerical results, we recognize that DEMiCs needs more computational time than MixedVol for an unmixed system and a semi-mixed system with a few distinct support sets. It appears that the dynamic enumeration method does not have a beneficial effect on such systems, because the structure of the dynamic tree is almost the same as that of the static tree. Although we proposed one strategy for constructing an enumeration tree dynamically, there may exist other strategies, which can improve our dynamic enumeration. This is our future work. Finding mixed cells plays a crucial role in the polyhedral homotopy method. We expect that DEMiCs opens the way for computing all isolated zeros of large-scale polynomial systems by polyhedral homotopy method.

Acknowledgments. The author is grateful to the anonymous referees for their valuable comments.

REFERENCES

[1] D.N. BERNSHTEIN, *The number of roots of a system of equations*, Funct. Anal. Appl. **9** (1975), pp. 183–185.

[2] U. BETKE, *Mixed volumes of polytopes*, Archiv der Mathematik **58** (1992), pp. 388–391.

[3] G. BJÖRK AND R.FRÖBERG, *A faster way to count the solutions of inhomogeneous systems of algebraic equations*, J. Symbolic Comput. **12**(3) (1991), pp. 329–336.

[4] W. BOEGE, R. GEBAUER, AND H. KREDEL, *Some examples for solving systems of algebraic equations by calculating Groebner bases*, J. Symbolic Comput. **2**(1) (1986), pp. 83–98.

[5] S. CHANDRASEKHAR, *Radiative Transfer*, Dover, NY, 1960.

[6] D.A. COX, J. LITTLE AND D. O'SHEA, *Using Algebraic Geometry*, Springer-Verlag, New York, 2nd edition, 2004.

[7] Y. DAI, S. KIM, AND M. KOJIMA, *Computing all nonsingular solutions of cyclic-n polynomial using polyhedral homotopy continuation methods*, J. Comput. Appl. Math. **152** (2003), pp. 83–97.

[8] I.Z. EMIRIS AND J.F. CANNY, *Efficient incremental algorithms for the sparse resultant and the mixed volume*, J. Symbolic Comput. **20**(2) (1995), pp. 117–149. Software available at http://cgi.di.uoa.gr/~emiris/index-eng.html.

[9] T. GAO AND T.Y. LI, *Mixed volume computation via linear programming*, Taiwanese J. Math. **4** (2000), pp. 599–619.

[10] ———, *Mixed volume computation for semi-mixed systems*, Discrete and Comput. Geom. **29**(2) (2003), pp. 257–277.

[11] ———, The software package HOM4PS is available at http://www.mth.msu.edu/~li/.

[12] T. GAO, T.Y. LI, AND M. WU, *Algorithm 846: MixedVol: A Software Package for Mixed Volume Computation*, ACM Trans. Math. Software **31**(4) (2005), pp. 555 – 560. Software available at http://www.csulb.edu/~tgao/.

[13] T. GUNJI, S. KIM, M. KOJIMA, A. TAKEDA, K. FUJISAWA, AND T. MIZUTANI, *PHoM – a Polyhedral Homotopy Continuation Method.* Computing **73**(1) (2004), pp. 57–77.

[14] T. GUNJI, S. KIM, K. FUJISAWA, AND M. KOJIMA, *PHoMpara – Parallel implementation of the polyhedral homotopy continuation method*, Computing **77**(4) (2006), pp. 387–411.

[15] B. HUBER AND B. STURMFELS, *A Polyhedral method for solving sparse polynomial systems*, Math. Comp. **64** (1995), pp. 1541–1555.

[16] S. KIM AND M. KOJIMA, *Numerical Stability of Path Tracing in Polyhedral Homotopy Continuation Methods*, Computing **73** (2004), pp. 329–348.

[17] T.Y. LI, *Solving polynomial systems by polyhedral homotopies*, Taiwanese J. Math. **3** (1999), pp. 251–279.

[18] T.Y. LI AND X. LI, *Finding Mixed Cells in the Mixed Volume Computation*, Found. Comput. Math. **1** (2001), pp. 161–181. Software available at http://www.math.msu.edu/~li/.

[19] T. MIZUTANI, A. TAKEDA, AND M. KOJIMA, *Dynamic Enumeration of All Mixed Cells*, Discrete Comput. Geom. **37**(3), (2007), pp. 351–367.

[20] A. MORGAN, *Solving polynomial systems using continuation for engineering and scientific problems*, Pentice-Hall, New Jersey, 1987.

[21] H.-J. SU, J.M. MCCARTHY, AND L.T. WATSON, *Generalized Linear Product Homotopy Algorithms and the Computation of Reachable Surfaces*, ASME J. Comput. Inf. Sci. Eng. **4**(3) (2004), pp. 226–234.

[22] A. TAKEDA, M. KOJIMA, AND K. FUJISAWA, *Enumeration of all solutions of a combinatorial linear inequality system arising from the polyhedral homotopy continuation method*, J. Oper. Soc., Japan **45** (2002), pp. 64–82. Software available at http://www.is.titech.ac.jp/~kojima/index.html.

[23] J. VERSCHELDE, The database of polynomial systems is in his web site: http://www.math.uic.edu/~jan/.

[24] J. VERSCHELDE, P. VERLINDEN, AND R. COOLS, *Homotopies exploiting Newton polytopes for solving sparse polynomial systems*, SIAM J. Numer. Anal. **31** (1994), 915–930.

[25] J. VERSCHELDE, *Algorithm 795: PHCPACK: A general-purpose solver for polynomial systems by homotopy continuation*, ACM Trans. Math. Softw. **25** (1999), pp. 251–276. Software available at http://www.math.uic.edu/~jan/.

SYNAPS: A LIBRARY FOR DEDICATED APPLICATIONS IN SYMBOLIC NUMERIC COMPUTING

BERNARD MOURRAIN*, JEAN-PASCAL PAVONE*,
PHILIPPE TREBUCHET†, ELIAS P. TSIGARIDAS‡, AND JULIEN WINTZ*

Abstract. We present an overview of the open source library SYNAPS. We describe some of the representative algorithms of the library and illustrate them on some explicit computations, such as solving polynomials and computing geometric information on implicit curves and surfaces. Moreover, we describe the design and the techniques we have developed in order to handle a hierarchy of generic and specialized data-structures and routines, based on a view mechanism. This allows us to construct dedicated plugins, which can be loaded easily in an external tool. Finally, we show how the design of the library allows us to embed the algebraic operations, as a dedicated plugin, into the external geometric modeler AXEL.

Key words. Symbolic-numeric computation, software.

AMS(MOS) subject classifications. Primary 68W30, 68W25, 65D17.

The aim of this paper is to give an overview of the software library SYNAPS[1]. It is an open source project, the objective of which is to provide a coherent and efficient library for symbolic and numeric computations. It implements data-structures and classes for manipulating basic algebraic objects, such as (dense, sparse, structured) vectors, matrices, univariate and multivariate polynomials. It also provides fundamental methods such as algebraic number manipulation tools, different types of univariate and multivariate polynomial root solvers, resultant and gcd computations, etc. The main motivation behind this project, is the need to combine symbolic and numeric computations, which is ubiquitous in many problems. Starting with an exact description of the equations, in most cases, we will eventually have to compute an approximation of the solutions. Even more, in many problems, the coefficients of the equations may only be known with some inaccuracy (due, for instance, to measurement errors). In these cases, we are not dealing with a solely system but with a neighborhood of an exact system and we have to take into account the continuity of the solutions with respect to the input coefficients. This leads to new, interesting and challenging questions both from a theoretical and a practical point of view, that lie in the frontier between Algebra and Analysis and witnesses the emergence of new investigations. In order to develop efficient implementations for such problems we have to combine algorithms from numeric and symbolic computation and to develop and manipulate data structures

*GALAAD, Inria, BP 93, 06902 Sophia-Antipolis, France.

†SPIRAL, LIP6, 8 rue du capitaine Scott 75015 Paris, France.

‡Department of Informatics & Telecommunications, National Kapodistrian University of Athens, Panepistimiopolis 15784, Greece.

[1] `http://synaps.inria.fr/`

that are on one hand generic and on the other are easily tuned to specific problems. Moreover, the reusability of external or third-party libraries, such as LAPACK (Fortran library for numerical linear algebra), GMP (C library for extended arithmetic) has to be considered carefully. Specialized routines provided by these external tools have to coexist with generic implementation. Therefore, the software should be designed so that it can connect, in an automatic and invisible to end-user way, the appropriate implementation with the needed operation.

In this paper, we first describe representative algorithms available in the library, and illustrate them by some explicit computations. We begin with a description of the solvers of polynomial equations. These tools are used as black boxes in geometric computations on implicit curves and surfaces. We show how the first level of data structures and polynomial solving implementations are composed to build such algorithms. Such higher level operations on geometric objects are embedded in the geometric modeler AXEL[2], as a dedicated plugin. We describe the design and techniques we have developed to handle a hierarchy of generic and specialized implementations, based on a view mechanism. This approach is extended to build plugins, which provide the equivalent functions in an interactive environment. In particular, we show how template mechanisms can be exploited to transform static strongly typed code into dynamic polymorphic and generic functions, assembled into a plugin that can be loaded easily in an external tool.

1. Solvers of polynomial equations. A critical operation, which we will have to perform in geometric computations on curves and surfaces, is to solve polynomial equations. In such a computation, we start with input polynomial equations (possibly with some incertitude on the coefficient) and we want to compute an approximation of the (real) roots of these equations or boxes containing these roots. Such operation should be performed very efficiently and with guarantee, since they will be used intensively in geometric computation.

In sections 1.1, 1.2, 1.3, we describe subdivision solvers which are based on *certified* exclusion criteria. In other words, starting from an initial bounded domain, we remove subdomains which are guaranteed not to contain a real solution of the polynomial equations. A parameter $\epsilon > 0$ is controlling the size of the boxes that are kept. For univariate polynomials, existence and uniqueness criteria are applied to produce certified isolation intervals which contain a single root. Such criteria also exist in the multivariate case, but are not yet available in our current implementation. The interest of these subdivision methods, compared to homotopy solvers [34], [15] or algebraic solvers [13], [33] is that only local information related to the initial domain are used and it avoids the representation or approximation of all the complex roots of the system. The methods

[2]http://axel.inria.fr/

are particularly efficient for systems where the number of real roots is much smaller that the number of complex roots or where the complex roots are far from the domain of interest. However multiple roots usually reduces their performance if their isolation is required, in addition to their approximation.

1.1. Univariate subdivision solvers. Let us consider first an exact polynomial $f = \sum_{i=0}^{d} a_i x^i \in \mathbb{Q}[x]$. Our objective is to isolate the real roots of f, i.e. to compute intervals with rational endpoints that contain one and only one root of f, as well as the multiplicity of every real root. The algorithms take these exact input polynomials and output certified isolation intervals of their real roots. Some parts of the computation are performed with exact integer or rational arithmetic (using the library GMP), but some other parts might be implemented using floating point arithmetic. It uses adapted rounding modes, to be able to certify the result. Here is the general scheme of the subdivision solver that we consider, augmented appropriately so that it also outputs the multiplicities. It uses an external function $V(f, I)$, which bounds the number of roots of f in the interval I.

ALGORITHM 1.1. REAL ROOT ISOLATION

INPUT: A polynomial $f \in \mathbb{Z}[x]$, such that $\deg(f) = d$ and $\mathcal{L}(f) = \tau$.
OUTPUT: A list of intervals with rational endpoints, which contain one and only one real root of f and the multiplicity of every real root.

1. Compute the square-free part of f, i.e. f_{red}
2. Compute an interval $I_0 = (-B, B)$ with rational endpoints that contains all the real roots. Initialize a queue Q with I_0.
3. While Q is not empty do
   a) Pop an interval I from Q and compute $v := V(f, I)$.
   b) If $v = 0$, discard I.
   c) If $v = 1$, output I.
   d) If $v \geq 2$, split I into I_L and I_R and push them to Q.
4. Determine the multiplicities of the real roots, using the square-free factorization of f.

Two families of solvers have been developed. One using Sturm theorem, where $V(f, I)$ returns the exact number (counted without multiplicities) of the real roots of f in I. The second one based on Descartes' rule and Bernstein representation, where $V(f, I)$ bounds the number of real roots of f in I (counted with multiplicities). As analyzed in [10], the bit complexity of both approaches is in $\widetilde{\mathcal{O}}_B(d^4\tau^2)$, if $f \in \mathbb{Z}[x]$, $\deg(f) = d$ is the degree of f and $\mathcal{L}(f) = \tau$ the maximal bitsize of its coefficients. Notice that with the same complexity bound, we can also compute the multiplicities of the real roots. However in practice, the behavior is not exactly the same, as we will illustrate it.

1.1.1. Sturm subdivision solver. We recall here the main ingredients related to Sturm(-Habicht) sequence computations.

Let $f = \sum_{k=0}^{p} f_k x^k, g = \sum_{k=0}^{q} g_k x^k \in \mathbb{Z}[x]$ where $\deg(f) = p \geq q = \deg(g)$ and $\mathcal{L}(f) = \mathcal{L}(g) = \tau$. We denote by $\mathtt{rem}(f,g)$ and $\mathtt{quo}(f,g)$ the remainder and the quotient, respectively, of the Euclidean division of f by g, in $\mathbb{Q}[x]$.

DEFINITION 1.2. [35, 3] *The* signed polynomial remainder sequence *of f and g, denoted by* $\mathbf{SPRS}(f,g)$, *is the polynomial sequence*

$$R_0 = f, R_1 = g, R_2 = -\mathtt{rem}(f,g), \ldots, R_k = -\mathtt{rem}(R_{k-2}, R_{k-1}),$$

with k the minimum index such that $\mathtt{rem}(R_{k-1}, R_k) = 0$. *The* quotient sequence *of f and g is the polynomial sequence $\{Q_i\}_{0 \leq i \leq k-1}$, where $Q_i =$* $\mathtt{quo}(R_i, R_{i+1})$ *and the* quotient boot *is $(Q_0, Q_1, \ldots, Q_{k-1}, R_k)$.*

Another construction yields the Sturm-Habicht sequence of f and g, i.e. $\mathbf{StHa}(f,g)$, which achieves better bounds on the bit size of the coefficients.

Let M_j be the matrix which has as rows the coefficient vectors of the polynomials $fx^{q-1-j}, fx^{q-2-j}, \ldots, fx, f, g, gx, \ldots, gx^{p-2-j}, gx^{p-1-j}$ with respect to the monomial basis $x^{p+q-1-j}, x^{p+q-2-j}, \ldots, x, 1$. The dimension of M_j is $(p+q-2j) \times (p+q-j)$. For $l = 0, \ldots, p+q-1-j$ let M_j^l be the square matrix of dimension $(p+q-2j) \times (p+q-2j)$ obtained by taking the first $p+q-1-2j$ columns and the $l+(p+q-2j)$ column of M_j.

DEFINITION 1.3. *The Sturm-Habicht sequence of f and g, is the sequence* $\mathbf{StHa}(f,g) = (H_p = H_p(f,g), \ldots, H_0 = H_0(f,g))$, *where* $H_p = f$, $H_{p-1} = g$, $H_j = (-1)^{(p+q-j)(p+q-j-1)/2} \sum_{l=0}^{j} \det(M_j^l) x^l$.

For two polynomials of degree p and q and of bit size bounded by τ, such sequences and their evaluation at a rational point $\mathtt{a}$, where $\mathtt{a} \in \mathbb{Q} \cup \{\pm\infty\}$ and $\mathcal{L}(\mathtt{a}) = \sigma$ can be done respectively with complexity $\widetilde{\mathcal{O}}_B(p^2 q\tau)$ and $\widetilde{\mathcal{O}}_B(q \max\{p\tau, q\sigma\})$. For more details, see [35, 3, 18, 19].

The structure `SturmSeq` encodes these Sturm sequences in SYNAPS. Several constructions are implemented, specified by a class in the constructor, Euclidean, primitive and subresultant polynomial remainder sequences. Let us present an example of code for constructing the Sturm-Habicht sequence `s` of two polynomials `p`, `q` $\in \mathbb{Z}[x]$. The implementation corresponds to a variant of the inductive construction described in [3].

```
UPolDse<ZZ> p("3*x^5+23*x^3-x^2+234"), q("10*x^4+200*x^2-13243");
SturmSeq<ZZ> s(p,q,HABICHT());
```

The result is a sequence of polynomials, with coefficients in the initial ring ZZ:

```
[3*x^5+23*x^3-x^2+234, 10*x^4+200*x^2-13243,
   3700*x^3+100*x^2-397290*x-23400, -174378300*x^2-4685100*x+1813200700,
   796993090279590*x+51961697166600, -37867420670503735668763]
```

Such a sequence can be used to count roots in an interval. Let $W_{(f,g)}(\mathtt{a})$ denote the number of modified sign changes of the evaluation of $\mathbf{StHa}(f,g)$ at $\mathtt{a}$.

THEOREM 1.1. [3, 36] *Let $f, g \in \mathbb{Z}[x]$, where f is square-free and f' is the derivative of f and its leading coefficient $f_d > 0$. If $\mathsf{a} < \mathsf{b}$ are both non-roots of f and γ ranges over the roots of f in (a, b), then $W_{(f,g)}(\mathsf{a}) - W_{(f,g)}(\mathsf{b}) = \sum_\gamma \operatorname{sign}(f'(\gamma)g(\gamma))$.* If $g = f'$ then $\mathbf{StHa}(f, f')$ is a Sturm sequence and Th. 1.1 counts the number of distinct real roots of f in (a, b). For the Sturm solver $V(f, [\mathsf{a}, \mathsf{b}])$ will denote $V(f, [\mathsf{a}, \mathsf{b}]) = W_{(f,f')}(\mathsf{a}) - W_{(f,f')}(\mathsf{b})$.

1.1.2. Bernstein subdivision solver. In this section, we recall the background of Bernstein polynomial representation and how it is used in the subdivision solver. Given an arbitrary univariate polynomial function $f(x) \in \mathbb{K}$, we can convert it to a representation of degree d in Bernstein basis, which is defined by:

$$f(x) = \sum_i b_i B_i^d(x), \text{ and } B_i^d(x) = \binom{d}{i} x^i (1-x)^{d-i} \tag{1.1}$$

where b_i is usually referred as controlling coefficients. Such conversion is done through a basis conversion [11]. The above formula can be generalized to an arbitrary interval $[a, b]$ by a variable substitution $x' = (b-a)x + a$. We denote by $B_d^i(x; a, b) = \binom{d}{i}(x-a)^i(b-x)^{d-i}(b-a)^{-d}$ the corresponding Bernstein basis on $[a, b]$. There are several useful properties regarding Bernstein basis given as follows:

- *Convex-Hull Properties*: Since $\sum_i B_d^i(x; a, b) \equiv 1$ and $\forall x \in [a, b]$, $B_d^i(x; a, b) \geq 0$ where $i = 0, ..., d$, the graph of $f(x) = 0$, which is given by $(x, f(x))$, should always lie within the convex-hull defined by the control coefficients $(\frac{i}{d}, b_i)$ [11].
- *Subdivision* (de Casteljau): Given $t_0 \in [0, 1]$, $f(x)$ can be represented piece-wisely by:

$$f(x) = \sum_{i=0}^{d} b_0^{(i)} B_d^i(x; a, c) = \sum_{i=0}^{d} b_i^{(d-i)} B_d^i(x; c, b), \text{ where} \tag{1.2}$$

$$b_i^{(k)} = (1-t_0) b_i^{(k-1)} + t_0 b_{i+1}^{(k-1)} \text{ and } c = (1-t_0)a + t_0 b. \tag{1.3}$$

Another interesting property of this representation related to Descartes rule of signs is that there is a simple and yet efficient test for the existence of real roots in a given interval. It is based on the number of sign variation $V(\mathbf{b}_k)$ of the sequence $\mathbf{b}_k = [b_1, \ldots, b_k]$ that we define recursively as follows:

$$V(\mathbf{b}_{k+1}) = V(\mathbf{b}_k) + \begin{cases} 1, \text{if } b_k b_{k+1} < 0 \\ 0, \text{else.} \end{cases} \tag{1.4}$$

With this definition, we have:

PROPOSITION 1.1. *Given a polynomial* $f(x) = \sum_i^n b_i B_i^d(x; a, b)$, *the number* N *of real roots of* f *on* $]a, b[$ *is less than or equal to* $V(\boldsymbol{b})$, *where* $\boldsymbol{b} = (b_i)_{i=1,\ldots,n}$ *and* $N \equiv V(\boldsymbol{b}) mod\ 2$. With this proposition,

- if $V(\mathbf{b}) = 0$, the number of real roots of f in $[a, b]$ is 0;
- if $V(\mathbf{b}) = 1$, the number of real roots of f in $[a, b]$ is 1.

This function V yields another variant of subdivision algorithm 1.1. In order to analyze its behavior, a partial inverse of Descartes' rule and lower bounds on the distance between roots of a polynomial have been used. It is proved that the complexity of isolating the roots of a polynomial of degree d, with integer coefficients of bit size $\leq \tau$ is bounded by $\mathcal{O}\left(d^4\tau^2\right)$ up to some poly-logarithmic factors. See [8, 10] for more details.

Notice that this localization algorithm extends naturally to B-splines, which are piecewise polynomial functions [11].

The approach can also be extended to polynomials with interval coefficients, by counting 1 sign variation for a sign sub-sequence $+, ?, -$ or $-, ?, +$; 2 sign variations for a sign sub-sequence $+, ?, +$ or $-, ?, -$; 1 sign variation for a sign sub-sequence $?, ?$, where $?$ is the sign of an interval containing 0. Again in this case, if a family $\underline{\bar{\mathrm{f}}}$ of polynomials is represented by the sequence of intervals $\underline{\bar{\mathbf{b}}} = [\underline{\bar{\mathrm{b}}}_0, \ldots, \underline{\bar{\mathrm{b}}}_d]$ in the Bernstein basis of the interval $[a, b]$

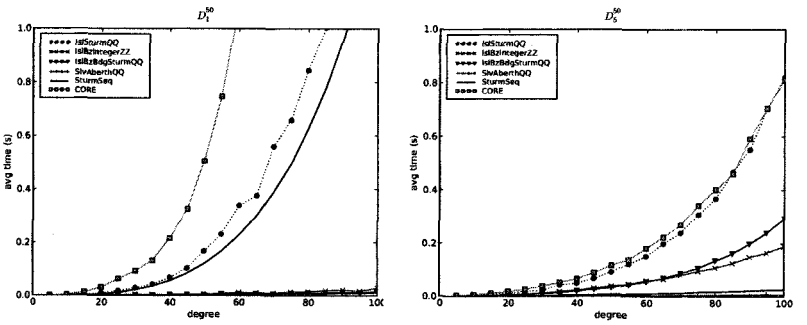
- if $V(\underline{\bar{\mathbf{b}}}) = 1$, all the polynomials of the family $\underline{\bar{\mathrm{f}}}$ have one root in $[a, b]$,
- if $V(\underline{\bar{\mathbf{b}}}) = 0$, all the polynomials of the family $\underline{\bar{\mathrm{f}}}$ have no roots in $[a, b]$.

This subdivision algorithm, using interval arithmetic, yields either intervals of size smaller than ϵ, which might contain the roots of $f = 0$ in $[a, b]$ or isolating intervals for all the polynomials of the family defined by the interval coefficients.

A variant of such approach in the monomial basis is called Uspensky's method, e.g. [8, 29] and references therein. Another variant, using Cauchy's lower bound on the positive roots of the polynomial, isolates the real roots by computing their continued fraction expansion, c.f. [9] and references therein. The expected complexity of this variant is the same as the worst case bound of the subdivision solvers, i.e $\widetilde{\mathcal{O}}_B(d^4\tau^2)$.

1.1.3. Experimentation. In this section, we describe the experimental behavior of some of the implemented subdivision solvers on specific data sets. The solvers take as input, a polynomial `f` with integer, rational or interval coefficients and output intervals with rational endpoints. All use the same initial interval, given by Cauchy bound.

The following graphs illustrate the behavior of various univariate solvers, on random (D_1) and Mignotte (D_5) polynomials of maximum coefficient bit size 50 bits:



The different solvers are: `IslSturmQQ` based on the construction of the Sturm-Habicht sequence and subdivisions, using rational numbers or large integers; `IslBzIntegerZZ` implementing the Bernstein subdivision solver, using extended integer coefficients; `IslBzBdgSturmQQ` combining two solvers (in a first part, the polynomial is converted to the Bernstein representation on the initial interval, using rational arithmetic and its coefficients are rounded to `double` intervals. The Bernstein solver is applied on the polynomial with interval coefficients. If the size of the domain is too small, the Sturm solver is launched), `CORE` [16] and `SlvAberthQQ` corresponding to MPSOLVE, a numerical solver based on Aberth's method [4] and implemented by G. Fiorentino and D. Bini.

The average time over 100 runs is in seconds. The experiments were performed on a Pentium (2.6 GHz), using g++ 3.4.4 (Suse 10). The extended arithmetic is based on the library GMP. For polynomials with few, distinct and well separated real roots (D_1), we observe that the Bernstein subdivision solver perform well. When there are roots that are very close (D_5), the computation time of the Sturm-Habicht sequence is negligible. A combined solver based on numerical solvers such as MPSOLVE and subdivision techniques using for instance the Bernstein representation seems to be the most efficient approach. For more details, the reader may refer to [10].

1.2. Algebraic numbers. Algebraic numbers are of particular importance in geometric problems such as arrangement or topology computation. In geometric modeling the treatment of algebraic curves or surfaces leads implicitly or explicitly to the manipulation of algebraic numbers. A package of the library is devoted to such problems. It is dealing with real algebraic numbers, i.e. those real numbers that satisfy a polynomial equation with integer coefficients, form a real closed field denoted by $\mathbb{R}_{alg} = \overline{\mathbb{Q}}$. From all integer polynomials that have an algebraic number α as root, the primitive one (the gcd of the coefficients is 1) with the minimum degree is called *minimal*. The minimal polynomial is unique (up to a sign),

primitive and irreducible, e.g. [36]. For the computation with real algebraic numbers, we use Sturm-Habicht sequences, hence it suffices to deal with algebraic numbers, as roots of a square-free polynomial and not as roots of their minimal one. In order to represent a real algebraic number we choose the *isolating interval representation*, i.e. by a square-free polynomial and an isolating interval, that is an interval with rational endpoints, which contains only one root of the polynomial:

$$r \equiv (f(X), [a, b]), \text{where } f \in \mathbb{Z}[X] \text{and } a, b \in \mathbb{Q}.$$

In the geometric applications (topology of algebraic curves and surfaces, arrangement computation, ...) that we are targeting, this representation is enough. This is the reason why, we have not considered towers of algebraic extensions (algebraic numbers which defining polynomials are also algebraic numbers).

In order to achieve high performance for problems involving small degree polynomials (typically geometric problems for conics), the treatment of polynomials and algebraic numbers of degree up to 4, is preprocessed. We use precomputed discrimination systems (Sturm-Habicht sequences) in order to determine the square-free polynomial and to compute the isolating interval as function in the coefficients of the polynomial (and to compare algebraic numbers).

For polynomials of higher degree, we use a Sturm-like solver in order to isolate the roots of the polynomial, but we can use any other solver that can return isolating intervals. Evidently, a real algebraic number is constructed by solving (in our case by isolating) the real roots of an integer univariate polynomial.

Let us demonstrate the capabilities of the library by an example:

```
 1: #include <synaps/usolve/Algebraic.h>

 2: using namespace::SYNAPS;
 3: using namespace::std;

 4: typedef ZZ NT;
 5: typedef Algebraic<NT> SOLVER;

 6: int main() {
 7:   SOLVER::Poly f("x^9-29*x^8+349*x^7-2309*x^6+9307*x^5-23771*x^4
                     +38511*x^3-38151*x^2+20952*x-4860");
 8:   Seq<SOLVER::RO_t> sol= solve(f, SOLVER());
 9:   for (unsigned i=0; i< sol.size(); ++i)
10:    cout << "(" << i << ") " << sol[i] << endl;
11:  return 0; }
```

First the user declares the number type of the coefficients of the polynomials that he wants to deal with. In our case we use ZZ, which correspond to GMP integers. In the sequel, he declares the solve algorithm, which means that he chooses an algorithm in order to isolate the real roots of an integer polynomial. There are many solvers in SYNAPS and each of them

has a similar structure. In our example, we choose the Sturm subdivision solver. For the various univariate solvers available in SYNAPS, the reader may refer to the previous section or to [10]. Inside the main routine, the user constructs a polynomial and solves it using the `solve` function. This function constructs a sequence of real algebraic numbers that are printed subsequently.

The implementation of the algebraic numbers is in the namespace `ALGEBRAIC`. Since algebraic numbers need a lot of information concerning the ring and the field number type, the number type of the approximation etc, we gathered all this information into a `struct` called `ALG_STURM<RT>`, which takes only the ring number type `RT` as parameter and from this class we can determine all the other types. The class which implements the real algebraic numbers is `root_of<RT>`. It provides construction functions (such as `solve`, `RootOf`), comparison functions, sign function and extensions to bivariate problems, considered as univariate over a univariate polynomial ring. In order to compare two algebraic numbers, filtering techniques improving the numerical approximation combined with explicit methods based on Sturm's theorem are used. For the complexity of these operations, the reader may refer to [10] and references therein.

Moreover, projection-based algorithms exists for constructing pairs of real algebraic numbers that are real solutions of bivariate polynomials systems, as well as functions for the computing the sign of a bivariate integer polynomial, evaluated over two real algebraic numbers.

1.3. Multivariate Bernstein subdivision solver. We consider here the problem of computing the solutions of a polynomial system

$$\begin{cases} f_1(x_1, \ldots, x_n) = 0, \\ \vdots \\ f_s(x_1, \ldots, x_n) = 0, \end{cases}$$

in a box $B := [a_1, b_1] \times \cdots \times [a_n, b_n] \subset \mathbb{R}^n$. The method for approximating the real roots of this system, that we describe now uses the representation of multivariate polynomials in Bernstein basis, analysis of sign variations and univariate solvers (Section 1.1.2). The output is a set of small-enough boxes, which contain these roots. The boxes which are removed are guaranteed to contain no root, but the existence and uniqueness of a root in each output box is not provided, neither the multiplicity. The computation is done with floating point arithmetic, using controlled rounding modes.

The subdivision solver [24] that we describe now, can be seen as an improvement of the *Interval Projected Polyhedron* algorithm in [31].

In the following, we use the Bernstein basis representation of a multivariate polynomial f of the domain $I := [a_1, b_1] \times \cdots \times [a_n, b_n] \subset \mathbb{R}^n$:

$$f(x_1,\ldots,x_n) = \sum_{i_1=0}^{d_1} \cdots \sum_{i_n=0}^{d_n} b_{i_1,\ldots,i_n}\, B_{d_1}^{i_1}(x_1;a_1,b_1)\cdots B_{d_n}^{i_n} x(x_n;a_n,b_n).$$

DEFINITION 1.4. *For any $f \in \mathbb{R}[\mathbf{x}]$ and $j = 1,\ldots,n$, let*

$$m_j(f;x_j) = \sum_{i_j=0}^{d_j} \min_{\{0\le i_k\le d_k, k\neq j\}} b_{i_1,\ldots,i_n}\, B_{d_j}^{i_j}(x_j;a_j,b_j)$$

$$M_j(f;x_j) = \sum_{i_j=0}^{d_j} \max_{\{0\le i_k\le d_k, k\neq j\}} b_{i_1,\ldots,i_n} x\, B_{d_j}^{i_j}(x_j;a_j,b_j).$$

THEOREM 1.2 (Projection Lemma). *For any $\mathbf{u} = (u_1,\ldots,u_n) \in I$, and any $j = 1,\ldots,n$, we have*

$$m(f;u_j) \le f(\mathbf{u}) \le M(f;u_j).$$

As a direct consequence, we obtain the following corollary:

COROLLARY 1.1. *For any root $\mathbf{u} = (u_1,\ldots,u_n)$ of the equation $f(\mathbf{x}) = 0$ in the domain I, we have $\underline{\mu}_j \le u_j \le \overline{\mu}_j$ where*

- *$\underline{\mu}_j$ (resp. $\overline{\mu}_j$) is either a root of $m_j(f;x_j) = 0$ or $M_j(f;x_j) = 0$ in $[a_j,b_j]$ or a_j (resp. b_j) if $m_j(f;x_j) = 0$ (resp. $M_j(f;x_j) = 0$) has no root on $[a_j,b_j]$,*
- *$m_j(f;u) \le 0 \le M_j(f;u)$ on $[\underline{\mu}_j,\overline{\mu}_j]$.*

The general scheme of the solver implementation consists in

1. applying a preconditioning step to the equations;
2. reducing the domain;
3. if the reduction ratio is too small, we split the domain

until the size of the domain is smaller than a given epsilon.

The following important ingredients of the algorithm parametrize its implementation:

Preconditioner. It is a transformation of the initial system into a system, which has a better numerical behavior. Solving the system $\mathbf{f} = 0$ is equivalent to solving the system $M\,\mathbf{f} = 0$, where M is an $s \times s$ invertible matrix. As such a transformation may increase the degree of some equations, with respect to some variables, it has a cost, which might not be negligible in some cases. Moreover, if for each polynomial of the system not all the variables are involved, that is if the system is sparse with respect to the variables, such a preconditioner may transform it into a system which is not sparse anymore. In this case, we would prefer a partial preconditioner on

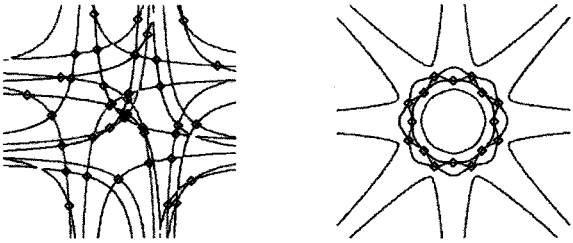
a subset of the equations sharing a subset of variables. We consider Global transformations, which minimize the distance between the equations, considered as vectors in an affine space of polynomials of a given degree and Local straightening (for $s = n$), which transform locally the system $\mathbf{f}$ into a system $J^{-1}\mathbf{f}$, where $J = (\partial_{x_i} f_j(\mathbf{u}_0)_{1 \leq i,j \leq s}$ is the Jacobian matrix of $\mathbf{f}$ at a point $\mathbf{u}$ of the domain I, where it is invertible.

It can be proved that the reduction based on the polynomial bounds m and M behaves like Newton iteration near a simple root, that is we have a quadratic convergence, with this transformation.

Reduction strategy. It is the technique used to reduce the initial domain, for searching the roots of the system. It can be based on Convex hull properties as in [31] or on Root localization, which is a direct improvement of the convex hull reduction and consists in computing the first (resp. last) root of the polynomial $m_j(f_k; u_j)$, (resp. $M_j(f_k; u_j)$), in the interval $[a_j, b_j]$. The current implementation of this reduction steps allows us to consider the convex hull reduction, as one iteration step of this reduction process. The guarantee that the computed intervals contain the roots of f, is obtained by controlling the rounding mode of the operations during the de Casteljau computation.

Subdivision strategy. It is the technique used to subdivide the domain, in order to simplify the forthcoming steps, for searching the roots of the system. Some simple rules that can be used to subdivide a domain and reduce its size. The approach, that we are using in our implementation is the parameter domain bisection: The domain B is then split in half in a direction j for which $|b_j - a_j|$ is maximal. But instead of choosing the size of the interval as a criterion for the direction in which we split, we may choose other criterion depending also on the value the functions m_i, M_j or f_j (for instance where $M_j - m_j$ is maximal). A bound for the complexity of this method is detailed in [24]. It involves metric quantities related to the system $\mathbf{f} = 0$, such as the Lipschitz constant of $\mathbf{f}$ in B, the entropy of its near-zero level sets, a bound d on the degree of the equations in each variable and the dimension n.

Examples. Here are some comparisons of the different strategies, describing the number of iterations in the main loop, the number of subdivision of a domain, the number of boxes produced by the method, the time it takes. We compare the method **sbd**, a pure subdivision approach, **rd** a method doing first reduction and based on a univariate root-solver using the Descarte's rule. **sbds** a subdivision approach using the global preconditioner, **rds** a reduction approach using the global preconditioner, **rdl**, a reduction approach using the Jacobian preconditioner. The first example is a bivariate system, with equations of degree 12) in each variable, the second example is of bi-degree $(8, 8)$.



(a) (b)

Example a:

method	iterations	subdivisions	results	time (ms)
sbd	4826	4826	220	217
rd	2071	1437	128	114
sbds	3286	3286	152	180
rds	1113	748	88	117
rdl	389	116	78	44

Example b:

method	iterations	subdivisions	results	time (ms)
sbd	84887	84887	28896	3820
rd	82873	51100	20336	4553
sbds	6076	6076	364	333
rds	1486	920	144	163
rdl	1055	305	60	120

For more details on this solver, see [28], [24].

1.4. Resultant-based methods. A projection operator is an operator which associates to an overdetermined polynomial system in several variables a polynomial depending only on the coefficients of this system, which vanishes when the system has a solution. This projection operation is a basic ingredient of many methods in Effective Algebraic Geometry. It has important applications in CAGD (Computer Aided Geometric Design), such as for the problem of implicitization of parametric surfaces, or for surface parametrization inversion, intersection, and detection of singularities of a parametrized surface. The library implements a set of resultants

Such approach based on resultant constructions yields a preprocessing step in which we generate a dedicated code for the problem we want to handle. The effective resolution, which then requires the instantiation of

the parameters of the problems and the numerical solving, is thus highly accelerated. Such solvers, exploiting adequately tools from numerical linear algebra, are numerically robust and compute efficiently approximations of all the roots, even if they have multiplicities. They can be used directly in geometric applications (see eg. [20]) or embedded in an algorithm which validates afterward the approximation. Such validation step is not yet provided in our current implementation of these resultant based methods.

The library SYNAPS contains several types of resultant constructions, such that the projective resultant, the toric resultant (based on an implementation by I. Emiris), and the Bezoutian resultant. Using these resultant matrix formulations, solving a polynomial problem can be reduced to solving the generalized eigenvector problem $T^t(x)\mathbf{v} = 0$, where $T(x)$ is a matrix of size $N \times N$ with polynomial coefficients, or equivalently a polynomial with $N \times N$ matrix coefficients.

If $d = \max_{i,j}\{\deg(T_{ij}(x))\}$, we obtain $T(x) = T_d x^d + T_{d-1}x^{d-1} + \cdots + T_0$, where T_i are $n \times n$ matrices. The problem is transformed into a generalized eigenvalue problem $(A - \lambda B)\,\mathbf{v} = 0$ where

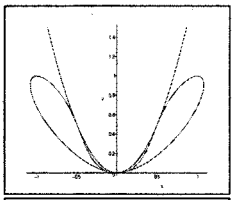
$$A = \begin{pmatrix} 0 & I & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & I \\ T_0^t & T_1^t & \cdots & T_{d-1}^t \end{pmatrix}, \qquad B = \begin{pmatrix} I & 0 & \cdots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & I & 0 \\ 0 & \cdots & 0 & -T_d^t \end{pmatrix},$$

where A, B are $N \times d$ constant matrices. And we have the following interesting property :

$$T^t(x)\,\mathbf{v} = 0 \Leftrightarrow (A - xB) \begin{pmatrix} \mathbf{v} \\ x\,\mathbf{v} \\ \vdots \\ x^{d-1}\,\mathbf{v} \end{pmatrix} = 0.$$

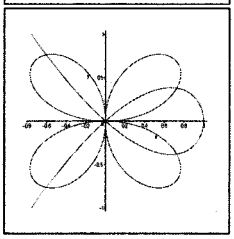
We apply it for implicit curve intersection problems in [5]. Given two polynomials $p, q \in \mathbb{Q}[x, y]$, we compute their resultant matrix, with respect to y. This yields a matrix $T(x)$, from which we deduce the coordinates of the intersection points by solving the generalized eigenvector problem $T(x)^t\,\mathbf{v} = 0$. The case of multiple roots in the resultant and of intersection points with the same x-coordinates is analyzed in details, so that we can recover their multiplicities.

Examples. Here are two examples which illustrate the behavior of the implementation. The eigenvalue computation is using the routine dgegv from the FORTRAN library LAPACK. The connection with this external library is performed transparently, through the mechanisms of views, as detailed in Section 3.



$$\begin{cases} p = x^4 - 2x^2y + y^2 + y^4 - y^3 \\ q = y - 2x^2 \end{cases}$$

```
(x1, y1) = (1.6e-09,0) of multiplicity 4
(x2, y2) = (-0.5,0.5)  of multiplicity 2
(x3, y3) = (0.5,0.5)   of multiplicity 2
Execution time = 0.005s; Eps =  10^-6.
```



$$\begin{cases} p = x^6 + 3x^4y^2 + 3x^2y^4 + y^6 - 4x^2y^2 \\ q = y^2 - x^2 + x^3 \end{cases}$$

```
(x1, y1) = (-3e-16,2e-17) of multiplicity 8
(x2, y2) = (-0.60,-0.76)  of multiplicity 1
(x3, y3) = (-0.60,0.76)   of multiplicity 1
(x4, y4) = (0.72, -0.37)  of multiplicity 1
(x5, y5) = (0.72,0.37)    of multiplicity 1
Execution time = 0.011s; Eps = 10^-3.
```

Such tools have also been used in the following CAD modeling problem: the extraction of a set of geometric primitives properly describing a given 3D point cloud obtained by scanning a real scene. If the extraction of planes is considered as a well-solved problem, the extraction of circular cylinders, these geometric primitives are basically used to represent "pipes" in an industrial environment, is not easy and has been recently addressed. We describe an application of resultant based method to this problem which has been experimented in collaboration with Th. Chaperon from the MENSI company. It proceeds as follows: First, we devise a polynomial dedicated solver, which given 5 points randomly selected in our 3D point cloud, computes the cylinders passing through them (recall that 5 is the minimum number of points defining generically a finite number of cylinders, actually 6 in the *complex* numbers). Then we apply it for almost all (or randomly chosen) sets of 5 points in the whole point could, and extract the "clusters of directions" as a primitive cylinder. This requires the intensive resolution of thousands of small polynomial systems. Classical resultant or residual resultant constructions are used in this case, to compute quickly their roots, even in the presence of multiple roots.

1.5. Generalized normal form. The solver that we describe now computes all the complex roots of a zero-dimensional polynomial systems in $\mathbb{C}^n$. It proceeds in two steps:

- Computation of the generalized normal form modulo the ideal $(f_1, \ldots, f_n)$.
- Computation of the roots from eigencomputation.

Hereafter, the polynomials are in the ring $\mathbb{K}[x_1, \ldots, x_n]$ with coefficients in the field $\mathbb{K}$ (eg. $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$) and variables $x_1, \ldots, x_n$.

A classical approach for normal form computation is through Gröbner basis [6]. Unfortunately, their behavior on approximate data is not satisfactory [23]. In SYNAPS, a generalized normal form method is implemented which allows us to treat polynomial systems with approximate coefficients,

more safely. Such a normal form computation is available with exact arithmetic (rational numbers or modular numbers) and with extended floating point arithmetic. The numerical stability of the generalized normal form computation is improved by the allowing column pivoting on the coefficient matrices of the involved polynomials, which is not possible for Gröbner basis computation. In the case of floating point arithmetic, no certification is provided yet in the current implementation.

The method constructs a set of monomials B and *a rewriting family* F on the monomial set B, which have the following property: $\forall f, f' \in F$,

- f has exactly **one** monomial that we denoted by $\gamma(f)$ (also called the *leading monomial* of f) in ∂B,
- $\text{supp}(f) \subset B^+$, $\text{supp}(f - \gamma(f)) \subset B$,
- if $\gamma(f) = \gamma(f')$ then $f = f'$,

where $B^+ = B \cup x_1 B \cup \cdots \cup x_n B$, $\partial B = B^+ \backslash B$. Moreover, the monomial set B will be connected to 1: for every monomial m in B, there exists a finite sequence of variables $(x_{i_j})_{j \in [1,l]}$ such that $1 \in B$, $\Pi_{j=1 \ldots l'} x_{i_j} \in B$, $\forall l' \in [1, l]$ and $\Pi_{j \in [1,l]} x_{i_j} = m$.

The normal form construction is based on the following criteria:

THEOREM 1.3. *[22] Let B be a monomial set connected to 1. Let $R_F : B^+ \to B$ be a projection from B^+ to B, with kernel F and let $I = (F)$ be the ideal generated by F. Let $M_i : B \to B$ be defined by $b \mapsto R_F(x_i b)$. Then, the two properties are equivalent:*

1. *For all $1 \leq i, j \leq n$, $M_i \circ M_j = M_j \circ M_i$.*
2. *$\mathbb{K}[x_1, \ldots, x_n] = \langle B \rangle \oplus I$.*

The algorithm consists in constructing degree by degree the set B and in checking at each level, whether the partial operator of multiplication commutes. For more details, see [33, 25, 26, 23].

From this normal form N, we deduce the roots of the system as follows. We use the properties of the operators of multiplication by elements of $\mathcal{A} = R/(f_1, .., f_s))$, as follows (see [2], [21], [32]):

ALGORITHM 1.5. SOLVING IN THE CASE OF SIMPLE ROOTS

Let $a \in \mathcal{A}$ such that $a(\zeta_i) \neq a(\zeta_j)$ if $i \neq j$ (which is generically the case).

1. Compute the M_a the multiplication matrix by a in the basis $x^E = (1, x_1, \ldots, x_n, \ldots)$ of $\mathcal{A}$.
2. Compute the eigenvectors $\Lambda = (\Lambda_1, \Lambda_{x_1}, \ldots, \Lambda_{x_n}, \ldots)$ of M_a^t.
3. For each eigenvector Λ with $\Lambda_1 \neq 0$, compute and return the point $\zeta = (\frac{\Lambda_{x_1}}{\Lambda_1}, \ldots, \frac{\Lambda_{x_n}}{\Lambda_1})$.

The case of multiple roots is treated by simultaneous triangulation of several multiplication matrices. The main ingredients which are involved here are sparse linear algebra (implemented from the direct sparse linear solver **superLU**), and eigenvalue and eigenvector computation (based on LAPACK

routines). These different types of tools are combined together, in order to obtain an efficient zero-dimensional polynomial system solver. We are currently working on techniques adapted from thoses of [30, 27] to certify *a posteriori* the numerical approximation of the roots obtained by such eigenvalues/eigenvectors computation. These techniques will also allow certified root refinement. Eventually a purely symbolic representation of the roots will be added.

2. Geometric computing with curves and surfaces. A special context where algebraic operations on curves and surfaces are critical is non-linear computational geometry. Shapes are modeled by semi-algebraic sets, where the local primitives are implicit or parametric curves or surfaces. This is typically the case in CAGD, where NURBS or Bspline functions [11] are used standard primitives. The degree of the piecewise polynomial models is usually 3 (or 5) in each variable. In this section, we describe some functions from the module **shape** of the library SYNAPS. We focus on tools for computing the topology of implicit curves and surfaces, which are fundamental in arrangement problems. All these methods rely on polynomial solvers as fundamental ingredients. They guaranty their results, provided the polynomial solvers are able to perform certified root isolation.

2.1. Sweeping of planar curves. Let $\mathcal{C}$ be a planar implicit curve defined by a polynomial equation $f(x,y) = 0$, where $f \in \mathbb{Q}[x,y]$. Here is the scheme of an algorithm, which outputs a graph of points in the plane, isotopic to the curve $\mathcal{C}$, and with the following properties:

ALGORITHM 2.1. TOPOLOGY OF AN IMPLICIT CURVE

INPUT: an algebraic curve $\mathcal{C}$ given by a square-free equation $f(x,y) = 0$ with $f \in \mathbb{Q}[x,y]$.

- Choose a direction (say the x-axis direction), and consider a virtual line orthogonal to this direction, sweeping the plane;
- Detect at which critical position, the topology of the intersection of the line and the curve changes (the roots of the resultant $\mathtt{res}_y(f, \partial_y f)(x) = 0$.
- Compute the corresponding intersection points $f(\alpha, y) = 0$ for α a root of $\mathtt{res}_y(f, \partial_y f)(x) = 0$.
- Compute the intersection points for sample lines in between these critical positions.
- Check the generic position (at most one critical point per sweeping line) and connect the computed points by segments, in order to get a graph isotopic to $\mathcal{C}$.

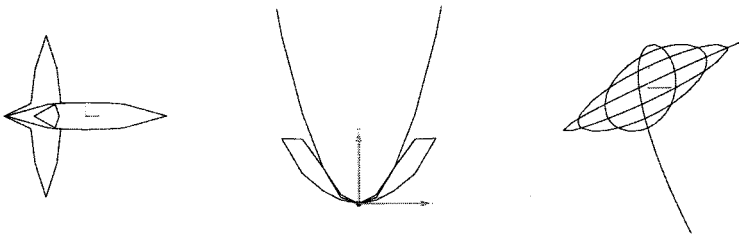
In this algorithm, we compute the Sturm-Habicht sequence of the two polynomials $f, \partial_y f$ with respect to the variable y. The last term of this sequence is their resultant $r(x) \in \mathbb{Q}[x]$. We solve this equation. Let $\alpha_1 < \cdots < \alpha_s$

be the real roots of $r(x) = 0$. For each α_i, we compute the corresponding y such that $f(\alpha_i, y) = 0$. At this point, we also check that the curve is in *generic* position, that is, there is at most one multiple solution of $f(\alpha_i, y) = 0$. This construction involve the manipulation of algebraic numbers and the use of univariate solvers (see Sections 1.1 and 1.2).

Once these points are computed, we compute intermediate (rational) points $\mu_0 < \mu_1 < \cdots < \mu_s$, such that $\mu_{i-1} < \alpha_i < \mu_i$ and the corresponding y such that $f(\mu_i, y) = 0$. Here again we use a univariate solver, knowing that the roots are simple (using a Bernstein subdivision solver).

All these points are collected and sorted by lexicographic order such that $x > y$. The subset of points, corresponding to two consecutive x-coordinates are connected, according to their y-coordinates. See [17] for more details.

Examples.



Here is how the function and the external viewer AXEL in SYNAPS are called:

```
MPol<QQ> p("5*x^4*y-2*x^6-4*x^2*y^2+y^3+y^5-2*y^4*x^2-y^4+2*x^2*y^3");
topology::point_graph<double> g;
topology(g, p,TopSweep2d<double,SlvBzBdg<double> >());
axel::ostream os("tmp.axl"); os<<g; os.view();
```

It computes a graph of points `g` (of type `topology::point_graph<C>`) which is isotopic to the curve $\mathcal{C}$, defined by the polynomial `p`. The coefficients of the points in the computed graph are of type `C`. The polynomial `p` is converted to a polynomial with rational coefficients, if needed. The class `TopSweep2d<C,SLV>` depends on two parameters:

- `C`, which is the type of coefficients of the points in the result,
- `SLV`, which is the type of univariate solver to be used. The default value is `SlvbzBdg<double>`.

2.2. Subdivision method for planar curves. In this section, we consider a curve $\mathcal{C}$ in $\mathbb{R}^2$, defined by the equation $f(x, y) = 0$ with $f \in \mathbb{Q}[x, y]$ and a domain $B = [a, b] \times [c, d] \subset \mathbb{R}^2$.

In order to compute the topology of this curve in a box, we use the notion of regularity:

DEFINITION 2.2. *We say that the curve $\mathcal{C}$ is y-regular (resp. x-regular) in B, if $\mathcal{C}$ has no tangent parallel to the y-direction (resp. x-direction) in B.*

Notice that if C is x-regular (or y-regular) it is smooth in B since it cannot have singular points in B. A curve is *regular* in B, if it is x-regular *or* y-regular in B.

We use the property that if C is x-regular in B, then its topology can be deduced from its intersection with the boundary ∂B.

PROPOSITION 2.1. *[17] If C is regular in B, its topology in B is uniquely determined by its intersection $C \cap \partial B$ with the boundary of B.*

Here is a simple test for the regularity of a curve in a domain, which extends in some way the criterion in [12]:

PROPOSITION 2.2. *[17] If the coefficients of $\partial_y f(x,y) \neq 0$ (resp. $\partial_x f(x,y) \neq 0$) in the Bernstein basis of the domain $B = [a,b] \times [c,d] \subset \mathbb{R}^2$ have the same sign $\in \{-1,1\}$, then the curve C is regular on B.*

In this case, we have the following connection algorithm, which output a set of segments isotopic to the curve in the domain.

- Compute the points of $C \cap \partial B$, repeating a point if its multiplicity is even.
- Sort them by lexicographic order so that $x > y$: $\mathcal{L} := \{p_1, p_2, \ldots\}$
- Connect them by pair $[p_1, p_2], [p_3, p_4], \ldots$ of consecutive points in $\mathcal{L}$.

This yields the following subdivision algorithm for a planar implicit curve:

ALGORITHM 2.3. TOPOLOGY OF A PLANAR IMPLICIT CURVE

INPUT: A box $B_0 \subset \mathbb{R}^2$, a curve C defined by the squarefree polynomial equation $f(x,y) = 0$ with $f \in \mathbb{Q}[x,y]$ and $\epsilon > 0$.

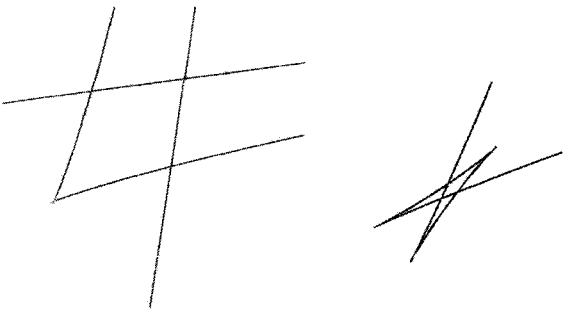
- Compute the x and y critical points of $f(x,y) = 0$.
- $\mathcal{L} = \{B_0\}$
- While $\mathcal{L}$ is not empty,
  - Choose and remove a domain B of $\mathcal{L}$;
  - Test if there is a unique critical point, which is not singular in B;
  - If it is the case, compute the topology of C in B,
  - else if $|B| > \epsilon$, subdivide the box into subdomains and add them to $\mathcal{L}$,
  - otherwise connect the center of the box to the points of $C \cap \partial B$.

OUTPUT: a set of points and a set of (smooth) arcs connecting these points.

PROPOSITION 2.3. *For $\epsilon > 0$ small enough, the algorithm 2.3 computes a graph of points which is isotopic to the curve $C \cap B$.*

The subdivision level (controlled by ϵ) can be improved significantly by analyzing the number of real branches at a singular point of the curve C, using topological degree computation. The solver used to compute these singular points should be able to isolate them from other extremal points of f (real roots of $\partial_x f = \partial_y f = 0$) to guaranty the topology of the curve.

Example c:



This curve is the discriminant curve of a bivariate system with few monomials used in [7] to give a counter-example to Kushnirenko's conjecture. It is of degree 47 in x and y, and the maximal bit size of its coefficient is of order 300. It takes less that 10 s. (on an Intel M 2.00GHz i686) to compute the topological graph, by rounding up and down the Bernstein coefficients of the polynomial to the nearest double machine precision numbers, applying the subdivision techniques on the enveloping polynomials. We observe a very tiny domain, which at first sight looks like a cusp point, but which contains in reality, 3 cusps points and 3 crossing points. The central region near these cusp points is the region where counter-examples have been found.

Example d:

This curve is the projection onto the (x, y) plane of the curve of points with tangent parallel to the z-direction for a surface of degree 4. It is defined by the equation of degree 12, and has 4 real cusps and 2 real crossing points. The size of the coefficients is small and the topological graph is computed in less than 1 s. It defines 4 connected regions in the plane, one of these being very small and difficult to see on the picture.

2.3. Subdivision approach for space curves. In this section, we consider a curve $\mathcal{C}$ of $\mathbb{R}^3$. We suppose that $I(\mathcal{C}) = (f_1, f_2, \ldots, f_k)$ and for two polynomials $f(x, y, z), g(x, y, z) \in I(\mathcal{C})$, we define $\mathbf{t} = \nabla(f) \wedge \nabla(g)$. We are interested in the topology of $\mathcal{C}$ in a box $B = [a_0, b_0] \times [a_1, b_1] \times [a_2, b_2]$. Similar to the 2D case, we can represent f, g and each component of $\mathbf{t}$ in the Bernstein basis for the domain B. As we will see, the sign changes of the resulting Bernstein coefficients will make it possible to test the regularity of the curve with minimal effort.

Here is a criteria of regularity of space curves which allows us to deduce the topology of $\mathcal{C}$ in the domain:

PROPOSITION 2.4. *[17] Let $\mathcal{C}$ be a 3D spatial curve defined by $f = 0$ and $g = 0$. If*

- $\boldsymbol{t_x(x) \neq 0}$ *on* B, *and*
- $\partial_y h \neq 0$ *on* z*-faces, and* $\partial_z h \neq 0$ *and it has the same sign on both* y*-faces of* B, *for* $h = f$ *or* $h = g$,

then the topology of $\mathcal{C}$ *is uniquely determined from the points* $\mathcal{C} \cap \partial B$.

A similar criterion applies by symmetry, exchanging the roles of the x, y, z coordinates. If one of these criteria applies with $\mathbf{t}_i(x) \neq 0$ on B (for $i = x, y, z$), we will say that $\mathcal{C}$ is i-regular on B.

From a practical point of view, the test $\mathbf{t}_i(x) \neq 0$ or $\partial_i(h) \neq 0$ for $i = x, y$ or z, $h = f$ or g can be replaced by the stronger condition that their coefficients in the Bernstein basis of B have a constant sign, which is straightforward to check. Similarly, such a property on the faces of B is also direct to check, since the coefficients of a polynomial on a facet form a subset of the coefficients of this polynomial in the box.

In addition to these tests, we also test whether both surfaces penetrate the cell, since a point on the curve must lie on both surfaces. This test could be done by looking at the sign changes of the Bernstein coefficients of the surfaces with respect to that cell. If no sign change occurs, we can rule out the possibility that the cell contains any portion of the curve $\mathcal{C}$, and thus terminate the subdivision early. In this case, we will also say that the cell is regular.

This regularity criterion is sufficient for us to uniquely construct the topological graph g of $\mathcal{C}$ within B. Without loss of generality, we suppose that the curve $\mathcal{C}$ is x-regular in B. Hence, there is no singularity of $\mathcal{C}$ in B. Furthermore, this also guarantees that there is no 'turning-back' of the curve tangent along x-direction, so the mapping of $\mathcal{C}$ onto the x axis is injective. Intuitively, the mapped curve should be a series of non-overlapping line segments, whose ends correspond to the intersections between the curve $\mathcal{C}$ and the cell, and such mapping is injective.

This property leads us to a unique way to connect those intersection points, once they are computed in order to obtain a graph representing the topology of $\mathcal{C}$, similar to the 2D method.

In order to apply this algorithm, we need to compute the points of $\mathcal{C} \cap B$, that is to solve a bivariate system of each facet of B. This is performed by applying the algorithm described in Section 1.3.

The special treatment of points of $\mathcal{C}$ on an edge of B or where $\mathcal{C}$ is tangent to a face requires the computation of tangency information at these points. This is performed by evaluating the derivatives of the defining equations of $\mathcal{C}$ at these points.

Collecting these properties, we have the following subdivision algorithm, which subdivides the domain B until some size ϵ, if the curve is not regular in B. It relies on a bivariate solver, for computing the intersection of the curve with the faces of the box.

ALGORITHM 2.4. TOPOLOGY OF A SPACE CURVE

INPUT: a curve C defined by equations $f_1 = 0, f_2 = 0, \ldots, f_k = 0$ and a domain $B = [a_0, b_0] \times [a_1, b_1] \times [a_2, b_2] \subset \mathbb{R}^3$ and $\epsilon > 0$.

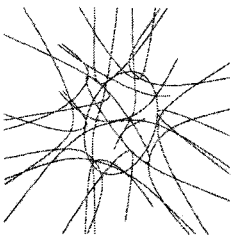
- For $1 \leq i < j \leq k$,
  - compute the Bernstein coefficients of the x, y, z coordinates of $\nabla f_i \wedge \nabla f_j$ in B
  - check that they are of the same sign for one of the coordinates (say x);
  - check the x-regularity condition on the facets of B.
- If such a pair (i, j) satisfying the previous regularity condition exists,
  - Compute the points of $C \cap \partial B$ and connect them.
- else if $|B| > \epsilon$, subdivide B and proceed recursively on each subdomain.
- otherwise find a point p in $\overset{\circ}{B}$, compute the point $C \cap \partial B$ and connect them to p.

OUTPUT: a set of points p and a set of arcs connecting them.

As in the 2D case, we have the following "convergence" property:

PROPOSITION 2.5. *For $\epsilon > 0$ small enough, the graph of points and arcs computed by the algorithm has the same topology as $C \cap B$.*

Example. This curve is defined by $f(x, y, z) = 0$, $\partial_z f(x, y, z) = 0$ where



$$f(x,y,z) = 4(\tau^2 x^2 - y^2)(\tau^2 y^2 - z^2)(\tau^2 z^2 - x^2) - (1 + 2\tau)(x^2 + y^2 + z^2 - 1)^2,$$
$$\tau = \frac{1}{2}(1 + \sqrt{5}),$$

of degree 6 is defining the surface S called Barth's Sextic. This curve, called the polar curve of S in the z direction, is of degree $30 = 6 \times 5$. We compute the topology by approximating the coefficients of f and $\partial_z f$ by floating point numbers.

2.4. Subdivision method for surfaces. In this section, we consider a surface $\mathcal{S}$ defined by the equation $f(x, y, z) = 0$, with $f \in \mathbb{Q}[x, y, z]$. We assume that f is squarefree, that is f has no irreducible factors of multiplicity ≥ 2. For more details, see [1].

Unlike in the 2 dimensional case, the topology of the singular locus and the way to smooth locus is attached to it can be really complicated. Topologically we can characterize the topological situation as follows:

- Near a 2-dimensional stratum the topology is the same as a hyperplane.
- Near a 1-dimensional stratum the topology is the same as a cylinder on a singular planar curve.

- Near a 0-dimensional stratum the topology is the same as a cone with base the surface intersect a small ball containing the 0-dimensional stratum.

Moreover, we know only one of these three situations can and will happen locally. So we just have to design a solution for each one of the above three cases.

For efficiency reasons the criteria we have designed work for situations more general than the limit three cases. The 2-dimensional strata criterion can succeed even with several hyperplanes in the box not only just one. For the 1-dimensional strata we can triangulate even when some patches of the 2-dimensional strata lie in the box even though they are disconnected from the singular locus (in the box). In the case of the 0-dimensional strata we need to have the exact topology in the box. Of course the criteria eventually succeed if the box is small enough. We now describe each one of these criteria and the matching connection algorithm.

Let $B = [a, b] \times [c, d] \times [e, f] \subset \mathbb{R}^3$ and a surface $\mathcal{S} \subset B$. The boundary of B is denoted hereafter by ∂B. The x-faces (resp. y, z-facet) of B are the planar domains of the boundary ∂B of B, orthogonal to the direction x (resp. y, z).

DEFINITION 2.5. *The surface $\mathcal{S}$ is z-regular (resp. y, z-regular) in the domain B if,*

- *$\mathcal{S}$ has no tangent line parallel to the z-direction (reps. x, y-direction),*
- *$\mathcal{S} \cap F$ is regular, for F a z-facet (resp. x, y-facet) of B.*

We will say that $\mathcal{S}$ is *regular* in B if it is regular in B for the direction x, y or z. Here again, if a point $p \in \mathcal{S}$ is singular, then any line through this point is tangent to $\mathcal{S}$ at p. Thus a surface in B which is regular is also smooth.

PROPOSITION 2.6. *[1] If $\mathcal{S}$ is regular in B, then its topology is uniquely determined by its intersection with the edges of B.*

As in the 2D case, simple tests of regularity can be derived from the representation of f in the Bernstein basis.

PROPOSITION 2.7. *Let (u, v, w) be any permutation of (x, y, z). Suppose that the coefficients of $\partial_u f$ in the Bernstein basis of B have the same sign $\in \{-1, 1\}$ and that the coefficients of $\partial_v f$ or $\partial_w f$ on the u-facets of B are also of the same sign $\in \{-1, 1\}$. Then $\mathcal{C}$ is regular in B.*

This criterion implies that in the valid cells, the derivative of f in one direction is of constant sign and on the two faces transverse to this direction, another derivative is of constant sign. This may be difficult to obtain, when a point of the surface where two derivatives vanish is on (or near) the boundary of the cell. A situation where $\partial_u f \neq 0$ but where both derivative $\partial_v f$, $\partial_w f$ are not of constant sign on a u-facet F of B, can be handled by applying recursively the 2D algorithm of the facet F.

For handling the singularities of an algebraic surface $f(x, y, z) = 0$, we exploit the properties of the polar variety $\mathfrak{C}_z(\mathcal{S})$ defined by $f(x, y, z) =$

$0, \partial_z f(x,y,z) = 0$. For that purpose, we apply the implicit space curve algorithm of Section 2.3.

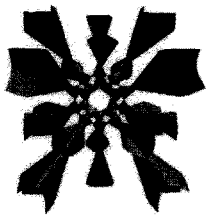
ALGORITHM 2.6. TOPOLOGY OF A SURFACE

INPUT: a surface $\mathcal{S}$ defined by a squarefree equation $f(x,y,z) = 0$, a domain $B_0 = [a_0, b_0] \times [a_1, b_1] \times [a_2, b_2] \subset \mathbb{R}^3$ and $\epsilon > 0$.

- Compute generators $g_1, \ldots, g_k$ of the ideal $I(\mathfrak{C}_z(\mathcal{S})) = (f, \partial_z f) : J(f, \partial_z f)$.
- Compute the Bernstein coefficients of the f and g_i in the Bernstein basis of $B := B_0$.
- If $\mathcal{S}$ is regular in B, compute its topological structure.
- Else if the polar variety $\mathfrak{C}_z(\mathcal{S})$ is regular and connected in B, compute the topological structure of $\mathcal{S} \cap \partial B$ by Algorithm 2.3 on each facet of B
- Else if $|B| > \epsilon$, subdivide the box B and proceed recursively on each subdomain.
- Otherwise find a point p in $\overset{\circ}{B}$, compute the topological structure of $\mathcal{S} \cap \partial B$ by Algorithm 2.3 and its link over p.

OUTPUT: a set of points, arcs and patches and adjacency relations describing the topology of $\mathcal{S} \cap B_0$ arcs connecting them.

As in the previous case, for $\epsilon > 0$ small enough, the output of this algorithm is topologically equivalent to $\mathcal{S} \cap B$.
Example.



Here is the Barth sextic surface whose polar variety has been computed in section 2.3. This surface of degree 6 has the maximum number of isolated singularities for this degree, that is 65. These singular points are also singular points of its polar variety.

3. The design of the library. As illustrated in previous sections, various internal representations (eg. dense Bernstein basis or sparse representations for polynomials) for the abstract data-types (eg. vectors, polynomials) are required, together with algorithm specializations on these representations. The library SYNAPS makes it possible to define parametrized but efficient data structures for fundamental algebraic objects such as vectors, matrices, monomials and polynomials ... which can easily be used in the construction of more elaborated algorithms.

We pay special attention to genericity in designing structures for which effectiveness can be maintained. Thanks to the parametrization of the code using *templates* and to the control of their instantiations using *traits* and *template expressions* [14], they offer generic programming without losing effectiveness. We need to combine generic implementations, which allow to

reuse code on different types of data representation, with specialized implementations tuned to specific critical operations on some of these data structures. This is typically the case if we want to use external or third-party libraries, such as LAPACK (Fortran library for numerical linear algebra), GMP (C library for extended arithmetics), or MPSOLVE (C univariate solver implementation, using extended multiprecision arithmetic). For instance LAPACK routines should coexist with generic ones on matrices. In order to optimize the implementation, while avoiding rewriting several times the same code, we need to consider hierarchical classes of data-structures and a way to implement specializations for each level in this hierarchy. In this section, we describe the design of the library, which allows such a combination. Since, it is aimed to be used in dedicated applications, we should also take care of techniques to embed the library functionalities into an external application. In this section, we also describe the method, that we adopt to build dynamic modules, combining generic and special code through a transparent plugin mechanism. This approach is illustrated by the connection with the geometric modeler AXEL, which uses SYNAPS library for geometric computation on algebraic curves and surfaces.

3.1. The view hierarchy. A mathematical object can have different logic representations. Moreover, the fact that a data structure represents a particular mathematical object is a meaningful information in generic programming in order to dispatch algorithms. Template functions can be defined for a formal type parameter T, assuming a semantically and syntactically well defined interface. This interface is called a *concept*. When an instantiation on T of a generic algorithm makes sense, we say that T is a *representation* of the *concept* assumed by the algorithm.

In order to dispatch a function call to the right generic version, people have to write some type declarations associated to T, specifying what kind of *concept* it implements. This can be done using *traits*, which are types wearing explicit information about types. In our development, we choose another solution, based on a *view* mechanism.

The idea behind a *view* is to reify a *concept* in order to dispatch algorithms by concepts using function overloading: when a data type `T` implements a *concept* `C`, we say that it *can be seen* as a `C`. A *view* is then defined as a generic type parametrized by a formal type `T` that is assumed to implement the *concept* associated to the *view*. As an example, `Polynomial<T>` is a *view* associated to the `Polynomial` *concept*, reifying the information that `T` implements the required interface for the `Polynomial` *concept*. Let's assume we write a generic function `f` expecting a type implementing the `Polynomial` *concept*, written in the namespace `POLYNOMIAL`:

```
namespace POLYNOMIAL {
    // generic implementation of f assuming the Polynomial concept
    template<typename Polynomial> void f(const Polynomial & p)
    { ... ; g(p) ; ... }
} ;
```

and the following function:

```
template<class PolynomialType> struct Polynomial {} ;

template<class PolynomialType> void f(const Polynomial<PolynomialType> & p)
{
    using namespace POLYNOMIAL ; f((const PolynomialType&)p) ;
}
```

Then, by specifying that a data type T inherits from `Polynomial<T>`, the preceding code ensures that the generic function `POLYNOMIAL::f` will be used when there is no version of `f` written for T. A polynomial having dense representation can allow a better implementation of `f`, in this case, we can say that it implements the `DensePolynomial` *concept* which is a *sub-concept* of `Polynomial`.

To reify the idea of *sub-concept*, we define the `DensePolynomial<T>` *view* as inheriting from `Polynomial<T>`. This way, a type `T` inheriting from `DensePolynomial<T>`, will have an implementation of `f` corresponding to the first one available, *seeing* the type successively as a `T`, a `DensePolynomial<T>` and a `Polynomial<T>`. This mechanism also allows to specialize a function for a given representation (eg. representation in the Bernstein basis `bezier::repId<C>`), by providing the function

```
namespace bezier { template<class C> void f(const bezier::repId<C> & p) ; }
```

3.2. Interfacing with an interactive environment. The preceding *view* mechanism is in fact a way to associate functions to objects considering a hierarchy of *concepts*. It is actually closely linked to another kind of design by virtual classes, which we used to make the library collaborate with the modeler AXEL. In this framework, the external tool defines a virtual hierarchy of objects and each interface defines a set of member functions inheriting one from another, and, corresponds to a *concept*, each inheritance relationship being seen as a *sub-concept* declaration. We describe here the procedure used to link the static view hierarchy with the dynamic virtual hierarchy.

The interfaces `I` are implemented as classes with virtual functions. In order to automatically construct a class which implements the set of functions for the interface `I` and the representation `R`, we define a wrapper class `W<I,R>`.

To choose, at compilation time (see section 3.1), the most specialized function for a given data type, we define a view class `V<I,R>`, where `R` is a representation implementing the *concept* associated to the interface `I`.

Suppose that we have defined a function α, which associates to a type `T`, its interface class `I` (implemented by the traits class `interfaceof<T>`), and, that we also have defined β(`T`) operating on types, which computes the base class of `T` (accessible as `T::base_t`). Then for a given representation type `R`, the following inheritance relationship of classes defined by induction as:

$$\texttt{R} \rightarrow \texttt{W<}\alpha\texttt{(R),R>}; \texttt{W<I,R>} \rightarrow \texttt{V<I,R>}; \texttt{V<I,R>} \rightarrow \texttt{W<}\beta\texttt{(I),R>}; \texttt{V<}\emptyset\texttt{,R>} \rightarrow \alpha\texttt{(R)}$$

yields the following inheritance chain:

$$\texttt{R} \rightarrow \texttt{W<I}_0\texttt{,R>} \rightarrow \texttt{V<I}_0\texttt{,R>} \rightarrow \texttt{W<I}_1\texttt{,R>} \rightarrow \ldots \rightarrow I_*,$$

where I_0 is the interface of R and $I_1, \ldots, I_*$, the upper level of classes in the hierarchy. This allows us to define the specialized implementations of this hierarchy level by level and to automatically choose most optimized functions.

Let's assume that we have defined two `length` generic functions for the `ParametricCurve` and `BSplineCurve` *concepts*, that is, for the types `V<IParametricCurve,R>` and `V<IBSplineCurve,R>`:

```
template<class C, class R> C length(const V<IParametricCurve, R>
& c, const C & eps)
{
    using namespace PARAMETRICCURVE ; return length(c.rep(), eps) ;
}

template<class C, class R> C length(const V<IBSplineCurve, R>
& c, const C & eps)
{
    using namespace RATIONALCURVE ; return length(c.rep(), eps) ;
}
```

We define now the implementation of the interface, using the wrapper class associated to the `IParametricCurve`:

```
template<class R> struct W<IParametricCurve,R> : V<IParametricCurve,R>
{
    double length(double eps) const { return length(rep(), eps) ; }
} ;
```

In order to insert a specific representation of rational curves `MPolDseRationalCurve<C>` in the hierarchy, we use the following construction which specifies its interface and its implementation:

```
template<class C> struct interfaceof< MPolDseRationalCurve<C> >
{
    typedef IRationalCurve T ;
} ;

template<class C> struct MPolDseRationalCurve
: W< IRationalCurve,MPolDseRationalCurve<C> >
{
    typedef IRationalSurface base_t ;
    ...
} ;
```

This technique allows us to easily embed the library into an external interactive environment such as the geometric modeler AXEL, as we illustrate it now. This yields a plugin `ShapePlugin`, which will be compiled separately and loaded dynamically into the modeler. To build this plugin, we first furnish a factory from a list of types:

```
typedef type::gentlist<
    MPolDseRationalCurve<double>,
    MPolDseRationalSurface<double>,
    ...
>::T TypeList ;
```

```
void ShapePlugin::init(void)
{
    factory = new WShapeFactory<TypeList> ;
}
```

This factory allows to maintain a map from the interfaces of external tools (as in AXEL) to SYNAPS interfaces, which is enriched each time an object is created:

```
void ShapePlugin::create(QRationalCurve * c)
{
    IShape * shape = factory->interface("IRationalCurve") ;
    IRationalCurve * rc ;
    IParametricCurve * pc ;
    rc = dynamic_cast<IRationalCurve *>(shape) ;
    pc = dynamic_cast<IParametricCurve *>(shape) ;
    rc->setEquations(c->pw(),c->px(),c->py(),"t") ;
    pc->setRange(c->smin(),c->smax()) ;
    map.insert(c, shape) ;
}
```

The application forwards function calls over its objects to calls of functions in the plugin, following the same pattern. This approach allows to make code designed for different purposes, occasionally in different languages, coexist.

4. Conclusion. SYNAPS is an open source c++ library for symbolic and numeric computations, that provides algebraic algorithms and data structures for manipulating polynomials, for solving polynomial equations and computing with real algebraic numbers. The wide range of the algebraic operations that are implemented in the library, as well as the design of it, based on the view mechanism, allows us to tune the algebraic operations in order to tackle difficult problems in non-linear computational geometry, such as the computation of the topology of real plane and space curves and surfaces. Last but not least, the design of the library permits the development of algebraic plugins that can be used as algebraic primitives to external software packages specialized to geometric modeling, such as AXEL.

SYNAPS is a continuous effort to combine symbolic and numeric techniques in algebraic computing under the generic programming paradigm. The library contains more than 200,000 lines of codes. In a short term, we plan to structure it into autonomous subpackages and to extend its capability to build dedicated plugins for external interactive tools, while improving the implementations for polynomials.

Acknowledgments. B. Mourrain, J.P. Pavone, and J. Wintz are partially supported by the European projects ACS (Algorithms for Complex Shapes, IST FET Open 006413), AIM@Shape (IST Network of Excellence 506766) and the ANR project GECKO (Geometry and Complexity).

REFERENCES

[1] L. ALBERTI, G. COMTE, AND B. MOURRAIN. Meshing implicit algebraic surfaces: the smooth case. In L.L. Schumaker M. Maehlen, K. Morken (editors), *Mathematical Methods for Curves and Surfaces: Tromso'04*, pages 11–26. Nashboro, 2005.

[2] W. AUZINGER AND H. J. STETTER. An elimination algorithm for the computation of all zeros of a system of multivariate polynomial equations. In *Proc. Intern. Conf. on Numerical Math.*, Vol. **86** of *Int. Series of Numerical Math*, pages 12–30. Birkhäuser Verlag, 1988.

[3] S. BASU, R. POLLACK, AND M.-F. ROY. *Algorithms in Real Algebraic Geometry.* Springer-Verlag, Berlin, 2003. ISBN 3-540-00973-6.

[4] D. BINI. Numerical computation of polynomial zeros by means of aberth's method. *Numerical Algorithms*, 13, 1996.

[5] L. BUSÉ, H. KHALIL, AND B. MOURRAIN. Resultant-based method for plane curves intersection problems. In *Proc. of the conf. Computer Algebra in Scientifc Computing*, Vol. **3718** of *LNCS*, pages 75–92. Springer, 2005.

[6] D. COX, J. LITTLE, AND D. O'SHEA. *Ideals, Varieties, and Algorithms*, 2nd edition. Undergraduate Texts in Mathematics. Springer, New York, 1997.

[7] A. DICKENSTEIN, M.J. ROJAS, K. RUSEKZ, AND J. SHIHX. Extremal real algebraic geometry and A-discriminants. Preprint, 2007.

[8] A. EIGENWILLIG, V. SHARMA, AND C. K. YAP. Almost tight recursion tree bounds for the descartes method. In *ISSAC '06: Proceedings of the 2006 International Symposium on Symbolic and Algebraic Computation*, pages 71–78. ACM Press. New York, NY, USA, 2006.

[9] I.Z. EMIRIS AND E.P. TSIGARIDAS. Univariate polynomial real root isolation: Continued fractions revisited. In *14th Annual European Symposium on Algorithms*, Vol. **4168** of *LNCS*, pages 817–828. Springer, 2006.

[10] I.Z. EMIRIS, B. MOURRAIN, AND E.P. TSIGARIDAS. Real algebraic numbers: Complexity analysis and experimentations. In P. Hertling, C.M. Hoffmann, W. Luther, N. Revol (editors), *Reliable Implementation of Real Number Algorithms: Theory and Practice*, *LNCS*, Springer-Verlag, 2007 (to appear).

[11] G. FARIN. *Curves and surfaces for computer aided geometric design: A practical guide.* Comp. Science and Sci. Computing. Acad. Press, 1990.

[12] M.S. FLOATER. On zero curves of bivariate polynomials. *Journal Advances in Computational Mathematics*, **5**(1):399–415, 1996.

[13] J.C. FAUGÈRE AND F. ROUILLIER, *FGB/RS Gröbner basis computation and real root isolation*, `http://fgbrs.lip6.fr/salsa/Software/`

[14] ERICH GAMMA, RICHARD HELM, RALPH JOHNSON, AND JOHN VLISSIDES. Design patterns: Abstraction and reuse of object-oriented design. *Lecture Notes in Computer Science*, **707**:406–431, 1993.

[15] TAKAYUKI GUNJI, SUNYOUNG KIM, MASAKAZU KOJIMA, AKIKO TAKEDA, KATSUKI FUJISAWA, AND TOMOHIKO MIZUTANI, *PHoM – A Polyhedral Homotopy Continuation Method for Polynomial Systems*, Research Report B-386, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, Meguro, Tokyo 152-8552, Japan, December 2002,

[16] V. KARAMCHETI, C. LI, I. PECHTCHANSKI, AND C. YAP. A CORE library for robust numeric and geometric computation. In *15th ACM Symp. on Computational Geometry*, 1999.

[17] CHEN LIANG, B. MOURRAIN, AND J.P. PAVONE. Subdivision methods for 2d and 3d implicit curves. In *Computational Methods for Algebraic Spline Surfaces.* Springer-Verlag, 2006 (to appear).

[18] T. LICKTEIG AND M.-F. ROY. Sylvester-habicht sequences and fast cauchy index computations. *J. of Symbolic Computation*, **31**:315–341, 2001.

[19] H. LOMBARDI, M.-F. ROY, AND M. SAFEY EL DIN. New structure theorems for subresultants. *J. of Symbolic Computation*, **29**:663–690, 2000. Special Issue Symbolic Computation in Algebra, Analysis, and Geometry.

[20] V.J. MILENKOVIC AND E. SACKS. *An approximate arrangement algorithm for semi-algebraic curves.* Proceedings of the 22th Annual Symposium on Computational Geometry, ACM, pages 237–245, June 2006.

[21] B. MOURRAIN. Computing isolated polynomial roots by matrix methods. *J. of Symbolic Computation, Special Issue on Symbolic-Numeric Algebra for Polynomials*, **26**(6):715–738, Dec. 1998.

[22] B. MOURRAIN. A new criterion for normal form algorithms. In M. Fossorier, H. Imai, Shu Lin, and A. Poli, editors, *Proc. AAECC*, Vol. **1719** of *LNCS*, pages 430–443. Springer, Berlin, 1999.

[23] B. MOURRAIN. *Pythagore's dilemma, Symbolic-Numeric Computation and the Border Basis Method*, pages 223–243. Mathematics and Visualisation. Birkhäuser, 2006.

[24] B. MOURRAIN AND J.-P. PAVONE. Subdivision methods for solving polynomial equations. Technical Report 5658, INRIA Sophia-Antipolis, 2005.

[25] B. MOURRAIN AND PH. TRÉBUCHET. Solving projective complete intersection faster. In C. Traverso, editor, *Proc. Intern. Symp. on Symbolic and Algebraic Computation*, pages 231–238. New-York, ACM Press., 2000.

[26] B. MOURRAIN AND PH. TRÉBUCHET. Generalised normal forms and polynomial system solving. In M. Kauers, editor, *Proc. Intern. Symp. on Symbolic and Algebraic Computation*, pages 253–260. New-York, ACM Press., 2005.

[27] TAKESHI OGITA, SHIN'ICHI OISHI. *Fast Inclusion of Interval Matrix Multiplication.* Reliable Computing **11**(3):191–205 (2005)

[28] J.P. PAVONE. *Auto-intersection de surfaces pamatrées réelles.* PhD thesis, Université de Nice Sophia-Antipolis, 2004.

[29] F. ROUILLIER AND P. ZIMMERMANN. Efficient isolation of a polynomial real roots. *Journal of Computational and Applied Mathematics*, **162**(1):33–50, 2003.

[30] S. RUMP, *Computational error bounds for multiple or nearly multiple eigenvalues*, Linear Algebra and its Applications, **324** (2001), pp. 209–226.

[31] E.C. SHERBROOKE AND N.M. PATRIKALAKIS. Computation of the solutions of nonlinear polynomial systems. *Comput. Aided Geom. Design*, **10**(5): 379–405, 1993.

[32] H.J. STETTER. *Numerical polynomial algebra.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2004.

[33] PH. TRÉBUCHET. *Vers une résolution stable et rapide des équations algébriques.* PhD thesis, Université Pierre et Marie Curie, 2002.

[34] J. VERSCHELDE. *Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation.* ACM Transactions on Mathematical Software **25**(2):251–276, 1999

[35] J. VON ZUR GATHEN AND J. GERHARD. *Modern computer algebra.* Cambridge University Press, New York, 1999.

[36] C.K. YAP. *Fundamental Problems in Algorithmic Algebra.* Oxford University Press, 2000. ftp:/Preliminary/cs.nyu.edu/pub/local/yap/algebra-bk.html.

TROPICAL IMPLICITIZATION AND MIXED FIBER POLYTOPES

BERND STURMFELS* AND JOSEPHINE YU†

Abstract. The software TrIm offers implementations of tropical implicitization and tropical elimination, as developed by Tevelev and the authors. Given a polynomial map with generic coefficients, TrIm computes the tropical variety of the image. When the image is a hypersurface, the output is the Newton polytope of the defining polynomial. TrIm can thus be used to compute mixed fiber polytopes, including secondary polytopes.

Key words. Elimination theory, fiber polytope, implicitization, mixed volume, Newton polytope, tropical algebraic geometry, secondary polytope.

AMS(MOS) subject classifications. 14Q10, 52B20, 52B55, 65D18.

1. Introduction. Implicitization is the problem of transforming a given parametric representation of an algebraic variety into its implicit representation as the zero set of polynomials. Most algorithms for elimination and implicitization are based on multivariate resultants or Gröbner bases, but current implementations of these methods are often too slow. When the variety is a hypersurface, the coefficients of the implicit equation can also be computed by way of numerical linear algebra [3, 7], provided the Newton polytope of that implicit equation can be predicted a priori.

The problem of predicting the Newton polytope was recently solved independently by three sets of authors, namely, by Emiris, Konaxis and Palios [8], Esterov and Khovanskii [13], and in our joint papers with Tevelev [18, 19]. A main conclusion of these papers can be summarized as follows: *The Newton polytope of the implicit equation is a mixed fiber polytope.*

The first objective of the present article is to explain this conclusion and to present the software package `TrIm` for computing such mixed fiber polytopes. The name of our program stands for *Tropical Implicitization*, and it underlines our view that the prediction of Newton polytopes is best understood within the larger context of tropical algebraic geometry. The general theory of tropical elimination developed in [18] unifies earlier results on discriminants [4] and on generic polynomial maps whose images can have any codimension [19]. The second objective of this article is to explain the main results of tropical elimination theory and their implementation in `TrIm`. Numerous hands-on examples will illustrate the use of the software. At various places we give precise pointers to [8] and [13], so as to highlight similarities and differences among the different approaches to the subject.

Our presentation is organized as follows. In Section 2 we start out with a quick guide to `TrIm` by showing some simple computations. In Section

*University of California, Berkeley, CA 94720 (bernd@math.berkeley.edu).

†Massachusetts Institute of Technology, Cambridge, MA 02139 (jyu@math.mit.edu).

3 we explain mixed fiber polytopes. That exposition is self-contained and may be of independent interest to combinatorialists. In Section 4 we discuss the computation of mixed fiber polytopes in the context of elimination theory, and in Section 5 we show how the tropical implicitization problem is solved in `TrIm`. Theorem 5.1 expresses the Newton polytope of the implicit equation as a mixed fiber polytope. In Section 6 we present results in tropical geometry on which the development of `TrIm` is based, and we explain various details concerning our algorithms and their implementation.

2. How to use `TrIm`. The first step is to download `TrIm` from the following website which contains information for installation in `Linux`:

`http://math.mit.edu/~jyu/TrIm`

`TrIm` is a collection of C++ programs which are glued together and integrated with the external software `polymake` [9] using `perl` scripts. The language `perl` was chosen for ease of interfacing between various programs.

The fundamental problem in tropical implicitization is to compute the Newton polytope of a hypersurface which is parametrized by Laurent polynomials with sufficiently generic coefficients. As an example we consider the following three Laurent polynomials in two unknowns x and y with sufficiently generic coefficients $\alpha_1, \alpha_2, \alpha_3, \beta_1, \beta_2, \beta_3, \gamma_1, \gamma_2, \gamma_3$:

$$\begin{aligned} u &= \alpha_1 \cdot \frac{1}{x^2y^2} + \alpha_2 \cdot x + \alpha_3 \cdot xy \\ v &= \beta_1 \cdot x^2 + \beta_2 \cdot y + \beta_3 \cdot \frac{1}{x} \\ w &= \gamma_1 \cdot y^2 + \gamma_2 \cdot \frac{1}{xy} + \gamma_3 \cdot \frac{1}{y}. \end{aligned}$$

We seek the unique (up to scaling) irreducible polynomial $F(u, v, w)$ which vanishes on the image of the corresponding morphism $(\mathbb{C}^*)^2 \to \mathbb{C}^3$. Using our software `TrIm`, the Newton polytope of the polynomial $F(u, v, w)$ can be computed as follows. We first create a file `input` with the contents

```
[x,y]
[x^(-2)*y^(-2) + x + x*y,
 x^2 + y + x^(-1),
 y^2 + x^(-1)*y^(-1) + y^(-1)]
```

Here the coefficients are suppressed: they are tacitly assumed to be generic. We next run a `perl` script using the command `./TrIm.prl input`. The output produced by this program call is quite long. It includes the lines

```
VERTICES
1  9 2 0
1  0 9 2
1  0 9 0
```

```
1  0 0 9
1  6 6 0
1  2 2 8
1  0 6 6
1  2 8 2
1  6 0 6
1  0 0 0
1  9 0 0
1  2 0 9
1  8 2 2
```

Ignoring the initial 1, this list consists of 13 lattice points in $\mathbb{R}^3$, and these are precisely the vertices of the Newton polytope of $F(u, v, w)$. The above output format is compatible with the polyhedral software `Polymake` [9]. We find that the Newton polytope has 10 facets, 21 edges, and 13 vertices. Further down in the output, `TrIm` prints a list of all lattice points in the Newton polytope, and it ends by telling us the number of lattice points:

```
N_LATTICE_POINTS
383
```

Each of the 383 lattice points (i, j, k) represents a monomial $u^i v^j w^k$ which might occur with non-zero coefficient in the expansion of $F(u, v, w)$. Hence, to recover the coefficients of $F(u, v, w)$ we must solve a linear system of 382 equations with 383 unknowns. Interestingly, in this example, 39 of the 383 monomials always have coefficient zero in $F(u, v, w)$. Even when $\alpha_1, \ldots, \gamma_3$ are completely generic, the number of monomials in $F(u, v, w)$ is only 344.

The command `./Trim.prl` implements a certain algorithm, to be described in the next sections, whose input consists of n lattice polytopes in $\mathbb{R}^{n-1}$ and whose output consists of one lattice polytope in $\mathbb{R}^n$. In our example, with $n = 3$, the input consists of three triangles and the output consisted of a three-dimensional polytope. These are depicted in Figure 1.

The program also works in higher dimensions but the running time quickly increases. For instance, consider the hypersurface in $\mathbb{C}^4$ represented by the following four Laurent polynomials in x, y, z, written in `TrIm` format:

```
[x,y,z]
[x*y + z + 1,
 x*z + y + 1,
 y*z + x + 1,
 x^3 + y^5 + z^7]
```

It takes `TrIm` a few moments to inform us that the Newton polytope of this hypersurface has 40 vertices and contains precisely 5026 lattice points. The f-vector of this four-dimensional polytope equals $(40, 111, 103, 32)$.

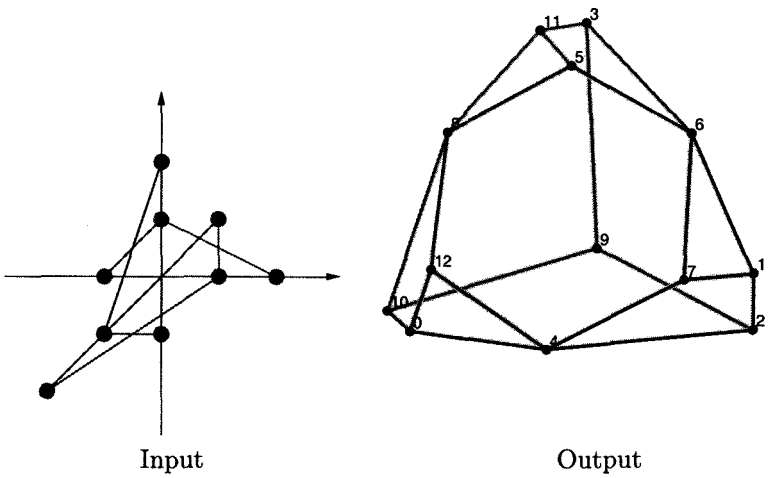


FIG. 1. *Tropical implicitization constructs the three-dimensional Newton polytope of a parametrized surface from the three Newton polygons of the given parametrization.*

REMARK 2.1. The examples above may serve as illustrations for the results in the papers [8] and [13]. Emiris, Konaxis and Palios [8] place the emphasis on computational complexity, they present a precise formula for plane parametric curves, and they allow for the map to given by rational functions. Esterov and Khovanskii develop a general theory of polyhedral elimination, which parallels the tropical approach in [18], and which includes implicitization as a very special case. A formula for the leading coefficients of the implicit equation is given in [8, §4]. This formula is currently not implemented in `TrIm` but it could be added in a future version.

What distinguishes `TrIm` from the approaches in [8] and [13] is the command `TrCI` which computes the tropical variety of a generic complete intersection. The relevant mathematics will be reviewed in Section 6. This command is one of the ingredients in the implementation of tropical implicitization. The input again consists of m Laurent polynomials in n variables whose coefficients are tacitly assumed to be generic, or, equivalently, of m lattice polytopes in n-space. Here it is assumed that $m \leq n$. If equality holds then the program simply computes the mixed volume of the given polytopes. As an example, delete the last line from the previous input file:

```
[x,y,z]
[x*y + z + 1,
 x*z + y + 1,
 y*z + x + 1]
```

The command `./TrCI.prl input` computes the mixed volume of the three given lattice polytopes in $\mathbb{R}^3$. Here the given polytopes are triangles. The last line in the output shows that their mixed volume equals five.

Now repeat the experiment with the input file `input` as follows:

```
[x,y,z]
[x*y + z + 1,  x*z + y + 1]
```

The output is a one-dimensional tropical variety given by five rays in $\mathbb{R}^3$:

```
DIM
1

RAYS
 0 -1 -1
 0  0  1
 1  0  0
 0  1  0
-1  1  1

MAXIMAL_CONES
0
1
2
3
4

MULTIPLICITIES
2
1
1
1
1
```

Note that the first ray, here indexed by 0, has multiplicity two. This scaling ensures that the sum of the five `RAYS` equals the zero vector $(0, 0, 0)$.

For a more interesting example, let us tropicalize the complete intersection of two generic hypersurfaces in $\mathbb{C}^5$. We prepare `input` as follows:

```
[a,b,c,d,e]
[a*b + b*c + c*d + d*e + a*e + 1,
 a*b*c + b*c*d + c*d*e + d*e*a + e*a*b]
```

When applied to these two polynomials in five unknowns, the command `./TrCI.prl input` produces a three-dimensional fan in $\mathbb{R}^5$. This fan has 26 rays and it has 60 maximal cones. Each maximal cone is the cone over a triangle or a quadrangle, and it has multiplicity one. The rays are

```
RAYS
-1  1  0  0  1
-1  1  1 -1  3
 0  1  0  0  1
-1  3 -1  1  1
 ...  ...  ...
```

The rays are labeled $0, 1, \ldots, 25$, in the order in which they were printed. The maximal (three-dimensional) cones appear output in the format

```
MAXIMAL_CONES
0 1 2 3
0 1 7 10
0 1 12
0 3 4 7
0 3 12
0 7 12
... ...
```

It is instructive to compute the tropical intersection of two generic hypersurfaces with the same support. For example, consider the input file

```
[x,y,z]
[1 + x + y + z + x*y + x*z + y*z + x*y*z,
 1 + x + y + z + x*y + x*z + y*z + x*y*z]
```

As before, the reader should imagine that the coefficients are generic rational numbers instead of one's. The tropical complete intersection determined by these two equations consists of the six rays normal to the six facets of the given three-dimensional cube. The same output would be produced by Jensen's software `GFan` [2, 12], which computes arbitrary tropical varieties, provided we input the two equations with generic coefficients.

3. Mixed fiber polytopes. We now describe the construction of mixed fiber polytopes. These generalize ordinary fiber polytopes [1], and hence they generalize secondary polytopes [10]. The existence of mixed fiber polytopes was predicted by McDonald [14] and Michiels and Cools [16] in the context of polynomial systems solving. They were first constructed by McMullen [15], and later independently by Esterov and Khovanskii [13].

The presentation in this section is written entirely in the language of combinatorial geometry, and it should be of independent interest to some of the readers of Ziegler's text book [20]. There are no polynomials or varieties in this section, neither classical nor tropical. The connection to elimination and tropical geometry will be explained in subsequent sections.

Consider a linear map $\pi : \mathbb{R}^p \to \mathbb{R}^q$ and a p-dimensional polytope $P \subset \mathbb{R}^p$ whose image $Q = \pi(P)$ is a q-dimensional polytope in $\mathbb{R}^q$. If x is any point in the interior of Q then its fiber $\pi^{-1}(x) \cap P$ is a polytope of dimension $p - q$. The *fiber polytope* is defined as the Minkowski integral

$$\Sigma_\pi(P) \quad = \quad \int_Q (\pi^{-1}(x) \cap P)\, dx. \tag{3.1}$$

It was shown in [1] that this integral defines a polytope of dimension $p-q$. The fiber polytope $\Sigma_\pi(P)$ lies in an affine subspace of $\mathbb{R}^p$ which is a parallel translate of kernel(π). Billera and Sturmfels [1] used the notation $\Sigma(P,Q)$ for the fiber polytope, and they showed that its faces are in bijection with the coherent polyhedral subdivisions of Q which are induced from the boundary of P. We here prefer the notation $\Sigma_\pi(P)$ over the notation $\Sigma(P,Q)$, so as to highlight the dependence on π for fixed P and varying π.

EXAMPLE 3.1. Let $p = 3$ and take P to be the standard 3-cube

$$P \quad = \quad \text{conv}\{(000),(001),(010),(011),(100),(101),(110),(111)\}.$$

We also set $q = 1$ and we fix the linear map

$$\pi : \mathbb{R}^3 \to \mathbb{R}^1,\ (u,v,w) \mapsto u + 2v + 3w.$$

Then $Q = \pi(P)$ is the line segment $[0,6]$. For $0 < x < 6$, each fiber $\pi^{-1}(x) \cap P$ is either a triangle, a quadrangle or a pentagon. Since the fibers have a fixed normal fan over each open segment $(i, i+1)$, we find

$$\Sigma_\pi(P) \quad = \quad \sum_{i=0}^{5} \int_i^{i+1} (\pi^{-1}(x) \cap P)\, dx \quad = \quad \sum_{i=0}^{5} (\pi^{-1}(i+\frac{1}{2}) \cap P).$$

Hence the fiber polygon is really just the Minkowski sum of two triangles, two quadrangles and two pentagon, and this turns out to be a hexagon:

$$\Sigma_\pi(P) \quad = \quad \text{conv}\{(1,10,5),(1,4,9),(5,2,9),(11,2,7),(11,8,3),(7,10,3)\}.$$

In the next section we shall demonstrate how `TrIm` can be used to compute fiber polytopes. The output produced will be the planar hexagon which is gotten from the coordinates above by applying the linear map $(u,v,w) \mapsto (w-3, v+w-9)$. Hence `TrIm` produces the following coordinatization:

$$\Sigma_\pi(P) \quad = \quad \text{conv}\{(2,6),\ (6,4),\ (6,2),\ (4,0),\ (0,2),\ (0,4)\}. \tag{3.2}$$

It is no big news to polytope aficionados that the fiber polygon of the 3-cube is a hexagon. Indeed, by [20, Example 9.8], the fiber polytope obtained by projecting the p-dimensional cube onto a line is the *permutohedron* of dimension $p-1$. For $p = 3$ the vertices of the hexagon $\Sigma_\pi(P)$ correspond to the six monotone edge paths on the 3-cube from (000) to (111). □

As a special case of the construction of fiber polytopes we get the secondary polytopes. Suppose that P is a polytope with n vertices in $\mathbb{R}^p$ and let Δ denote the standard $(n-1)$-simplex in $\mathbb{R}^n$. There exists a linear map $\rho : \mathbb{R}^n \to \mathbb{R}^p$ such that $\rho(\Delta) = P$, and this linear map is unique if we prescribe a bijection from the vertices of Δ onto the vertices of P. The

polytope $\Sigma_\rho(\Delta)$ is called the *secondary polytope* of P; see [20, Definition 9.9]. Secondary polytopes were first introduced in an algebraic context by Gel'fand, Kapranov and Zelevinsky [10]. For example, if we take P to be the 3-dimensional cube as above, then the simplex Δ is 7-dimensional, and the secondary polytope $\Sigma_\rho(\Delta)$ is a 4-dimensional polytope with 74 vertices. These vertices are in bijection with the 74 triangulations of the 3-cube.

A detailed introduction to triangulations and a range of methods for computing secondary polytopes can be found in the forthcoming book [5]. We note that the computation of fiber polytopes can in principle be reduced to the computation of secondary polytopes, by means of the formula

$$\Sigma_\pi(\rho(\Delta)) \;=\; \rho(\Sigma_{\pi\circ\rho}(\Delta)). \tag{3.3}$$

Here $\pi \circ \rho$ is the composition of the following two linear maps of polytopes:

$$\Delta \xrightarrow{\rho} P \xrightarrow{\pi} Q.$$

The formula (3.3) appears in [1, Lemma 2.3] and in [20, Exercise 9.6]. The algorithm of Emiris *et al.* [8, §4] for computing Newton polytopes of specialized resultants is based on a variant of (3.3). Neither our software `TrIm` nor the Esterov-Khovanskii construction [13] uses the formula (3.3).

We now come to the main point of this section, namely, the construction of *mixed fiber polytopes*. This is primarily due to McMullen [15], but was rediscovered in the context of elimination theory by Khovanskii and Esterov [13, §3]. We fix a linear map $\pi : \mathbb{R}^p \to \mathbb{R}^q$ as above, but we now consider a collection of c polytopes $P_1, \ldots, P_c$ in $\mathbb{R}^p$. We consider the Minkowski sum $P_\lambda = \lambda_1 P_1 + \cdots + \lambda_c P_c$ where $\lambda = (\lambda_1, \ldots, \lambda_c)$ is a parameter vector of unspecified positive real numbers. We shall assume that P_λ is of full dimension p, but we do allow its summands P_i to be lower-dimensional. The image of P_λ under the map π is the q-dimensional polytope

$$\pi(P_\lambda) \;=\; \lambda_1 \cdot \pi(P_1) + \cdots + \lambda_c \cdot \pi(P_c).$$

The following result concerns the fiber polytope from P_λ onto $\pi(P_\lambda)$.

THEOREM 3.2 ([15, 13]). *The fiber polytope $\Sigma_\pi(P_\lambda)$ depends polynomially on the parameter vector λ. This polynomial is homogeneous of degree $q+1$. Moreover, there exist unique polytopes $M_{i_1 i_2 \cdots i_c}$ such that*

$$\Sigma_\pi(\,\lambda_1 P_1 + \cdots + \lambda_c P_c\,) \;=\; \sum_{i_1+\cdots+i_c=q+1} \lambda_1^{i_1}\lambda_2^{i_2}\cdots\lambda_c^{i_c} \cdot M_{i_1 i_2 \cdots i_c}. \tag{3.4}$$

To appreciate this theorem, it helps to begin with the case $c = 1$. That corresponds to scaling the polytopes P and Q above by the same factor λ. This results in the Minkowski integral (3.1) being scaled by the factor λ^{q+1}. More generally, the coefficients of the pure powers λ_j^{q+1} in the expansion (3.4) are precisely the fiber polytopes of the individual P_j, that is,

$$M_{0,\ldots,0,q+1,0,\ldots,0} \;=\; \Sigma_\pi(P_j).$$

On the other extreme, we may consider $i_1 = i_2 = \cdots = i_c = 1$, which is the term of interest for elimination theory. Of course, if all i_j's are equal to one then the number c of polytopes P_j is one more than the dimension q of the image of π. We now assume that this holds, i.e., we assume that $c = q+1$. We define the *mixed fiber polytope* to be the coefficient of the monomial $\lambda_1\lambda_2\cdots\lambda_c$ in the formula (3.4). The mixed fiber polytope is denoted

$$\Sigma_\pi(P_1, P_2, \ldots, P_c) \quad := \quad M_{11\cdots 1}. \tag{3.5}$$

The smallest non-trivial case arises when $p = 3$, $c = 2$ and $q = 1$, where we are projecting two polytopes P_1 and P_2 in $\mathbb{R}^3$. Their mixed fiber polytope with respect to a linear form $\pi : \mathbb{R}^3 \to \mathbb{R}^1$ is the coefficient of $\lambda_1\lambda_2$ in

$$\Sigma_\pi(\lambda_1 P_1 + \lambda_2 P_2) \quad = \quad \lambda_1^2 \cdot \Sigma_\pi(P_1) + \lambda_1\lambda_2 \cdot \Sigma_\pi(P_1, P_2) + \lambda_2^2 \cdot \Sigma_\pi(P_2).$$

The following is [18, Example 4.10]. It will be revisited in Example 4.2.

EXAMPLE 3.3. Consider the following two tetrahedra in three-space:

$$P_1 = \operatorname{conv}\{0, 3e_1, 3e_2, 3e_3\} \text{ and } P_2 = \operatorname{conv}\{0, -2e_1, -2e_2, -2e_3\}.$$

Their Minkowski sum $P_1 + P_2$ has 12 vertices, 24 edges and 14 facets. If we take $\pi : \mathbb{R}^3 \to \mathbb{R}^1$ to be the linear form $(u, v, w) \mapsto u - 2v + w$ then the fiber polytope $\Sigma_\pi(P_1 + P_2) = M_{20} + M_{11} + M_{02}$ is a polygon with ten vertices. Its summands $M_{20} = \Sigma_\pi(P_1)$ and $M_{02} = \Sigma_\pi(P_2)$ are quadrangles, while the mixed fiber polytope $M_{11} = \Sigma_\pi(P_1, P_2)$ is a hexagon. □

We remark that fiber polytopes are special instances of mixed fiber polytopes. Suppose that $P_1 = P_2 = \cdots = P_c$ are all equal to the same fixed polytope P in $\mathbb{R}^p$. Then the fiber polytope $\Sigma_\pi(P_\lambda)$ in (3.4) equals

$$\Sigma(\lambda_1 P_1 + \cdots + \lambda_c P_c) \quad = \quad (\lambda_1 + \cdots + \lambda_c)^c \cdot \Sigma_\pi(P).$$

Hence the fiber polytope $\Sigma_\pi(P)$ is the mixed fiber polytope $\Sigma_\pi(P, \ldots, P)$ scaled by a factor of $1/c!$. Similarly, any of the coefficients in the expansion (3.4) can be expressed as mixed fiber polytopes. Up to scaling, we have

$$M_{i_1 i_2 \cdots i_c} \quad = \quad \Sigma_\pi\big(\underbrace{P_1, \ldots, P_1}_{i_1 \text{ times}}, \underbrace{P_2, \ldots, P_2}_{i_2 \text{ times}}, \ldots, \underbrace{P_c, \ldots, P_c}_{i_c \text{ times}}\big).$$

In the next section we shall explain how mixed fiber polytopes, and hence also fiber polytopes and secondary polytopes, can be computed using `TrIm`.

4. Elimination. Let $f_1, f_2, \ldots, f_c \in \mathbb{C}[x_1^{\pm 1}, x_2^{\pm 1}, \ldots, x_p^{\pm 1}]$ be Laurent polynomials whose Newton polytopes are $P_1, P_2, \ldots, P_c \subset \mathbb{R}^p$, and suppose that the coefficients of the f_i are generic. This means that

$$f_i(x) \quad = \quad \sum_{a \in P_i \cap \mathbb{Z}^n} c_{i,a} \cdot x_1^{a_1} x_2^{a_2} \cdots x_p^{a_p},$$

where the coefficients $c_{i,a}$ are assumed to be sufficiently generic non-zero complex numbers. The corresponding variety

$$X \quad = \quad \{u \in (\mathbb{C}^*)^p \ : \ f_1(u) = f_2(u) = \cdots = f_c(u) = 0\}$$

is a complete intersection of codimension c in the algebraic torus $(\mathbb{C}^*)^p$.

We set $r = p - c + 1$ and we fix an integer matrix $\mathbf{A} = (a_{ij})$ of format $r \times p$ where the rows of $\mathbf{A}$ are assumed to be linearly independent. We also let $\pi : \mathbb{R}^p \to \mathbb{R}^{c-1}$ be any linear map whose kernel equals the row space of $\mathbf{A}$. The matrix $\mathbf{A}$ induces the following monomial map:

$$\alpha : (\mathbb{C}^*)^p \to (\mathbb{C}^*)^{p-c+1}, \ (x_1, \ldots, x_p) \mapsto (\prod_{j=1}^{p} x_j^{a_{1j}}, \ldots, \prod_{j=1}^{p} x_j^{a_{rj}}). \quad (4.1)$$

Let Y be the closure in $(\mathbb{C}^*)^{p-c+1}$ of the image $\alpha(X)$. Then Y is a hypersurface, and we are interested in its Newton polytope. By this we mean the Newton polytope of the irreducible equation of that hypersurface.

THEOREM 4.1 (Khovanskii and Esterov [13]). *The Newton polytope of Y is affinely isomorphic to the mixed fiber polytope $\Sigma_\pi(P_1, \ldots, P_c)$.*

A proof of this result using tropical geometry is given in [18]. The computation of the hypersurface Y from the defining equations $f_1, \ldots, f_c$ of X is a key problem of elimination theory. Theorem 4.1 offers a tropical solution to this problem. It predicts the Newton polytope of Y. This information is useful for symbolic-numeric software. Knowing the Newton polytopes reduces computing the equation of Y to linear algebra.

The numerical mathematics of this linear algebra problem is interesting and challenging, as seen in [3] and confirmed by the experiments reported in [19, §5.2]. We hope that our software `TrIm` will eventually be integrated with software exact linear algebra or numerical linear algebra (e.g. `LAPack`). Such a combination would have the potential of becoming a useful tool for practitioners of non-linear computational geometry.

In what follows, we demonstrate how `TrIm` computes the Newton polytope of Y and hence the mixed fiber polytope $\Sigma_\pi(P_1, \ldots, P_c)$. The input consists of the polytopes $P_1, \ldots, P_c$ and the matrix $\mathbf{A}$. The map π is tacitly understood as the map from $\mathbb{R}^p$ onto the cokernel of the transpose of $\mathbf{A}$.

EXAMPLE 4.2. Let $p = 3, c = 2$ and consider [18, Example 1.3]. Here the variety X is the curve in $(\mathbb{C}^*)^3$ defined by the two Laurent polynomials

$$f_1 = \alpha_1 x_1^3 + \alpha_2 x_2^3 + \alpha_3 x_3^3 + \alpha_4 \quad \text{and} \quad f_2 = \beta_1 x_1^{-2} + \beta_2 x_2^{-2} + \beta_3 x_3^{-2} + \beta_4.$$

We seek to compute the Newton polygon of the image curve Y in $(\mathbb{C}^*)^2$ where $\mathbf{A} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \end{pmatrix}$. The curve is written on a file `input` as follows:

```
[x1,x2,x3]
[x1^3 + x2^3 + x3^3 + 1, x1^(-2)+x2^(-2)+x3^(-2)+1]
```

We also prepare a second input file `A.matrix` as follows:

```
LINEAR_MAP
1 1 1
0 1 2
```

We now execute the following two commands in `TrIm`:

```
./TrCI.prl input > fan
./project.prl fan A.matrix
```

The output we obtain is the Newton polygon of the curve Y:

```
VERTICES
1 36  0
1  0 36
1 30 12
1 18 12
1  6 24
1 18 24
```

This hexagon coincides with the hexagon in [18, Examples 1.3 and 4.10]. It is isomorphic to the mixed fiber polytope $\Sigma_\pi(P_1, P_2)$ in Example 3.3. □

We may use `TrIm` to compute arbitrary fiber polytopes. For example, to carry out the computation of Example 3.1, we prepare `input` as

```
[x,y,z]
[1 + x + y + z + x*y + x*z + y*z + x*y*z,
 1 + x + y + z + x*y + x*z + y*z + x*y*z]
```

and `A.matrix` as

```
LINEAR_MAP
1  1 -1
2 -1  0
```

The two commands above now produce the hexagon in (3.2). Our next example shows how to compute secondary polytopes using `TrIm`.

EXAMPLE 4.3. Following [20, Example 9.11], we consider the hexagon with vertices $(0,0), (1,1), (2,4), (3,9), (4,16), (5,25)$. This hexagon is represented in `TrIm` by the following file `A.matrix`. The rows of this matrix span the linear relations among the five non-zero vertices of the hexagon:

```
LINEAR_MAP
 3   -3  1  0  0
 8   -6  0  1  0
15  -10  0  0  1
```

On the file `input` we take three copies of the standard 5-simplex:

```
[a,b,c,d,e]
[a+b+c+d+e+1,  a+b+c+d+e+1, a+b+c+d+e+1]
```

Running our two commands, we obtain a 3-dimensional polytope with 14 vertices, 21 edges and 9 facets. That polytope is the *associahedron* [20]. □

We close this section with another application of tropical elimination.

EXAMPLE 4.4. For two subvarieties X_1 and X_2 of $(\mathbb{C}^*)^n$ we define their *coordinate-wise product* $X_1 \star X_2$ to be the closure of the set of all points $(u_1v_1, \ldots, u_nv_n)$ where $(u_1, \ldots, u_n) \in X_1$ and $(v_1, \ldots, v_n) \in X_2$. The expected dimension of $X_1 \star X_2$ is the sum of the dimensions of X_1 and X_2, so we can expect $X_1 \star X_2$ to be a hypersurface when $\dim(X_1) + \dim(X_2) = n - 1$. Assuming that X_1 and X_2 are generic complete intersections then the Newton polytope of that hypersurface can be computed using `TrIm` as follows. Let $p = 2n$ and define X as the direct product $X_1 \times X_2$. Then $X_1 \star X_2$ is the image of X under the monomial map

$$\alpha : (\mathbb{C}^*)^{2n} \to (\mathbb{C}^*)^n, (u_1, \ldots, u_n, v_1, \ldots, v_n) \mapsto (u_1v_1, \ldots, u_nv_n).$$

Here is an example where X_1 and X_2 are curves in three-dimensional space ($n = 3$). The two input curves are specified on the file `input` as follows:

```
[u1,u2,u3, v1,v2,v3]
[u1 + u2 + u3 + 1,
 u1*u2 + u1*u3 + u2*u3 + u1 + u2 + u3,
 v1*v2 + v1*v3 + v2*v3 + v1 + v2 + v3 + 1,
 v1*v2*v3 + v1*v2 + v1*v3 + v2*v3 + v1 + v2 + v3]
```

The multiplication map $\alpha : (\mathbb{C}^*)^3 \times (\mathbb{C}^*)^3 \to (\mathbb{C}^*)^3$ is specified on `A.matrix`:

```
LINEAR_MAP
1 0 0 1 0 0
0 1 0 0 1 0
0 0 1 0 0 1
```

The image of $X_1 \times X_2$ under the map α is the surface $X_1 \star X_2$. We find that the Newton polytope of this surface has ten vertices and seven facets:

```
VERTICES
1 8 4 0
1 0 8 4
1 0 8 0
1 0 0 8
1 4 8 0
1 4 0 8
1 0 4 8
```

```
1 8 0 0
1 0 0 0
1 8 0 4

FACETS
128 -16   0   0
  0   0   0  32
128   0 -16   0
  0  32   0   0
  0   0  64   0
128   0   0 -16
192 -16 -16 -16
```

5. Implicitization. Implicitization is a special case of elimination. Suppose we are given n Laurent polynomials $g_1, \ldots, g_n$ in $\mathbb{C}[t_1^{\pm 1}, \ldots, t_{n-1}^{\pm 1}]$ which have Newton polytopes $Q_1, \ldots, Q_n \subset \mathbb{R}^{n-1}$ and whose coefficients are generic complex numbers. These data defines the morphism

$$g : (\mathbb{C}^*)^{n-1} \to (\mathbb{C}^*)^n, \, t \mapsto \big(g_1(t), \ldots, g_{n-1}(t)\big). \tag{5.1}$$

Under mild hypotheses, the closure of the image of g is a hypersurface Y in $(\mathbb{C}^*)^n$. Our problem is to compute the Newton polytope of this hypersurface. A first example of how this is done in `TrIm` was shown in the beginning of Section 2, and more examples will be featured in this section.

The problem of implicitization is reduced to the elimination computation in the previous section as follows. We introduce n new variables $y_1, \ldots, y_n$ and we consider the following n auxiliary Laurent polynomials:

$$f_1(x) = g_1(t) - y_1, \;\; f_2(x) = g_2(t) - y_2, \;\; \ldots, \; f_n(x) = g_n(t) - y_n. \tag{5.2}$$

Here we set $p = 2n-1$ and $(x_1, \ldots, x_p) = (t_1, \ldots, t_{n-1}, y_1, \ldots, y_n)$ so as to match the earlier notation. The subvariety of $(\mathbb{C}^*)^p = (\mathbb{C}^*)^{n-1} \times (\mathbb{C}^*)^n$ defined by $f_1, \ldots, f_n$ is a generic complete intersection of codimension n, namely, it is the graph of the map g. The image of g is obtained by projecting the variety $\{f_1 = \cdots = f_n = 0\}$ onto the last n coordinates. This projection is the monomial map α specified by the $n \times p$-matrix $A = (\mathbf{0}\ I)$ where $\mathbf{0}$ is the $n \times (n-1)$ matrix of zeroes and I is the $n \times n$ identity matrix.

This shows that we can solve the implicitization problem by doing the same calculation as in the previous section. Since that calculation is a main application of `TrIm`, we have hard-wired it in the command `./TrIm.prl`. Here is an example that illustrates the advantage of using tropical implicitization in analyzing parametric surfaces of high degree in three-space.

EXAMPLE 5.1. Consider the parametric surface specified by the `input`

```
[x,y]
[x^7*y^2 + x*y + x^2*y^7 + 1,
```

```
x^8*y^8 + x^3*y^4 + x^4*y^3 + 1,
x^6*y + x*y^6 + x^3*y^2 + x^2*y^3 + x + y]
```

Using the technique shown in Section 2, we learn in a few seconds to learn that the irreducible equation of this surface has degree 90. The command `./TrIm.prl input` reveals that its Newton polytope has six vertices

```
VERTICES
1  80  0  0
1   0 45  0
1   0  0 80
1   0 10 80
1   0  0  0
1  28  0 54
```

This polytope also has six facets, namely four triangles and two quadrangles. The expected number of monomials in the implicit equation equals

```
N_LATTICE_POINTS
62778
```

At this point the user can make an informed choice as to whether she wishes to attempt solving for the coefficients using numerical linear algebra. □

Returning to our polyhedral discussion in Section 3, we next give a conceptual formula for the Newton polytope of the implicit equation as a mixed fiber polytope. The given input is a list of n lattice polytopes $Q_1, Q_2, \ldots, Q_n$ in $\mathbb{R}^{n-1}$. Taking the direct product of $\mathbb{R}^{n-1}$ with the space $\mathbb{R}^n$ with standard basis $\{e_1, e_2, \ldots, e_n\}$, we consider the auxiliary polytopes

$$Q_1 \times \{e_1\}\,,\ Q_2 \times \{e_2\}\,,\ \ldots\,,\ Q_n \times \{e_n\} \ \subset\ \mathbb{R}^{n-1} \times \mathbb{R}^n.$$

These are the Newton polytopes of the equations $f_1, f_2, \ldots, f_n$ in (5.2). We now define π to be the projection onto the first $n-1$ coordinates

$$\pi : \mathbb{R}^{n-1} \times \mathbb{R}^n \to \mathbb{R}^{n-1},\ (u_1, \ldots, u_{n-1}, v_1, v_2, \ldots, v_n) \mapsto (u_1, \ldots, u_{n-1}).$$

The following result is an immediate corollary to Theorem 4.1. We propose that it be named the *Fundamental Theorem of Tropical Implicitization.*

THEOREM 5.2. *The Newton polytope of an irreducible hypersurface in* $(\mathbb{C}^*)^n$ *which is parametrically represented by generic Laurent polynomials with given Newton polytopes* $Q_1, \ldots, Q_n$ *equals the mixed fiber polytope*

$$\Sigma_\pi\big(Q_1 \times \{e_1\}, Q_2 \times \{e_2\}, \ldots, Q_n \times \{e_n\}\big) \tag{5.3}$$

This theorem is the geometric characterization of the Newton polytope of the implicit equation, and it summarizes the essence of the recent progress obtained by Emiris, Konaxis and Palios [8], Esterov and Khovanskii [13], and Sturmfels, Tevelev and Yu [18, 19]. Our implementation of in

TrIm computes the mixed fiber polytope (5.3) for any given $Q_1, Q_2, \ldots, Q_n$, and it suggests that the Fundamental Theorem of Tropical Implicitization will be a tool of considerable practical value for computational algebra.

EXAMPLE 5.3. We consider a threefold in $\mathbb{C}^4$ which is parametrically represented by four trivariate polynomials. On the file input we write

```
[x,y,z]
[x + y + z + 1,
 x^2*z + y^2*x + z^2*y + 1,
 x^2*y + y^2*z + z^2*x + 1,
 x*y + x*z + y*z + x + y + z]
```

The Newton polytope of this threefold has the f-vector $(8, 16, 14, 6)$, and it contains precisely 619 lattice points. The eight vertices among them are

```
VERTICES
1 15 0 0 0
1  0 6 0 0
1  0 0 0 9
1  0 0 6 0
1  0 0 0 0
1 12 0 0 3
1  9 3 0 0
1  9 0 3 0
```

This four-dimensional polytope is the mixed fiber polytope (5.3) for the tetrahedra Q_1, Q_2, Q_3 and the octahedron Q_4 specified in the file input. □

In (5.1) we assumed that the $g_i(t)$ are Laurent polynomials but this hypothesis can be relaxed to other settings discussed in [8, 13]. In particular, the Fundamental Theorem of Tropical Implicitization extends to the case when the $g_i(t)$ are rational functions. Here is how this works in TrIm.

EXAMPLE 5.4. Let $\alpha_1, \ldots, \alpha_5$ and $\beta_1, \ldots, \beta_5$ be general complex numbers and consider the plane curve which has the rational parametrization

$$x = \frac{\alpha_1 t^3 + \alpha_2 t + \alpha_3}{\alpha_4 t^2 + \alpha_5} \quad \text{and} \quad y = \frac{\beta_1 t^4 + \beta_2 t^3 + \beta_3}{\beta_4 t^2 + \beta_5}. \tag{5.4}$$

This curve appears in [8, Example 4.7]. The input for TrIm is as follows:

```
[ t, x, y ]
[ t^3 + t + 1   +  x*t^2 + x ,
  t^4 + t^3 + 1  +  y*t^2 + y ]
```

The equation of the plane curve is gotten by eliminating the unknown t from the two equations (5.4). Tropical elimination using TrIm predicts that the Newton polygon of that plane curve is the following pentagon:

```
POINTS
1 4 2
1 0 3
1 2 3
1 0 0
1 4 0
```

This prediction is the correct Newton polygon for generic coefficients α_i and β_j. In particular, generically the pairs of coefficients (α_4, α_5) and (β_4, β_5) in the denominators are distinct. If we assume, however, that the denominators are the same, then the correct Newton polytope is a quadrangle inside this pentagon, as computed in [8, Example 5.6]. □

6. Tropical varieties. We now explain the mathematics on which `TrIm` is based. The key idea is to embed the study of Newton polytopes into the context of tropical geometry [2, 4, 18, 19]. Let I be any ideal in the Laurent polynomial ring $\mathbb{C}[x_1^{\pm 1}, \ldots, x_p^{\pm 1}]$. Then its *tropical variety* is

$$\mathcal{T}(I) \;=\; \bigl\{w \in \mathbb{R}^p \,:\, \mathrm{in}_w(I) \text{ does not contain a monomial}\bigr\}.$$

Here $\mathrm{in}_w(I)$ is the ideal of $\mathbb{C}[x_1^{\pm 1}, \ldots, x_p^{\pm 1}]$ which is generated by the w-initial forms of all elements in I. The set $\mathcal{T}(I)$ can be given the structure of a polyhedral fan, for instance, by restricting the Gröbner fan of any homogenization of I. A point w in $\mathcal{T}(I)$ is called *regular* if it lies in the interior of a maximal cone in some fan structure on $\mathcal{T}(I)$. Every regular point w naturally comes with a multiplicity m_w, which is a positive integer. We can define m_w as the sum of multiplicities of all minimal associate primes of the initial ideal $\mathrm{in}_w(I)$. The multiplicities m_w on $\mathcal{T}(I)$ are independent of the fan structure and they satisfy the *balancing condition* [18, Def. 3.3].

If I is a principal ideal, generated by one Laurent polynomial $f(x)$, then $\mathcal{T}(I)$ is the union of all codimension one cones in the normal fan of the Newton polytope P of $f(x)$. A point $w \in \mathcal{T}(I)$ is regular if and only if w supports an edge of P, and m_w is the lattice length of that edge. It is important to note that the polytope P can be reconstructed uniquely, up to translation, from the tropical hypersurface $\mathcal{T}(I)$ together with its multiplicities m_w. The following `TrIm` example shows how to go back and forth between the Newton polytope P and its tropical hypersurface $\mathcal{T}(I)$.

EXAMPLE 6.1. We write the following polynomial onto the file `poly`:

```
[ x, y, z ]
[ x + y + z + x^2*y^2 + x^2*z^2 + y^2*z^2 ]
```

The command `./TrCI.prl poly > fan` writes the tropical surface defined by this polynomial onto a file `fan`. That output file starts out like this:

```
AMBIENT_DIM
```

```
3

DIM
2
..........
```

The tropical surface consists of 12 two-dimensional cones on 8 rays in $\mathbb{R}^3$. Three of the 12 cones have multiplicity two, while the others have multiplicity one. Combinatorially, this surface is the edge graph of the 3-cube. The Newton polytope P is an octahedron, and it can be recovered from the data on `fan` after we place the 3×3 identity matrix in the file `A.matrix`:

```
LINEAR_MAP
1 0 0
0 1 0
0 0 1
```

Our familiar command

```
./project.prl fan A.matrix
```

now reproduces the Newton octahedron P in the familiar format:

```
VERTICES
1 2 2 0
1 0 2 2
1 0 1 0
1 2 0 2
1 1 0 0
1 0 0 1
```

Note that the edge lengths of P are the multiplicities on the tropical surface. □

The implementation of the command `project.prl` is based on the formula given in [19, Theorem 5.2]. See [4, §2] for a more general version. This result translates into an algorithm for `TrIm` which can be described as follows. Given a generic vector $w \in \mathbb{R}^k$ such that $\text{face}_w(P)$ is a vertex v, the i^{th} coordinate v_i is the number of intersections, counted with multiplicities, of the ray $w + \mathbb{R}_{>0}e_i$ with the tropical variety. Here, the multiplicity of the intersection with a cone Γ is the multiplicity of Γ times the absolute value of the i^{th} coordinate of the primitive normal vector to the cone Γ.

Intuitively, what we are doing in our software is the following. We wish to determine the coordinates of the extreme vertex $v = \text{face}_w(P)$ of a polytope P in a given direction w. Our polytope is placed so that it lies in the positive orthant and touches all the coordinate hyperplanes. To compute the i^{th} coordinate of the vertex v, we can walk from v toward the i^{th} hyperplane along the edges of P, while keeping track of the edge lengths

in the i^{th} direction. A systematic way to carry out the walk is to follow the edges whose inner normal cone intersect the ray $w + \mathbb{R}_{>0}e_i$. Recall that the multiplicity of a codimension one normal cone is the lattice length of the corresponding edge. Using this subroutine for computing extreme vertices, the whole polytope is now constructed using the method of Huggins [11].

To compute the tropical variety $\mathcal{T}(I)$ for an arbitrary ideal I one can use the Gröbner-based software `GFan` due to Jensen [2, 12]. Our polyhedral software `TrIm` performs the same computation faster when the generators of I are Laurent polynomials $f_1, f_2, \ldots, f_c$ that are generic relative to their Newton polytopes $P_1, P_2, \ldots, P_c$. It implements the following combinatorial formula for the tropical variety $\mathcal{T}(I)$ of the complete intersection I.

THEOREM 6.2. *The tropical variety $\mathcal{T}(I)$ is supported on a subfan of the normal fan of the Minkowski sum $\sum_{i=1}^{c} P_i$. A point w is in $\mathcal{T}(I)$ if and only if the polytope $\mathrm{face}_w(\sum_{j\in J} P_j)$ has dimension $\geq |J|$ for $J \subseteq \{1, \ldots, c\}$. The multiplicity of $\mathcal{T}(I)$ at a regular point w is the mixed volume*

$$m_w \;=\; \text{mixed volume}\big(\mathrm{face}_w(P_1), \mathrm{face}_w(P_2), \ldots, \mathrm{face}_w(P_c)\big), \qquad (6.1)$$

where we normalize volume respect to the affine lattice parallel to $\sum_{j\in J} P_j$.

For a proof of this theorem see [18, §4]. We already saw some examples in the second half of Section 2. Here is one more such illustration:

EXAMPLE 6.3. We consider the generic complete intersection of codimension three in six-dimensional space $(\mathbb{C}^*)^3$ given by the polynomials

```
[a,b,c,d,e,f]
[a*b*c*d*e*f + a + b + c + d + e + f + 1,
 a*b + b*c + c*d + d*e + e*f + f*a,
 a + b + c + d + e + f + 1]
```

The application of `./TrCI.prl` to this input internally constructs the corresponding six-dimensional polytope $P_1 + P_2 + P_3$. It lists all facets and their normal vectors, it generates all three-dimensional cones in the normal fan, and it picks a representative vector w in the relative interior of each such cone. The three polytopes $\mathrm{face}_w(P_1)$, $\mathrm{face}_w(P_2)$ and $\mathrm{face}_w(P_3)$ are translated to lie in the same three-dimensional space, and their mixed volume m_w is computed. If m_w is positive then `TrIm` outputs the rays of that cone and the mixed volume m_w. The output reveals that this tropical threefold in $\mathbb{R}^6$ consists of 117 three-dimensional cones on 22 rays. □

In our implementation of Theorem 6.2 in `TrIm`, the Minkowski sum is computed using the `iB4e` library [11], enumerating the d-dimensional cones is done by `Polymake` [9], the mixed volumes are computed using the `mixed volume library` [6], and integer linear algebra for lattice indices is done using `NTL` [17]. What oversees all the computations is the `perl` script `TrCI.prl`. The output format is consistent with that of the current

version of `Gfan` [12] and is intended to interface with the polyhedral software `Polymake` [9] when it supports polyhedral complexes and fans in the future.

Elimination theory is concerned with computing the image of an algebraic variety whose ideal I we know under a morphism $\alpha : (\mathbb{C}^*)^p \to (\mathbb{C}^*)^r$. We write β for the corresponding homomorphism from the Laurent polynomial ring in r unknowns to the Laurent polynomial ring in p unknowns. Then $J = \beta^{-1}(I)$ is the ideal of the image variety, and, ideally, we would like to find generators for J. That problem is too hard, and what we do instead is to apply tropical elimination theory as follows. We assume that α is a monomial map, specified by an $r \times p$ integer matrix $\mathbf{A} = (a_{ij})$ as in (4.1). Rather than computing the variety of J from the variety of I, we instead compute the tropical variety $\mathcal{T}(J)$ from the tropical variety $\mathcal{T}(I)$. This is done by the following theorem which characterizes the multiplicities.

THEOREM 6.4. *The tropical variety $\mathcal{T}(J)$ equals the image of $\mathcal{T}(I)$ under the linear map $\mathbf{A}$. If the monomial map α induces a generically finite morphism of degree δ from the variety of I onto the variety of J then the multiplicity of $\mathcal{T}(J)$ at a regular point w is computed by the formula*

$$m_w \;=\; \frac{1}{\delta} \cdot \sum_v m_v \cdot \mathit{index}\,(\mathbb{L}_w \cap \mathbb{Z}^r : \mathbf{A}(\mathbb{L}_v \cap \mathbb{Z}^p)). \tag{6.2}$$

The sum is over all points v in $\mathcal{T}(I)$ with $\mathbf{A}v = w$. We assume that the number of these points is finite. they are all regular in $\mathcal{T}(I)$, and $\mathbb{L}_v$ is the linear span of a neighborhood of v in $\mathcal{T}(I)$, and similarly for $w \in \mathcal{T}(J)$.

The formula (6.2) can be regarded as a push-forward formula in intersection theory on toric varieties, and it constitutes the workhorse inside the `TrIm` command `project.prl`. When the tropical variety $\mathcal{T}(J)$ has codimension one, then that tropical hypersurface determines the Newton polytope of the generator of J. The transformation from tropical hypersurface to mixed fiber polytope was behind all our earlier examples. That transformation was shown explicitly for an octahedron in Example 6.1.

In all our examples so far, the ideal I was tacitly assumed to be a generic complete intersection, and Theorem 6.2 was used to determine the tropical variety $\mathcal{T}(I)$ and the multiplicities m_v. In other words, the command `TrCI` furnished the ingredients for the formula (6.2). In particular, then I is the ideal of the graph of a morphism, as in Section 5, then Theorem 6.4 specializes to the formula for tropical implicitization given in [19].

It is important to note, however, that Theorem 6.4 applies to any ideal I whose tropical variety happens to be known, even if I is not a generic complete intersection. For instance, $\mathcal{T}(I)$ might be the output of a `GFan` computation, or it might be one of the special tropical varieties which have already been described in the literature. Any tropical variety with known multiplicities can serve as the input to the `TrIm` command `project.prl`.

Acknowledgments. We are grateful to Peter Huggins for his help at many stages of the `TrIm` project. His `iB4e` software became a crucial

ingredient in our implementation. Bernd Sturmfels was partially supported by the National Science Foundation (DMS-0456960), and Josephine Yu was supported by a UC Berkeley Graduate Opportunity Fellowship and by the Institute for Mathematics and its Applications (IMA) in Minneapolis.

REFERENCES

[1] Louis Billera and Bernd Sturmfels, *Fiber polytopes*, *Annals of Mathematics* **135** (1992), 527–549.

[2] Tristram Bogart, Anders Jensen, David Speyer, Bernd Sturmfels, and Rekha Thomas, *Computing tropical varieties*, *Journal of Symbolic Computation* **42** (2007), 54–73.

[3] Robert Corless, Mark Giesbrecht, Ilias Kotsireas, and Steven Watt, *Numerical implicitization of parametric hypersurfaces with linear algebra*, in: Artificial Intelligence and Symbolic Computation, Springer Lecture Notes in Computer Science, **1930** (2000), 174–183.

[4] Alicia Dickenstein, Eva Maria Feichtner, and Bernd Sturmfels, *Tropical discriminants*, *J. Amer. Math. Soc.* **20** (2007), 1111–1133.

[5] Jesus De Loera, Jörg Rambau, and Francisco Santos, *Triangulations: Applications, Structures and Algorithms*, Algorithms and Computation in Mathematics, Springer Verlag, Heidelberg, to appear.

[6] Ioannis Emiris and John Canny, *Efficient incremental algorithms for the sparse resultant and the mixed volume*, *J. Symbolic Computation* **20** (1995), 117–150.

[7] Ioannis Emiris and Ilias Kotsireas, *Implicitization exploiting sparseness*, in D. Dutta, M. Smid, R. Janardan (eds.), Geometric And Algorithmic Aspects Of Computer-aided Design And Manufacturing, pp. 281–298, DIMACS Series in Discrete Mathematics and Theoretical Computer Science **67**, American Mathematical Society, Providence RI, 2005.

[8] Ioannis Emiris, Christos Konaxis, and Leonidas Palios, *Computing the Newton polytope of specialized resultants.* Presented at the conference MEGA 2007 (Effective Methods in Algebraic Geometry), Strobl, Austria, June 2007.

[9] Ewgenij Gawrilow and Michael Joswig, *Polymake: A framework for analyzing convex polytopes.* In *Polytopes—Combinatorics and Computation (Oberwolfach, 1997)*, *DMV Seminar*, Vol. **29**, pages 43–73. Birkhäuser, Basel, 2000.

[10] Israel Gel'fand, Mikhael Kapranov, and Andrei Zelevinsky, *Discriminants, Resultants, and Multidimensional Determinants*; Birkhäuser, Boston, 1994.

[11] Peter Huggins, *iB4e: A software framework for parametrizing specialized LP problems*, A Iglesias, N Takayama (Eds.), *Mathematical Software - ICMS 2006, Second International Congress on Mathematical Software, Castro Urdiales, Spain, September 1–3, 2006, Proceedings.* Lecture Notes in Computer Science 4151 Springer 2006 (pp. 245–247).

[12] Anders N. Jensen, *Gfan, a software system for Gröbner fans.* Available at `http://home.imf.au.dk/ajensen/software/gfan/gfan.html`.

[13] Askold Khovanskii and Alexander Esterov, *Elimination theory and Newton polyhedra*, Preprint: math.AG/0611107.

[14] John McDonald, *Fractional power series solutions for systems of equations*, *Discrete and Computational Geometry* **27** (2002), 501–529.

[15] Peter McMullen, *Mixed fibre polytopes*, *Discrete and Computational Geometry* **32** (2004), 521–532.

[16] Tom Michiels and Ronald Cools, *Decomposing the secondary Cayley polytope*, *Discrete and Computational Geometry* **23** (2000), 367–380.

[17] Victor Shoup, NTL: A Library for doing Number Theory, available at `http://www.shoup.net/index.html`.

[18] BERND STURMFELS AND JENIA TEVELEV, *Elimination theory for tropical varieties*, Preprint: math.AG/0704347.

[19] BERND STURMFELS, JENIA TEVELEV, AND JOSEPHINE YU, *The Newton polytope of the implicit equation*, *Moscow Mathematical Journal* **7** (2007), 327–346.

[20] GÜNTER ZIEGLER, *Lectures on Polytopes*, Graduate Texts in Mathematics **152**, Springer-Verlag, New York, 1995.

TOWARDS A BLACK-BOX SOLVER FOR FINITE GAMES: COMPUTING ALL EQUILIBRIA WITH GAMBIT AND PHCPACK

THEODORE L. TUROCY*

Abstract. This paper describes a new implementation of an algorithm to find all isolated Nash equilibria in a finite strategic game. The implementation uses the game theory software package Gambit to generate systems of polynomial equations which are necessary conditions for a Nash equilibrium, and polyhedral homotopy continuation via the package PHCpack to compute solutions to the systems. Numerical experiments to characterize the performance of the implementation are reported. In addition, the current and future roles of support enumeration methods in the context of methods for computing Nash equilibria are discussed.

Key words. Game theory, Nash equilibrium, numerical solution of polynomial equations.

AMS(MOS) subject classifications. 91A05, 91A06, 91A12, 12-04.

1. Introduction.

Game theory is a cross-disciplinary field whose aim is to model and understand how rational decision-makers make choices. Game theory focuses on situations where the outcome depends on the choices of two or more such decision-makers. Some of game theory's most significant success has been in the social sciences, most notably economics. In recognition of this, the 1994 Nobel Memorial Prize in Economics was awarded to John Nash, John Harsanyi, and Reinhard Selten, three pioneers of the field. The scope of the influence of game theory is not limited to social science; for example, MAYNARD SMITH [8] applied game theoretic concepts to the study of evolutionary biology. Computer scientists have applied game theory to inform the design of automated agents.

A central concept of game theory is the Nash equilibrium. Nash equilibrium can be thought of as a condition of mutual consistency among the beliefs and actions of a collection of agents, each of whom seeks to maximize his own objective function. (A formal definition is given in Section 2.) Nash equilibria thus have a normative attraction, in that they represent points which are stable with respect to unilateral changes in action by any one player. At a Nash equilibrium, no individual player can change his action in such a way as to improve his outcome in the game.

For games in which there are a finite number of players, each of whom can choose one action from a finite set of possible actions, the existence of an equilibrium was established by NASH [12]. Nash's existence argument allows players to choose from the set of probability distributions over their actions. For many games, this is essential; the game "rock, paper, scissors" (RPS) is a familiar example in which randomization is necessary for the

*Department of Economics, Texas A&M University, College Station, TX 77843 (turocy@econmail.tamu.edu).

existence of equilibrium. While the mutual consistency property of Nash equilibrium suggests a fixed-point formulation, the Nash equilibrium problem can be stated in several equivalent ways; see the survey of McKelvey and McLennan [9] for more details. Each formulation suggests a different computational approach for locating equilibria.

The mathematical versatility and beauty of the Nash equilibrium formulation aside, practitioners in the various fields in which the concept is applied are primarily interested in getting the set of equilibria of their games of interest, and in estimating what games are feasible to solve with a given computational budget. The software package Gambit (McKelvey, McLennan, and Turocy [11]) aims to provide such users a set of routines for computing the equilibria of an arbitrary finite game. Currently, Gambit provides a number of methods which can compute one or more equilibria in games with more than two players. However, clear guidance from experience as to which method or methods to use remains absent. This paper documents one step in the direction of "black-boxing" the process of computing equilibria in those games.

One class of approaches which has received recent attention is based on the solution to systems of polynomial equations. The calculation is organized by enumerating supports, where a support is a set of strategies which are assigned positive probability. The Nash equilibrium conditions imply a set of polynomial equations, the solutions of which contain the set of Nash equilibria on that support.

This approach is not new; in fact, this is the way most humans go about trying to manually compute equilibria. Dickhaut and Kaplan [4] first documented a program to compute equilibria using support enumeration. A method which attempts to compute all isolated Nash equilibria, using essentially the support enumeration described in this paper, has been present in Gambit since the mid-1990s.

This paper reports computational results on a new implementation, using Gambit as the frontend for defining games and generating the systems of polynomial equations to be solved. Polyhedral homotopy continuation (Huber and Sturmfels [6]) is used as the backend to generate the solutions of these systems, using PHCpack (Verschelde [16]).

The program bridges a gap between two previous papers. Herings and Peeters [5] develop essentially the same algorithm, decomposing the game by means of "admissible" supports and showing how to solve the resulting systems of polynomial equations using a homotopy method. This paper augments their work by giving and proving the correctness of an explicit algorithm for finding all admissible supports, and characterizing the scaling of the algorithm as the size of the game increases. The choice of PHCpack as the backend is inspired in part by the results of Datta [3], who gives a performance report on solving for all totally mixed Nash equilibria of a game using PHCpack. Those results show that the numerical

approach in PHCpack outperforms programs to solve polynomial equations via computer algebra using Grobner bases.

The paper proceeds as follows. Section 2 outlines the Nash equilibrium problem. It describes the support enumeration process to compute all supports which might harbor Nash equilibria, and argues the correctness of the process. Section 3 works through the computational process for a sample game. Section 4 evaluates the performance and scalability of the program. Section 5 discusses games in which the set of Nash equilibria has components with positive dimension, and the implications for implementations of this method as a "black box" solver. Finally, Section 6 concludes with a discussion of future directions and extensions, and the role of support enumeration methods within the toolkit of computation in game theory.

2. The algorithm.

2.1. Games in strategic form. A game in strategic form is played by a collection of N players, indexed $i = 1, 2, \ldots, N$. Each player i independently and simultaneously chooses from a set of strategies $S_i = \{1, 2, \ldots, J_i\}$. These choices determine an outcome $\{j_1, j_2, \ldots, j_N\}$. Players rank outcomes according to a payoff function $u_i(j_1, j_2, \ldots, j_N)$, with larger values being preferred to smaller ones.

For example, consider the game played by two drivers approaching each other on a country road. Each driver can choose to steer his vehicle to the right (strategy number 1) or to the left (strategy number 2). The drivers prefer not having a collision to having one, but do not care whether they pass each other on the right or the left. This ranking of outcomes is captured by the payoff function $u_i(1,1) = u_i(2,2) = 1$ and $u_i(1,2) = u_i(2,1) = 0$ for both drivers $i = 1, 2$.

In some types of games, players may find it in their best interest to be unpredictable. In RPS, for example, a player who chooses rock with certainty will lose a majority of games, while a player who randomizes his play equally across the three choices can guarantee he will win at least one-third of games and lose no more than one-third of games. A randomized strategy for player i will be denoted π_i, with π_{ij} being the probability that player i chooses his jth "pure" strategy. The set of all randomized strategies for player i is Σ_i. The payoff function is extended to account for mixed strategies by taking the expected value

$$u_i(\pi_1, \pi_2, \ldots, \pi_N) = \sum_{j_1=1}^{J_1} \sum_{j_2=1}^{J_2} \cdots \sum_{j_N=1}^{J_N} u_i(j_1, j_2, \ldots, j_N) \pi_{1j_1} \pi_{2j_2} \cdots \pi_{Nj_N}. \tag{2.1}$$

Thus the payoff function is polynomial, or more specifically multilinear, in the probabilities π_{ij}.

2.2. Nash equilibrium. One objective of game theory is to predict how players will make their choices. The central concept of classical game theory is the Nash equilibrium. In words, a Nash equilibrium is a situation in which each player makes a choice which maximizes his expected payoff, given the choices all other players make. Returning to the example of the two drivers, it is a Nash equilibrium for both drivers to steer to the right. If driver 1 expects driver 2 to steer to the right, then driver 1 wants also to steer to the right. The same is true for driver 2, if he expects driver 1 to steer to the right. This is not the unique Nash equilibrium, however; it is also a Nash equilibrium for both drivers to steer to the left.

Nash equilibria may also involve active randomization by one or more players. Consider RPS, and assume both players get a payoff of 1 if they win, -1 if they lose, and 0 for a tie. Suppose player 1 anticipates player 2 will choose paper; then, player 1 prefers to play scissors. But, if player 1 is going to play scissors, player 2 does not want to play paper; rather, he would want to play rock. Continuing this chain of reasoning leads to a cycle. RPS does not have any Nash equilibria in which players do not randomize. NASH [12] established that an equilibrium point in randomized strategies always exists for games with a finite number of players and strategies.

Formally, a randomized strategy profile $\pi = (\pi_i)_{i=1}^N$ is a *Nash equilibrium* if and only if it satisfies

$$u_i(\pi) \geq u_i(\pi_1, \pi_2, \ldots, \rho_i, \ldots, \pi_N)$$

for all $\rho_i \in \Sigma_i$ for all $i = 1, 2, \ldots, N$. That is to say, for each player i there is no other randomized strategy ρ_i which gives strictly higher expected value than π_i.

Because the expected payoff to player i is linear in the probabilities π_{ij} assigned to the player's own strategies, a profile can only be a Nash equilibrium if it assigns positive probability only to pure strategies in the set $\operatorname{argmax}_{j \in S_i} u_i(\pi_1, \ldots, j, \ldots, \pi_N)$. Therefore, all strategies which are played with positive probability under π_i must give the same expected payoff: $u_i(s, \pi_{-i}) = u_i(t, \pi_{-i})$ for any two strategies $s, t \in S_i$ such that $\pi_{is} > 0$ and $\pi_{it} > 0$. This means the Nash equilibrium conditions can be written[1]

$$\begin{aligned} &u_i(s, \pi_{-i}) = u_i(t, \pi_{-i}) \forall s, t \in S_i \text{ such that } \pi_{is} > 0 \text{ and } \pi_{it} > 0; \\ &u_i(s, \pi_{-i}) \geq u_i(t, \pi_{-i}) \forall s \in S_i \text{ such that } \pi_{is} > 0, \forall t \in S_i; \\ &\pi_{is} \geq 0 \forall s \in S_i; \\ &\sum_{j=1}^{J_i} \pi_{ij} = 1. \end{aligned} \tag{2.2}$$

[1]The abusive subscript notation $-i$ indicates the components of an object belonging to all players other than player i.

In RPS, it is a Nash equilibrium for each player to play each of the three choices rock, paper, and scissors with probability $\frac{1}{3}$. Each player then will win one-third of the games and lose one-third of the games, with the remaining one-third of the games resulting in draws, for an expected payoff of zero. Furthermore, conditional on playing rock, each player will win one-third, lose one-third, and draw one-third; the same is true for paper and scissors. The equality conditions in (2.2) are therefore satisfied.

It turns out that this is the unique Nash equilibrium for RPS. To verify this, it is necessary to check all possible "supports," or sets of strategies played with positive probability. For example, one would need to verify that there are no solutions satisfying (2.2) when player 1 plays rock and scissors with positive probability, but never plays paper, while player 2 plays paper and scissors with positive probability, but never plays rock.

For any given game, the number of possible supports is

$$\prod_{i=1}^{N} 2^{J_i} - 1,$$

since each strategy $s_i \in S_i$ can independently be in or out of the support (giving 2^{J_i}), except the support where all strategies are out is not valid (minus one). The combinatoric burden can be lessened somewhat by identifying supports for which it can be easily determined that solutions to (2.2) are impossible.

2.3. Examining possible equilibrium supports. A method for efficiently enumerating "admissible" supports (following the terminology of HERINGS AND PEETERS [5]) has been part of the Gambit package (MCKELVEY ET AL [11]) since the mid-1990s. This was independently described in HERINGS AND PEETERS [5]. This section presents a formal correctness argument.

In what follows, a capital letter, for example T, refers to a subset of the strategies in a game. A subscripted capital refers to the strategies in that set belonging to a player; thus, T_i denotes the strategies in T belonging to player i. Calligraphic capitals refer to collections of sets of strategies, or collections of supports. A *support* is defined a subset of the strategies in a game that satisfies the additional requirement that each player has at least one strategy in the set. That is, a support T is a subset of $S \equiv \cup_{i=1}^{N} S_i$ such that, for each player i, $T_i \subseteq S_i$ and $T_i \neq \emptyset$.

A strategy $s_i \in S_i$ *strictly dominates* another strategy $r_i \in S_i$ if $u_i(s_i, t) > u_i(r_i, t)$ for all $t \in S_{-i}$. A strictly dominated strategy cannot be played with positive probability in a Nash equilibrium, since the conditions in (2.2) cannot be satisfied. This concept can be generalized in the context of determining the set of admissible supports. A mixed strategy π_i *strictly dominates* a pure strategy s_i *against a support* T if

$$\pi_i \succ_T s_i \equiv u_i(\pi_i, t_{-i}) > u_i(s_i, t_{-i}) \forall t_{-i} \in \times_{j \neq i} T_{-i}.$$

The operator $\succ_T$ is monotonic in T in that, if $\pi_i \succ_T s_i$, then $\pi_i \succ_V s_i$ for any support $V \subset T$.

The undominated strategies s_i in a support T are then defined as

$$U(T) = \{s \in T : \nexists\, \pi_i \in \Sigma_i \text{ such that } \pi_i \succ_T s_i\}.$$

The conditions (2.2) cannot be satisfied if T contains any strategies which are dominated against T, i.e., if $U(T) \subset T$. Let $U^*(T)$ be the support resulting from iteratively applying the operator U to T. Since the game has a finite number of strategies, $U^*(T)$ can be computed in a finite number of steps.

The set of admissible supports is

$$\mathcal{N} = \{T : U(T) = T\}.$$

The task is to efficiently enumerate the set $\mathcal{N}$. For any two sets $X, Y \subseteq S$ with $X \cap Y = \emptyset$, define

$$\mathcal{P}(X,Y) = \{T \in \mathcal{N} : x \in T \text{ for all } x \in X, \text{ and } y \notin T \text{ for all } y \in Y\}. \quad (2.3)$$

In words, $\mathcal{P}(X,Y)$ is the set of admissible supports in which all the strategies in X are present, and all the strategies in Y are not present. Observe that $\mathcal{P}(\emptyset,\emptyset) = \mathcal{N}$. If $X \cup Y = S$, then

$$\mathcal{P}(X,Y) = \begin{cases} \{X\} & \text{if } U(X) = X \text{ and } X_i \neq \emptyset \quad \text{for all } i, \\ \emptyset & \text{if } U(X) \neq X \text{ or } X_i = \emptyset \quad \text{for some } i. \end{cases}$$

If $X \cup Y \subset S$, then $\mathcal{P}(X,Y)$ can be decomposed into the union of two disjoint sets

$$\mathcal{P}(X,Y) = \mathcal{P}(X \cup \{s\}, Y) \cup \mathcal{P}(X, Y \cup \{s\}) \quad (2.4)$$

for any $s \in S \backslash (X \cup Y)$.

The following recursive procedure based on (2.4) efficiently uses dominance information to compute $\mathcal{N}$. For any partition of S into X, Y, and $Z \equiv S \backslash (X \cup Y)$,

- Compute $U^*(X \cup Z)$.
  - If some player i has no strategies in $U^*(X \cup Z)$, $\mathcal{P}(X,Y) = \emptyset$.
  - If some strategy $s \in X$ but $s \notin U^*(X \cup Z)$, $\mathcal{P}(X,Y) = \emptyset$.
- Let $Y' = Y \cup \{s \in Z : s \notin U^*(X \cup Z)\}$, $Z' = Z \backslash \{s \in Z : s \notin U^*(X \cup Z)\}$.
- If $Z' = \emptyset$, then $\mathcal{P}(X,Y) = \{X\}$.
- If $Z' \neq \emptyset$, select any strategy $z \in Z'$. Compute $\mathcal{P}(X,Y)$ recursively by

$$\mathcal{P}(X,Y) = \mathcal{P}(X \cup \{z\}, Y') \cup \mathcal{P}(X, Y' \cup \{z\}).$$

Thus the following proposition holds.

PROPOSITION 2.1. *Every support containing a Nash equilibrium is in the set* $\mathcal{P}(\emptyset, \emptyset)$. *Furthermore, the support enumeration process visits each of these supports exactly one time.*

Proof. $\mathcal{P}(\emptyset, \emptyset) = \mathcal{N}$ follows from the definition in (2.3), observing that the universal quantifiers hold trivially because X and Y are empty. The process visits any support at most once because the sets in the decomposition in (2.4) are disjoint. □

3. A worked example. To illustrate the computational process, consider the three-player game in Table 1. In this game, each player has two strategies, labeled t and b for Player 1, l and r for Player 2, and u and d for Player 3. Each row gives the respective payoffs to the three players for each of the possible strategy combinations. The set of Nash equilibria of this game consists of nine isolated points, which has been shown to be the maximal number a game of this dimension can have (MCKELVEY AND MCLENNAN [10]).

TABLE 1
A three-player game with nine isolated Nash equilibria.

1	2	3	u_1, u_2, u_3
t	l	u	9, 8, 12
t	l	d	0, 0, 0
t	r	u	0, 0, 0
t	r	d	3, 4, 6
b	l	u	0, 0, 0
b	l	d	3, 4, 6
b	r	u	9, 8, 2
b	r	d	0, 0, 0

At initialization, set $X = Y = \emptyset$, so $Z = \{t, b, l, r, u, d\}$. By inspection, no strategies are dominated against $X \cup Z$. Therefore, the algorithm selects a strategy from Z, say t, and recursively computes $\mathcal{P}(\{t\}, \emptyset)$ and $\mathcal{P}(\emptyset, \{t\})$.

For $X = \{t\}$ and $Y = \emptyset$, $X \cup Z = S$; as noted before, no strategies are dominated. Therefore, select a strategy from Z, for example b, and recursively compute $\mathcal{P}(\{t, b\}, \emptyset)$ and $\mathcal{P}(\{t\}, \{b\})$.

Take the case $X = \{t\}$, $Y = \{b\}$. No strategies are dominated against $X \cup Z = \{t, l, r, u, d\}$. Select l from Z, and recursively compute $\mathcal{P}(\{t, l\}, \{b\})$ and $\mathcal{P}(\{t\}, \{b, l\})$.

Take the case $X = \{t\}$, $Y = \{b, l\}$. Here, b and l are guaranteed not to be used by players 1 and 2, respectively. Therefore, only the two rows in Table 1 that are relevant are those where player 1 chooses t and player 2 chooses r. Against the support $\{t, r, u, d\}$, d dominates u for player 3, so $U(\{t, r, u, d\}) = \{t, r, d\}$. Therefore, $X = \{t\}$, $Y' = \{b, l, u\}$,

and $Z' = \{r, d\}$. At this point, r and d must be assigned to X, since otherwise the resulting support will not be valid. Therefore, $\{t, r, d\}$ is an admissible support.

Proceeding in this way, there are 11 admissible supports, out of $3 \times 3 \times 3 = 27$ possible.

$$\begin{aligned}\mathcal{N} = \{&\{t,b,l,r,u,d\}, \{t,b,l,r,u\}, \{t,b,l,r,d\}\\ &\{t,b,l,u,d\}, \{t,b,r,u,d\}, \{t,l,r,u,d\},\\ &\{t,l,u\}, \{t,r,d\}, \{b,l,r,u,d\}, \{b,l,d\}, \{b,r,u\}\}.\end{aligned}$$

For each support, consider whether conditions (2.2) can be satisfied. An immediate consequence of admissibility is that any support in $\mathcal{N}$ with exactly one strategy for each player is automatically an equilibrium.

Now consider the full support $\{t, b, l, r, u, d\}$. Let π_t be the probability player 1 chooses t, π_l the probability player 2 chooses l, and π_u the probability player 3 chooses u. The equal-payoff conditions in (2.2) for the three players are

$$\begin{aligned}9\pi_l\pi_u + 3(1-\pi_l)(1-\pi_u) &= 3\pi_l(1-\pi_u) + 9(1-\pi_l)\pi_u\\ 8\pi_t\pi_u + 4(1-\pi_t)(1-\pi_u) &= 4\pi_t(1-\pi_u) + 8(1-\pi_t)\pi_u\\ 12\pi_t\pi_l + 2(1-\pi_t)(1-\pi_l) &= 6\pi_t(1-\pi_l) + 6(1-\pi_t)\pi_l.\end{aligned}$$

This system of equations has two solutions: $(\pi_t, \pi_l, \pi_u) = \left(\frac{2}{5}, \frac{1}{2}, \frac{1}{3}\right)$ and $(\pi_t, \pi_l, \pi_u) = \left(\frac{1}{2}, \frac{2}{5}, \frac{1}{4}\right)$. Since all strategies are used with positive probability, all the inequality conditions in (2.2) hold automatically, and so both these solutions are Nash equilibria.

Next, consider the admissible support $\{t, l, r, u, d\}$. The equal-payoff conditions for players 2 and 3 simplify to

$$\begin{aligned}4(1-\pi_u) &= 8\pi_u\\ 2(1-\pi_l) &= 6\pi_l.\end{aligned}$$

This system has solution $(\pi_l, \pi_u) = \left(\frac{1}{4}, \frac{1}{3}\right)$, making $(\pi_t, \pi_l, \pi_u) = \left(0, \frac{1}{4}, \frac{1}{3}\right)$ a candidate Nash equilibrium. To verify that this is an equilibrium, the condition $u_1(b; \pi_{-1}) \geq u_1(t; \pi_{-1})$ must hold; it does, since

$$\begin{aligned}u_1(b; \pi_{-1}) &= 3\pi_l(1-\pi_u) + 9(1-\pi_l)\pi_u = 4\frac{3}{4}\\ u_1(t; \pi_{-1}) &= 9\pi_l\pi_u + 3(1-\pi_l)(1-\pi_u) = 2\frac{1}{4}.\end{aligned}$$

Not all solutions of the polynomial equations correspond to Nash equilibria. Consider the admissible support $\{b, l, r, u, d\}$. The equal-payoff conditions are

$$\begin{aligned}8\pi_u &= 4(1-\pi_u)\\ 12\pi_l &= 6(1-\pi_l),\end{aligned}$$

which has solution $(\pi_l, \pi_u) = \left(\frac{1}{3}, \frac{1}{3}\right)$, making $(\pi_t, \pi_l, \pi_u) = \left(0, \frac{1}{3}, \frac{1}{3}\right)$ a candidate Nash equilibrium. However, the condition $u_1(t; \pi_{-1}) \geq u_1(b; \pi_{-1})$ fails to hold, since $u_1(t; \pi_{-1}) = 2\frac{1}{3}$ but $u_1(b; \pi_{-1}) = 2\frac{2}{3}$. Thus, this solution does not correspond to a Nash equilibrium.

In total, eight of the admissible supports have at least one Nash equilibrium, with the full support having two, bringing the total number of Nash equilibria to nine:

$$(\pi_t, \pi_l, \pi_u) = \{(1/2, 2/5, 1/4), (2/5, 1/2, 1/3), \\ (1/2, 1/2, 1), (1/3, 1, 1/4), \\ (1, 1, 1), (1, 0, 0), (0, 1/4, 1/3), (0, 1, 0), (0, 0, 1)\}.$$

In addition, on three supports, $\{t, b, l, r, d\}$, $\{t, b, r, u, d\}$, and $\{t, l, r, u, d\}$, there is a solution to the equal-payoff conditions that does not correspond to a Nash equilibrium.

4. Performance and scalability. The support enumeration method has been implemented in Gambit. The program uses the Gambit library for the support enumeration, and hands off the systems of equations generated to PHCpack for solution. The numerical experiments in this section characterize the scalability of the method on games of various sizes, as a rough guide to the practical limitations of the algorithm in the current implementation.

4.1. Games with randomly-drawn payoffs. The first set of results evaluates the computational time for games of a given dimension with randomly-selected payoffs. Each payoff is drawn independently from the uniform distribution on $[0, 1]$.

TABLE 2
Some summary statistics on the average performance of the program on games with randomly-drawn payoffs.

				Runtime (sec)		#Ga-
Dimension	Supports	NE	non-NE	Enum	PHCpack	mes
2^3 = 2x2x2	2.98	1.92	1.63	0.01	0.02	100
2^4	12.75	3.86	22.15	0.27	0.28	100
2^5	54.98	7.34	195.51	0.48	4.14	100
2^6	224.48	15.52	1559.54	3.19	82.48	100
2^7	866.11	34.09	12538.22	21.57	1881.51	100
2^8	3123.45	78.45	106185.27	164.70	51362.03	11
3x3x3	55.45	4.68	111.93	0.27	1.49	100
4x4x4	909.80	11.61	3469.58	6.40	108.71	100
5x5x5	12616.64	28.24	94374.07	136.99	6652.47	87
3x3x3x3	805.34	18.45	6101.96	8.90	318.53	100

Table 2 presents summary statistics on the average performance of the program. The dimension column gives the number of strategies for each

player. The second column gives the average number of admissible supports considered in the games sampled. The third and fourth column together report the average number of solutions found by PHCpack, divided into those which correspond to Nash equilibria and those which do not. The fifth and sixth columns give the average runtime, in seconds, on a Pentium IV workstation with a single 2.8GHz processor. The column "Enum" gives the time spent performing the enumeration, and "PHCpack" the time spent solving the systems of equations.

The striking feature of Table 2 is that the time per invocation of PHCpack, and the number of solutions found that do not correspond to Nash equilibria, increase very rapidly in the size of the game. The average number of supports visited ranges from roughly one-third to one-half of the maximum possible for each size game.

4.2. Games with more structure: a coordination game example. It is possible, however, that these results are unduly pessimistic. Games which are of interest to practitioners are not drawn randomly from the set of possible games; rather, they often have a particular structure. This point is made, for example, by NUDELMAN ET AL [14], whose GAMUT database of games collects examples from several such classes. For some classes of games, the support enumeration process may be able to exploit this structure by eliminating many supports from consideration. Based on the timing results in the previous section, reducing the number of polynomial systems to be considered should permit significantly larger games to be solved feasibly.

As an example, consider the family of "minimum effort" games. Each player selects a level of effort from an interval of integers $\{1, 2, \ldots, J\}$. A player selecting a level of effort e incurs a cost of effort $c(e)$, where the function $c(\cdot)$ is increasing in effort. Each player receives a benefit $b(\min(e_1, e_2, \ldots, e_N))$, which depends only on the lowest level of effort chosen. Each player's payoff is given by

$$u_i(e_1, e_2, \ldots, e_N) = b(\min(e_1, e_2, \ldots, e_N)) - c(e_i).$$

It is an equilibrium in pure strategies for all players to choose the same effort level e, for each possible effort level e. In addition, there are many equilibria in which randomization is present.

For example, consider the case of $c(e) = 2e$ and $b(e) = 3e$, with $\omega_i = 0$. For $N = 3$ and $J = 4$, there are 336 admissible supports, and total runtime is 33.3 seconds. Similarly, for $N = 3$ and $J = 5$, there are 2677 admissible supports, and the program runs in 1590 seconds, and when $N = 4$ and $J = 4$, there are 3203 admissible supports, and runtime is 9138 seconds. These compare favorably with the "average" games from Table 2. In this example, the symmetry of the game is not exploited, which would further significantly decrease the computational load.

5. Games with non-isolated equilibria. In all the examples in Section 4, the set of Nash equilibria consists of a collection of isolated points in the space of randomized strategies. This phenomenon is generically true, in that it occurs with probability one if payoff values are drawn randomly. However, games which are of interest in applications often do not fall into this category. In these games, the set of equilibria may have components with positive dimension.

NAU ET AL [13] provide an example of such a game, with payoffs as given in Table 3.

TABLE 3

A three-player game with a positive-dimension component of Nash equilibria spanning five admissible supports.

1	2	3	u_1, u_2, u_3
t	l	u	0, 0, 2
t	l	d	1, 1, 0
t	r	u	0, 3, 0
t	r	d	0, 0, 0
b	l	u	3, 0, 0
b	l	d	0, 0, 0
b	r	u	0, 0, 0
b	r	d	0, 0, 3

This game has three Nash equilibria in pure strategies: $(\pi_t, \pi_l, \pi_u) = (1,0,1), (0,1,1), (0,0,0)$. The first two are extreme points of one connected component of equilibria $U_1 \cup U_2 \cup U_3$, where

$$U_1 = \left\{ (\pi_t, \pi_l, \pi_u) = (1,0,\alpha) \text{ for } \alpha \in \left[\frac{1}{4}, 1\right] \right\}$$

$$U_2 = \left\{ (\pi_t, \pi_l, \pi_u) = (0,1,\alpha) \text{ for } \alpha \in \left[\frac{1}{4}, 1\right] \right\}$$

$$U_3 = \left\{ (\pi_t, \pi_l, \pi_u) = \left(\frac{1-\alpha}{1-(1/3)\alpha}, \alpha, \frac{1}{4}\right) \text{ for } \alpha \in [0,1] \right\}.$$

The set of Nash equilibria is shown in Figure 1. The connected component contains equilibria on a total of five admissible supports.

This example illustrates the need for a full "black-box" solver to identify supports on which a positive-dimension set of solutions exists, and to identify components which span multiple admissible supports. This initial implementation does neither, and is able to identify only the two pure-strategy endpoints of the connected component, as well as the two "corner" points.

To understand the types of degeneracy which may arise, consider the polynomial systems generated by some of the admissible supports. On the

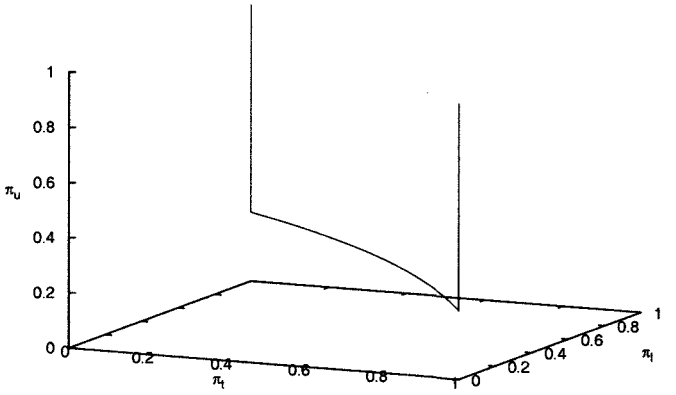


FIG. 1. *The set of Nash equilibria* (π_t, π_l, π_u) *for the game in Table 3.*

support $\{t, b, r, d\}$, the sole equal-payoff constraint reduces to $0 = 0$; this occurs on a total of six admissible supports. The implementation flags these as "singular," meaning the user needs to examine these supports manually for possible equilibria. The corner points are identified, but only as edge cases on larger supports. For example, the Nash equilibrium $\left(0, 1, \frac{1}{4}\right)$ is identified as a solution on the support $\{t, b, l, u, d\}$.

While it is fairly straightforward to flag some supports as singular, checking for possible positive-dimension solution sets remains imperfect. The positive-dimension component in this game passes through the interior of the cube in Figure 1, which corresponds to the full support $\{t, b, l, r, u, d\}$. On this support, the equal-payoff conditions are

$$\begin{aligned} \pi_l(1 - \pi_u) &= 3\pi_l\pi_u \\ \pi_t(1 - \pi_u) &= 3\pi_t\pi_u \\ 2\pi_t\pi_l &= 3(1 - \pi_t)(1 - \pi_u). \end{aligned}$$

Since $\pi_t, \pi_l \neq 0$, the first two equations are redundant, and the system has solutions along the curve U_3. The current implementation does not identify this redundancy. It does identify two solution points which are not Nash equilibria, in which π_t or π_l fall outside $[0, 1]$.

6. Discussion. Support enumeration methods will likely play a significant role in the computation of Nash equilibria in finite games. This method is the only one currently known that is guaranteed to find all isolated Nash equilibria in finite time. It will therefore be an essential tool for applications requiring identification of all equilibria. BAJARI ET AL [1] use this method in econometric estimation of games and equilibria based on empirical data.

In experimental economics, researchers design the parameters of games to be played by paid subjects in controlled laboratory settings. The properties of the set of Nash equilibria are one criterion often used in selecting those parameters. For example, CHEN ET AL [2] applied the program to a cost-sharing game to establish the nonexistence of Nash equilibria in which any player randomizes. With this knowledge, it is possible to test whether Nash equilibrium is a useful organizing principle for the observed data.

The support enumeration method can be specialized for particular tasks. For example, one might want to know whether a strategy is played with positive probability in any equilibrium. To do so, one only needs to consider the admissible supports containing that strategy. In other cases, one might be interested in finding out whether the game has a unique equilibrium. An effective method to accomplish this should minimize the time it takes to find two equilibria, if two are present. Some heuristics in this direction are investigated by PORTER ET AL [15], who find that support enumeration again stacks up well against other methods for finding a sample equilibrium.

Despair at the combinatoric complexity of the support enumeration process should also be tempered by the observation that many games are naturally represented in extensive, or game tree, form. Using the sequence form representation of KOLLER, MEGIDDO, AND VON STENGEL [7], the strategy space for each player may be represented more efficiently. Converting an extensive game to strategic form for use with the support enumeration method described here may result in a strategic form which is exponential in the size of the original tree. The sequence form representation, on the other hand, is linear in the size of the tree. A support enumeration method similar to the one described here can be used on the sequence form, and the Nash equilibrium conditions for each support result in a system of polynomial equations and inequalities, just as in the strategic form. An implementation of the support enumeration process has been available in Gambit since the mid-1990s; interfacing this with the PHCpack solver is a future objective.

A second avenue for future development is in solving the systems of equations generated by the Nash equilibrium conditions on a support. These systems have some regular structures. The equations generated by the indifference conditions for player i involve only the probabilities of strategies of players other than player i. In addition, the current implementation ignores the nonnegativity constraints on the variables in solving the systems. The equations generated by one support are similar to those generated by other supports which differ in the addition or removal of one strategy. In addition, HERINGS AND PEETERS [5] list several other possible optimizations for special-case supports. None of these possible optimizations are currently exploited by this implementation.

7. Acknowledgements. This paper is based on a talk given at the Institute for Mathematics and its Applications at the University of Minnesota, during the Software for Algebraic Geometry workshop in October 2006. Comments and suggestions from the participants there were greatly appreciated. Suggestions from three anonymous referees did much to sharpen the exposition.

The programs described here, including the scripts used to build the randomly-drawn and minimum-effort games, are all available under the GNU General Public License from the Gambit website at http://gambit.sourceforge.net.

REFERENCES

[1] PATRICK BAJARI, HAN HONG, AND STEPHEN RYAN. *Identification and estimation of discrete games of complete information.* Working paper, University of Minnesota, 2007.

[2] YAN CHEN, LAURA RAZZOLINI, AND THEODORE L. TUROCY. *Congestion Allocation for Distributed Networks: An Experimental Study.* Economic Theory, **33**:121–143, 2007.

[3] R.S. DATTA. *Using Computer Algebra To Compute Nash Equilibria.* In Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation (ISSAC 2003), pages 74–79, 2003.

[4] JOHN DICKHAUT AND TODD KAPLAN. *A Program for Finding Nash Equilibria.* The Mathematica Journal, **1**(4):87–93, 1992.

[5] P.J.J. HERINGS AND R.J.A.P. PEETERS. *A globally convergent algorithm to compute all Nash equilibria for n-Person games.* Annals of Operations Research, **137**:349–368, 2005.

[6] BIRK HUBER AND BERND STURMFELS. *A polyhedral method for solving sparse polynomial systems.* Mathematics of Computation, **64**:1541–1555, 1995.

[7] DAPHNE KOLLER, NIMROD MEGIDDO, AND BERNHARD VON STENGEL. *Efficient computation of equilibria for extensive two-person games.* Games and Economic Behavior, **14**:247–259, 1996.

[8] JOHN MAYNARD SMITH. *Evolution and the Theory of Games.* Cambridge UP, Cambridge, 1982.

[9] RICHARD D. MCKELVEY AND ANDREW M. MCLENNAN. *Computation of Equilibria in Finite Games.* In Hans Amman, David A. Kendrick, and John Rust, editors, The Handbook of Computational Economics, Volume I, pages 87–142. Elsevier, 1996.

[10] RICHARD D. MCKELVEY AND ANDREW M. MCLENNAN. *The maximal number of regular totally mixed Nash equilibria.* Journal of Economic Theory, **72**:411–425, 1997.

[11] RICHARD D. MCKELVEY, ANDREW M. MCLENNAN, AND THEODORE L. TUROCY. Gambit: Software Tools for Game Theory. http://gambit.sourceforge.net.

[12] JOHN F. NASH. *Equilibrium points in n-person games.* Proceedings of the National Academy of Sciences, **36**:48–49, 1950.

[13] ROBERT NAU, SABRINA GOMEZ CANOVAS, AND PIERRE HANSEN. *On the Geometry of Nash Equilibria and Correlated Equilibria.* International Journal of Game Theory, **32**:443–453, 2004.

[14] E. NUDELMAN, J. WORTMAN, Y. SHOHAM, AND K. LEYTON-BROWN. *Run the GAMUT: A comprehensive approach to evaluating game-theoretic algorithms.* In Proceedings of the Third International Joint Conference on Autonomous Agents & Multi Agent Systems, 2004.

[15] RYAN W. PORTER, EUGENE NUDELMAN, AND YOAV SHOHAM. *Simple search methods for finding a Nash equilibrium.* Games and Economic Behavior, forthcoming.

[16] JAN VERSCHELDE. *Algorithm 795: PHCpack: a general-purpose solver for polynomial systems by homotopy continuation.* ACM Transactions on Mathematical Software, **25**(2), 1999.

APATOOLS: A SOFTWARE TOOLBOX FOR APPROXIMATE POLYNOMIAL ALGEBRA

ZHONGGANG ZENG*

Abstract. Approximate polynomial algebra becomes an emerging area of study in recent years with a broad spectrum of applications. In this paper, we present a software toolbox `ApaTools` for approximate polynomial algebra. This package includes Maple and Matlab functions implementing advanced numerical algorithms for practical applications, as well as basic utility routines that can be used as building blocks for developing other numerical and symbolic methods in computational algebra.

Key words. software, polynomial, GCD, factorization, elimination, Jordan Canonical Form, multiplicity, rank.

AMS(MOS) subject classifications. 11R09,12D05,65F15,65F20,65F22,65H10.

1. Introduction. Approximate polynomial algebra emerges as a growing area of study in recent years. With rich theories and abundant symbolic algorithms already developed in commutative algebra and algebraic geometry, a solid foundation is in place for building numerical and hybrid computing methods. In fact, many robust numerical and numeric-symbolic algorithms have been developed for solving polynomial systems [1, 11, 17, 25, 28], univariate factorization [32, 33], univariate GCD [5, 9, 13, 15, 23, 30], multivariate GCD [10, 34, 35], computing the dual bases and multiplicity structures of polynomial ideals [2, 7, 20], multivariate factorization [4, 10, 34], and polynomial elimination [3, 8, 31], etc. Those algorithms have a broad spectrum of applications in scientific computing such as robotics [19, 24, 25, 26], control [6, 14], image processing [21, 22], computational biology and chemistry [8], and so on.

In this paper, we present a software toolbox `ApaTools` for approximate polynomial algebra. This toolbox includes Matlab and Maple implementations of basic algorithms, utility procedures and test suites. The Matlab version of `ApaTools` is also named `Apalab`. Those functions can be used either directly in applications or as building blocks for more advanced computing methods.

One of the main difficulties for numerical computation in polynomial algebra is the ill-posedness that frequently occurs when a problem has a discontinuous solution with respect to data. Those ill-posed problems are not directly suitable for floating point arithmetic since the solutions are infinitely sensitive to rounding errors. It has been shown in recent development that such difficulty can be overcome by seeking an *approximate solution* as formulated based on the three principles of *backward nearness*,

*Department of Mathematics, Northeastern Illinois University, Chicago, IL 60625, USA. Research supported in part by NSF under Grant DMS-0412003 and DMS-0715127 (zzeng@neiu.edu).

maximum co-dimension and *minimal distance* [30, 33, 36] which we shall elaborate in §2. With such formulations, algorithms can be constructed using a two-staged strategy: identifying the solution structure first followed by solving a least squares problem for an approximate solution. `ApaTools` includes generic routines for matrix building and Gauss-Newton iteration that are the main engines for both stages of computation.

Algorithms implemented in `ApaTools` are designed to regularize the problem for removing ill-posedness rather than extending machine precision for accurate computation. The Maple version of `ApaTools` can nonetheless take advantage of variable machine precision via setting the worksheet "Digits" conveniently. The Matlab version (`Apalab`) uses the hardware precision throughout, and is generally much faster than the Maple counterparts.

`ApaTools` is at its start as an on-going project. At this stage, the main objective is to provide researchers in polynomial algebra with generic and versatile tools that simplify and accelerate algorithm development, experimentation, and implementation. Particularly on the Maple platform, enabling convenient and faster coding take higher priority over fast execution. Moreover, we emphasize on achieving the highest possible accuracy and robustness in algorithm design and implementation. Nevertheless, the functions in `ApaTools` such as approximate GCD appear to be "the most efficient and reliable" [27, page 223].

`ApaTools` is maintained at the author's website and freely accessible to academic researchers and educators for the foreseeable future.

2. Exact and approximate polynomial algebra. Conventional symbolic computation assumes both data and arithmetic to be *exact*. In practical applications, however, problem data are likely to be approximate. As a result, exact solutions of those inexact problems may not serve the practical purposes. Alternatively, an *approximate solution* is sought to approximate the underlying exact solution. The comparison can be seen in following examples.

EXAMPLE 1. **Exact and approximate GCD.** The following polynomial pair has an exact GCD in $gcd(f,g) = x^2 - 2xy + 3z$:

$$\begin{aligned} f(x,y,z) &= \tfrac{11}{7}x^3 - \tfrac{23}{17}x^2z - \tfrac{22}{7}x^2y + \tfrac{46}{17}xyz + \tfrac{33}{7}zx - \tfrac{69}{17}z^2, \\ g(x,y,z) &= \tfrac{31}{29}x^4 + \tfrac{29}{23}x^2y - \tfrac{62}{29}x^3y - \tfrac{58}{23}xy^2 + \tfrac{93}{29}x^2z + \tfrac{87}{23}yz. \end{aligned} \tag{2.1}$$

When polynomials are represented with finite precision floating point numbers in coefficients

$$\begin{aligned} \tilde{f}(x,y,z) = {} & 1.57142857x^3 - 1.35294118x^2z - 3.14285714x^2y + 2.70588235xyz \\ & +4.71428571zx - 4.05882353z^2, \\ \tilde{g}(x,y,z) = {} & 1.06896552x^4 + 1.26086957x^2y - 2.13793103x^3y - 2.52173913xy^2 \\ & +3.20689655x^2z + 3.78260870yz, \end{aligned} \tag{2.2}$$

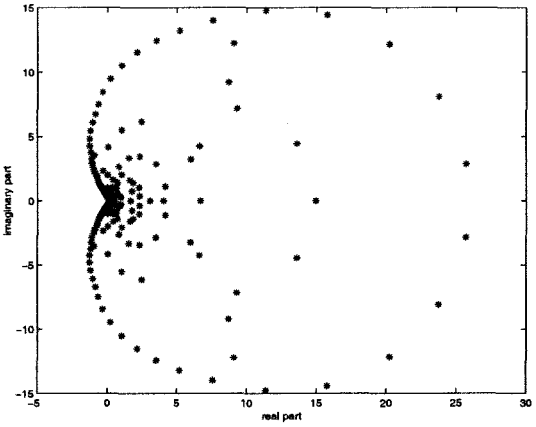


FIG. 1. *Matlab results for the polynomial. (2.3).*

the exact GCD degrades to $gcd(\tilde{f}, \tilde{g}) = 1$. The *approximate GCD within* ε, as calculated by our `ApaTools` function `mvGCD` for $2 \times 10^{-8} < \varepsilon < 10^{-1}$,

$$gcd_{\varepsilon}(\tilde{f}, \tilde{g}) \quad = \quad .9999999999x^2 - 1.999999994xy + 2.999999994z$$

is an accurate approximation to the underlying exact GCD. □

EXAMPLE 2. **Exact and approximate univariate factorization.** Univariate factorization in complex field $\mathbb{C}$ is equivalent to root-finding. Consider univariate polynomial

$$\begin{aligned} p(x) = {} & x^{200} - 400\,x^{199} + 79500\,x^{198} + \ldots + 2.04126914035338 \cdot 10^{86}\,x^{100} \\ & -3.55467815448396 \cdot 10^{86}\,x^{99} + \ldots + 1.261349023419937 \cdot 10^{53}\,x^2 \qquad (2.3) \\ & -1.977831229290266 \cdot 10^{51}\,x + 1.541167191654753 \cdot 10^{49} \\ \approx {} & (x-1)^{80}(x-2)^{60}(x-3)^{40}(x-4)^{20} \qquad (2.4) \end{aligned}$$

with coefficients in hardware precision (16-digits). Using the coefficient vector of p as input, the standard Matlab root-finder outputs a cluster of 200 roots as shown in Figure 1.

In contrast, our Matlab function `uvFactor` in `Apalab` calculates an *approximate factorization*:

```
>> [F,res,cond] = uvFactor(p);

THE CONDITION NUMBER:                    2.57705
THE BACKWARD ERROR:                   7.62e-016
THE ESTIMATED FORWARD ROOT ERROR:     3.93e-015

 FACTORS

  ( x -   4.000000000000008 )^20
  ( x -   2.999999999999994 )^40
  ( x -   2.000000000000002 )^60
  ( x -   1.000000000000000 )^80
```

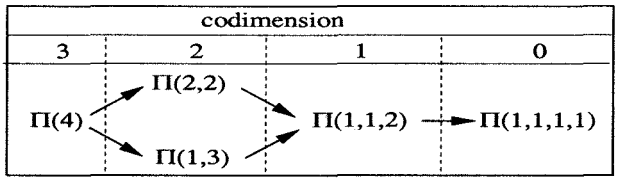


FIG. 2. *Stratification of manifolds of degree 4 polynomials, with "⟶" denoting "in the closure of".*

The *approximate factors* calculated by `uvFactor` approximate the underlying factors in (2.4) with 15 correct digits. In exact sense, however, the perturbed polynomial p in (2.3) has only linear factors corresponding to a cluster of simple roots. Figure 1 shows the root cluster computed by Matlab function ROOTS, which is backward accurate in the sense that the result consists of exact roots of a nearby polynomial. □

Both examples show the effect of ill-posedness on inexact problems. Since the solution structure can be completely altered by an infinitesimal error in data, it may not be meaningful to compute the exact solution of a perturbed ill-posed problem. Therefore, we need to formulate an *approximate solution* for the given problem in order to remove the ill-posedness. Using polynomial factorization as an example, we elaborate the formulation process below.

Associating a general monic polynomial of degree 4 with a vector

$$p(x) = x^4 + p_1x^3 + p_2x^2 + p_3x + p_4 \quad \sim \quad [p_1, p_2, p_3, p_4] \in \mathbb{C}^4,$$

the set of all such polynomials possessing the factorization of multiplicities 1 and 3

$$\begin{aligned}\{x^4 - (\alpha + 3\beta)x^3 + 3(\beta^2 + \alpha\beta)x^2 - (3\alpha\beta^2 + \beta^3)x + \alpha\beta^3 \\ = (x-\alpha)^1(x-\beta)^3 \mid \alpha, \beta \in \mathbb{C}, \alpha \neq \beta\}\end{aligned}$$

denoted by $\Pi(1,3)$ forms a manifold of codimension 2 with respect to the metric topology induced by the norm $\|p\| = \sqrt{|p_1|^2 + \cdots + |p_4|^2}$. On the other hand, manifold $\Pi(1,3)$ is in the closure $\overline{\Pi(1,1,2)}$ of manifold $\Pi(1,1,2)$ of codimension 1 since

$$\lim_{\varepsilon,\varepsilon' \to 0} (x-\alpha)^1(x-\beta+\varepsilon)^1(x-\beta+\varepsilon')^2 \quad = \quad (x-\alpha)^1(x-\beta)^3.$$

Likewise $\Pi(1,1,2) \subset \overline{\Pi(1,1,1,1)} \equiv \mathbb{C}^4$, and the five manifolds form a *stratification* as shown in Figure 2.

EXAMPLE 3. When polynomial $p \in \Pi(1,3)$, say

$$p \;=\; x^4 + 4x^3 - 16x - 16 \;=\; (x-2)^1(x+2)^3$$

is perturbed as $\tilde{p}$, say

$$\tilde{p} \;=\; x^4 + 3.9999x^3 - 16x - 16,$$

the original factoring structure $(1,3)$ is lost since $\tilde{p} \in \mathbb{C}^4 = \overline{\Pi(1,1,1,1)}$. However, structure $(1,3)$ can still be recovered from $\tilde{p}$ since $\Pi(1,3)$ is the manifold of *highest codimension* passing through the ε-neighborhood of $\tilde{p}$ for any ε satisfying

$$0.0001 = \|p - \tilde{p}\| \;<\; \varepsilon \;<\; \inf\left\{\|\tilde{p} - q\| \;\middle|\; q \in \Pi(2,2) \cup \Pi(4)\right\},$$

say $\varepsilon = 0.001$. After identifying $\Pi(1,3)$ this way, we define the *approximate factorization* of $\tilde{p}$ within ε as the exact factorization of $\hat{p} = (x - \hat{\alpha})^1(x - \hat{\beta})^3$ that is the nearest polynomial to $\tilde{p}$ in $\Pi(1,3)$. With this formulation, computing the approximate factorization of $\tilde{p}$ is a well-posed problem, and the roots $\hat{\alpha}$ and $\hat{\beta}$ of $\hat{p}$ are approximation to the intended roots α and β of p with error bounded by a structure preserving condition number [33]. We can verify this well-posedness using `ApaTools` function `uvFactor`:

```
> F, res := uvFactor(x^4+3.9999*x^3-16*x-16,x,0.001):
> evalf(F);
```

$$.9999887784\,(-2.000005124 + .9999999996\,x)\,(2.000004920 + 1.000000000\,x)^3$$

with backward error $\|\tilde{p} - \hat{p}\| \approx 0.0000907$ and the forward error 0.0000071 on the roots. The accuracy should be considered remarkable given the data error of 0.0001. □

Generally, algebraic problems are often ill-posed because the set of problems whose solutions possessing a distinct structure form a manifold of positive codimension, and perturbations generically pushes a given problem away from the manifold. Our strategy starts with formulating the *approximate solution* of an ill-posed algebraic problem following a "three strikes" principle for removing the discontinuity:

Backward nearness: The approximate solution to the given (inexact) problem $\tilde{P}$ is the exact solution of a nearby problem $\hat{P}$ within ε.

Maximum codimension: The nearby problem $\hat{P}$ is in the manifold Π possessing the highest codimension among all manifolds passing through the ε-neighborhood of $\tilde{P}$.

Minimum distance: Problem $\hat{P}$ is the nearest point in manifold Π to the given problem $\tilde{P}$

Here "the nearest point" may have a question mark in its uniqueness at least theoretically. However, multiple occurrences of the identical minimum distance appear to be non-generic and we have not yet encountered such cases in practical computation. At least, the nearest point is unique when the backward nearness is sufficiently small [30].

Following these principles, we can formulate the approximate polynomial GCD [30], the approximate matrix Jordan Canonical Form [36], the approximate factorization of polynomials [33], etc., as well-posed and often well-conditioned problems, so that it becomes feasible to calculate such approximate solutions using floating point arithmetic. Based on those formulation principles, computing the approximate solution can be carried out in two optimization processes: maximizing the codimension of manifolds followed by minimizing the distance to the manifold, leading to a two-staged strategy for designing robust algorithms:

Stage I: Calculating the solution structure by finding the nearby manifold Π of highest codimension.

Stage II: Solving for the approximate solution by minimizing the distance from given problem to manifold Π.

The main mechanism at Stage I is matrix rank-revealing, while Stage II relies on solving nonlinear least squares problems.

We illustrate the two stage-approach using the univariate GCD computation as an example. For given polynomials f and g of degrees $m \geq n > 0$ respectively, the k-th degree GCD manifold

$$\mathscr{P}_k^{m,n} = \{(p,q) \mid \deg(p) = m, \deg(q) = n, \deg(gcd(p,q)) = k\}$$

is a manifold of codimension k with respect to metric topology. Whenever such a manifold passes through an ε-neighborhood of (f,g), the k-th Sylvester matrix $S_k(f,g) = [C_{m-k}(f), C_{n-k}(g)]$ is rank-deficient by one in approximate sense (c.f. convolution matrix in the next section). Consequently, Stage I for finding the highest codimension GCD manifold nearby is a successive rank-revealing process on $S_j(f,g)$ for $j = n, n-1, \cdots$ until the degree k is identified.

Once the GCD degree k is identified, Stage II becomes solving an overdetermined quadratic system

$$\begin{cases} \phi(u) &= 1 \\ u \cdot v &= f \\ u \cdot w &= g \end{cases} \quad \text{with} \quad \begin{array}{l} \deg(u) = k, \quad \deg(v) = m\text{-}k, \quad \deg(w) = n\text{-}k, \\ \text{and a proper linear functional } \phi \end{array}$$

for its least squares solution. The Gauss-Newton iteration (see §4) is proven effective for this purpose.

The backward nearness threshold ε to be provided by users generally depends on the nature of the application. The rule of thump is to set ε slightly larger than the magnitude of data error. More specifically, if one knows the given (inexact) problem $\tilde{P}$ is a small perturbation from the underlying (exact) problem P that belongs to a manifold Π. Let $\delta > 0$ be the minimum distance from $\tilde{P}$ to manifold $\Pi' \neq \Pi$ of higher or equal codimension as Π. Then for $\|P - \tilde{P}\| < \varepsilon < \delta$, the three-strikes principle ensures the correct manifold Π to be identified, and the computed solution of problem $\tilde{P}$ will approximate the underlying solution of problem P.

Using the polynomial pair $(\tilde{f}, \tilde{g})$ in (2.2) as an example, its coefficients are 9-digits approximation to (f, g) in (2.1) with perturbation magnitude 1.1×10^{-8}. Any other GCD manifold of equal or higher codimension is at least 0.1 away from (f, g). Thus any threshold ε between 1.1×10^{-8} and 10^{-1} will ensure recovering the approximate GCD. In practice, ε should be set near the low end, say 10^{-7}.

3. Matrix computation and matrix building tools. Finding the structure of an approximate solution at Stage I almost always involve matrix rank-revealing, whereas the minimization at Stage II can be accomplished using the Gauss-Newton iteration. For those considerations, the base level tools in `ApaTools` include matrix builders, rank-revealing tools, and the Gauss-Newton iteration routines.

Polynomials with a certain degree bound form a vector space $\mathcal{P}$ with a monomial basis $\{p_1, \cdots, p_n\}$ arranged by a term order. The designated term order can be considered a linear mapping

$$\Psi \ : \ p_k = x_1^{j_1} x_2^{j_2} \cdots x_\ell^{j_\ell} \quad \longrightarrow \quad \mathbf{e}_k \ \in \ \mathbb{C}^n$$

that constitutes an isomorphism between $\mathcal{P}$ and $\mathbb{C}^n$, where $\mathbf{e}_k$ denotes the k-th canonical vector of $\mathbb{C}^n$ (i.e. the k-th column of the $n \times n$ identity matrix). Let $\Psi_1 : \mathcal{P}_1 \longrightarrow \mathbb{C}^m$ and $\Psi_2 : \mathcal{P}_2 \longrightarrow \mathbb{C}^n$ be isomorphisms and $\mathcal{L} : \mathcal{P}_1 \longrightarrow \mathcal{P}_2$ be a linear transformation. Then there is a matrix $A \in \mathbb{C}^{n \times m}$ that makes the following diagram commute:

$$\begin{array}{ccc} \mathcal{P}_1 & \xrightarrow{\ \mathcal{L}\ } & \mathcal{P}_2 \\ {\scriptstyle \Psi_1}\downarrow & & \uparrow{\scriptstyle \Psi_2^{-1}} \\ \mathbb{C}^m & \xrightarrow{\ A\ } & \mathbb{C}^n \end{array}$$

Matrix $A = [\mathbf{a}_1, \cdots, \mathbf{a}_m]$ can then be generated column-by-column in the loop as follows:

for $j = 1, 2, \cdots, m$ do
- $p = \Psi_1^{-1}(\mathbf{e}_j)$
- $q = \mathcal{L}(p)$
- $\mathbf{a}_j = \Psi_2(q)$

end do

Here $\mathbf{e}_j$ is the j-th canonical vector in $\mathbb{C}^m$, namely the j-th column of the $m \times m$ identity matrix. Implementation of software tools for isomorphisms between polynomial vector spaces and $\mathbb{C}^n$ constitutes *matrix building tools* in `ApaTools`.

EXAMPLE 4. For instance, let $\mathcal{P}^k$ denote the vector space of polynomials with total degree less than or equal to k. For a fixed polynomial f, polynomial multiplication with f is a linear transformation $\mathcal{L}_f \ : \ \mathcal{P}^n \longrightarrow \mathcal{P}^m$

$$\mathcal{L}_f(g) \ = \ f \cdot g \ \in \mathcal{P}^m \quad \text{for all} \ g \in \mathcal{P}^n \tag{3.1}$$

with $m = \deg(f)+n$ that corresponds to a convolution matrix $C_n(f)$. In `ApaTools`, we include a convenient function `LinearTransformMatrix` as a generic matrix builder for any linear transformation between vector spaces of polynomials. The user needs to provide a subroutine for the linear transformation as input for `LinearTransformMatrix`. Using the linear transformation in (3.1) as an example, we write a simple Maple procedure for $\mathcal{L}_f$ with f being an input parameter:

```
> PolynMultiply := proc( g::polynom, x::{name,list}, f::polynom);
      return expand(f*g)
  end proc:
```

The convolution matrix $C_2(f)$ for $f(x,y) = x+2y+3$ can then be generated by a simple call:

```
> T := LinearTransformMatrix(PolynMultiply,[x+2*y+3],[x,y],2,3);
```

$$T := \begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 0 \\ 1 & 3 & 0 & 0 & 0 & 0 \\ 2 & 0 & 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 3 & 0 & 0 \\ 0 & 2 & 1 & 0 & 3 & 0 \\ 0 & 0 & 2 & 0 & 0 & 3 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

□

In this example, the input `PolynMultiply` is the procedure name for $\mathcal{L}_f$, item `[x+2*y+3]` is the list of parameters for `PolynMultiply` besides `[x,y]` that stands for the list of indeterminates, and `2, 3` are the degree bounds for the vector spaces $\mathcal{P}^m$ and $\mathcal{P}^n$ respectively. The graded lexicographical monomial order is used here as the default monomial ordering.

With matrices being constructed, rank-revealing is frequently applied in approximate polynomial algebra. As an example, a polynomial pair $(f,g) \in \mathcal{P}^m \times \mathcal{P}^n$ having a nontrivial GCD of degree k can be written as $f = uv$ and $g = uw$ where $u = gcd(f,g)$ with cofactors v and w. Consequently, polynomials v and w satisfy $fw - gv = 0$, or equivalently a homogeneous system of linear equations

$$\left[C_{n-k}(f),\ C_{m-k}(g)\right] \begin{bmatrix} \mathbf{w} \\ -\mathbf{v} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}$$

where $\mathbf{v}$ and $\mathbf{w}$ are coefficient vectors of v and w respectively. As a result, finding the GCD structure is equivalent to computing the rank of the Sylvester subresultant matrix $\left[C_{n-k}(f), C_{m-k}(g)\right]$ for k decreasing from $\min\{m,n\}$ until rank-deficiency occurs.

Another example is numerical elimination. For given polynomials f and g in variables x and y, if there are polynomials p and q such that variable x is eliminated from polynomial $h = p \cdot f + q \cdot g$, then

$$\partial_x(pf + qg) \;=\; \big[(\partial_x f) + f \cdot \partial_x\big]p + \big[(\partial_x g) + g \cdot \partial_x\big]q \;=\; 0. \qquad (3.2)$$

Since mapping $p \longrightarrow \big[(\partial_x f) + f \cdot \partial_x\big]p$ is a linear transformation, equation (3.2) can be converted to a homogeneous system of linear equations in matrix-vector form

$$M \begin{bmatrix} \mathbf{p} \\ \mathbf{q} \end{bmatrix} \;=\; \mathbf{0}$$

that becomes a rank-revealing and kernel-finding problem for the elimination matrix M.

In approximate polynomial algebra, we seek the *approximate rank* and the *approximate kernel* of a matrix in contrast to seeking exact rank and kernel in exact polynomial algebra. Following the same "three-strikes" principle, the approximate rank and kernel of a matrix $A \in \mathbb{C}^{m \times n}$ are the exact rank and kernel of a nearby matrix $B \in \mathbb{C}^{m \times n}$. This matrix B is the nearest matrix to A on the manifold Π_k that possesses the highest codimension among manifolds $\Pi_1, \Pi_2, \cdots$ passing through an ε-neighborhood of A, where Π_j is the set of all $m \times n$ matrices with rank j. Approximate rank/kernel can be efficiently computed using numerical rank-revealing algorithms [16, 18] that are implemented as components of `ApaTools`.

4. Nonlinear least squares and the Gauss-Newton iteration. The minimization at Stage II is a nonlinear least squares problem that can be solved using the Gauss-Newton iteration on an overdetermined system of equations in the form of

$$\mathbf{f}(\mathbf{z}) \;=\; \mathbf{0} \quad \text{for } \mathbf{f} : \mathbb{C}^n \longrightarrow \mathbb{C}^m, \quad \mathbf{z} \in \mathbb{C}^n, \quad m \geq n. \qquad (4.1)$$

We seek the least squares solution $\hat{\mathbf{z}}$ satisfying

$$\big\|\mathbf{f}(\hat{\mathbf{z}})\big\|^2 \;=\; \min_{\mathbf{z} \in \mathbb{C}^n} \big\{\|\mathbf{f}(\mathbf{z})\|^2\big\} \qquad (4.2)$$

with a choice of $\|\cdot\|$ to be either weighted or straightforward Euclidean norm. Let $J(\mathbf{z})$ be the Jacobian of $\mathbf{f}(\mathbf{z})$. Then the minimum occurs at $\hat{\mathbf{z}}$ such that [33]

$$J(\hat{\mathbf{z}})^* \mathbf{f}(\hat{\mathbf{z}}) \;=\; \mathbf{0} \qquad (4.3)$$

where $(\cdot)^*$ denotes the Hermitian transpose of a matrix or vector $(\cdot)$.

If $\mathbf{f}(\mathbf{z})$ is analytic, the Jacobian $J(\hat{\mathbf{z}})$ is injective so that the Moore-Penrose inverse $J(\hat{\mathbf{z}})^+ = \big(J(\hat{\mathbf{z}})^* J(\hat{\mathbf{z}})\big)^{-1} J(\hat{\mathbf{z}})^*$, the minimum $\|\mathbf{f}(\hat{\mathbf{z}})\|$ is

small, and the initial iterate $\mathbf{z}_0$ is close to $\hat{\mathbf{z}}$, then the Gauss-Newton iteration

$$\mathbf{z}_{k+1} = \mathbf{z}_k - J(\mathbf{z}_k)^+ \mathbf{f}(\mathbf{z}_k), \quad k = 0, 1, \cdots \tag{4.4}$$

converges to $\hat{\mathbf{z}}$ [33].

The Gauss-Newton iteration is extensively used in `ApaTools`. A typical case is computing the approximate GCD: For given polynomial pair p and q with degrees m and n respectively, we seek a polynomial triplet $(\hat{u}, \hat{v}, \hat{w})$ such that $\deg(\hat{u}) = k$, $\deg(\hat{u}\hat{v}) = m$, $\deg(\hat{u}\hat{w}) = n$, and

$$\|(p,q) - (\hat{u}\hat{v}, \hat{u}\hat{w})\|^2 = \min_{\substack{\deg(u)=k \\ \deg(uv)=m \\ \deg(uw)=n}} \{\|(p,q) - (uv, uw)\|^2\}.$$

Let $\mathbf{p}$, $\mathbf{q}$, $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$ be the coefficient vectors of p, q, u, v, and w respectively, then the overdetermined system is constructed to have the least squares solution to $uv - p = 0$ and $uv - q = 0$ along with a proper scaling equation on u. This system can be written in matrix-vector form as

$$\mathbf{f}(\mathbf{u},\mathbf{v},\mathbf{w}) = \mathbf{0} \quad \text{for } \mathbf{f}(\mathbf{u},\mathbf{v},\mathbf{w}) = \begin{bmatrix} \mathbf{r}^*\mathbf{u} - 1 \\ C_k(v)\,\mathbf{u} - \mathbf{p} \\ C_k(w)\,\mathbf{u} - \mathbf{q} \end{bmatrix} \tag{4.5}$$

where $C_j(h)$ denotes the convolution matrix corresponding to the linear transformation $\mathcal{L}_h : g \longrightarrow h \cdot g \in \mathcal{P}^l$ on the vector space $\mathcal{P}^j$ (see §3) and $\mathbf{r}$ is called a scaling vector. The choices of $\mathbf{r}$ is random or parallel to the initial estimate of $\mathbf{u}$, which makes no noticeable difference in practical computation either way. The Jacobian can be easily constructed as

$$J(\mathbf{u},\mathbf{v},\mathbf{w}) = \begin{bmatrix} \mathbf{r}^* & & \\ C_k(v) & C_{m-k}(u) & \\ C_k(w) & & C_{n-k}(u) \end{bmatrix}.$$

The Gauss-Newton iteration (4.4) can thus be applied accordingly. The auxilliary equation $\mathbf{r}^*\mathbf{u} - 1 = 0$ is crucial in making $J(\mathbf{u},\mathbf{v},\mathbf{w})$ injective.

It is essential to formulate the overdetermined system $\mathbf{f}(\mathbf{z}) = \mathbf{0}$ with enough equations such that the Jacobian $J(\hat{\mathbf{z}})$ is injective at the least squares solution $\hat{\mathbf{z}}$ and, as a result, its Moore-Penrose inverse $J(\hat{\mathbf{z}})^+ = \left(J(\hat{\mathbf{z}})^* J(\hat{\mathbf{z}})\right)^{-1} J(\hat{\mathbf{z}})^*$, ensuring the Gauss-Newton iteration to be well defined in a neighborhood of $\hat{\mathbf{z}}$. Moreover, the norm $\left\|J(\hat{\mathbf{z}})^+\right\|_2$ serves as a condition number of the approximate solution [30, 33, 36].

We provide a generic blackbox function `GaussNewton` for the general purpose Gauss-Newton iteration. This function requires the user to provide a routine for computing function $\mathbf{f}(\mathbf{z})$ in (4.1), an optional routine for computing the Jacobian $J(\mathbf{z})$ and an initial iterate $\mathbf{z}_0$ as its input.

Then `GaussNewton` carries out the Gauss-Newton iteration and outputs the least squares solution $\hat{\mathbf{z}}$ along with the residual $\|\mathbf{f}(\hat{\mathbf{z}})\|_2$. As a convenient option, the Maple version of `GaussNewton` can forgo the requirement of providing Jacobian routine by computing $J(\mathbf{z})$ with Maple symbolic manipulation.

EXAMPLE 5. Using the above example of GCD computation, the generic Gauss-Newton function needs only a user-defined subroutine for computing $\mathbf{f}(\mathbf{z})$ that can be as simple as

```
> GcdFunc := proc( z, m, n, p, q, r)
    local F, u, v, w;
    u, v, w := z[1..m], z[m+1..m+n], z[m+n+1..-1];
    F := <LinearAlgebra[DotProduct](r,u)-1, Convolution(u,v)-p,
            Convolution(u,w)-q>;
    return F[1..-1,1];
  end proc:
```

Here, the Maple routine `GcdFunc` returns the vector value of $\mathbf{f}(\mathbf{z})$ from input vector $\mathbf{z}$ consists of coefficients of u, v and w, along with parameters `m, n, p, q, r` representing the length of $\mathbf{u}$, the length of $\mathbf{v}$, coefficient vectors $\mathbf{p}$, $\mathbf{q}$ and $\mathbf{r}$, respectively. The command `Convolution(u,v)` and `Convolution(u,w)` produces coefficient vectors of polynomial products $u \cdot v$ and $u \cdot v$ respectively by `ApaTools` function `Convolution`. Then the Gauss-Newton iteration is carried out by a simple call of `GaussNewton`:

```
> # get coef. vectors p, q
> p, q := PolynomialTools[CoefficientVector](x^5-x^3+5*x^2-6*x+10,x),
          PolynomialTools[CoefficientVector](2*x^4+x^2-6,x);
> r := Vector([.4,0,.2]):                          # define scaling vector r
> z0 := Vector([1.99,0.01,1.01,                    # initial guess for u, v, w
                4.99,-3.01,0,.99,
                -3.01,0,1.99]):
> # Gauss-Newton iteration
> z,res := GaussNewton(GcdFunc,z0,[3,4,p,q,r],[1e-9,9,true]):

        Gauss-Newton step  0,  residual =  8.00e-02
        Gauss-Newton step  1,  residual =  2.01e-04
        Gauss-Newton step  2,  residual =  3.16e-09
        Gauss-Newton step  3,  residual =  4.44e-16
```

The approximate GCD and cofactors can then be retrieved from the result of the iteration:

```
> CoefficientVector2UnivariatePolynomial(z[1..3],x);
  CoefficientVector2UnivariatePolynomial(z[4..7],x);
  CoefficientVector2UnivariatePolynomial(z[8..-1],x);
```

$$2.000000000000000 + 3.1779793 \times 10^{-17}\, x + 1.000000000000000\, x^2$$

$$4.999999999999999 - 3.000000000000000\, x - 1.530197036 \times 10^{-16}\, x^2 + 0.9999999999999998\, x^3$$

$$-2.999999999999999 + 2.54808329 \times 10^{-17}\, x + 2.000000000000000\, x^2$$

that are accurate approximation to the exact GCD $u = x^2 + 2$, cofactors $v = x^3 - 3x + 5$ and $w = 2x^2 - 3$. Notice that the Jacobian routine is not necessarily an input item for `GaussNewton`. This type of generic routines enables fast implementation of experimental algorithms. □

Example 5 illustrates the most basic application of `GaussNewton` for computing the least squares zero to a vector-valued function $f(\mathbf{z})$. In approximate polynomial algebra, we often encounter a task of finding the least squares zero of a function whose domain and range are lists of polynomials. The function `GaussNewton` provides another convenient feature of direct computation of the least squares zero in terms of a list of polynomials as shown below.

EXAMPLE 6. Considering the same GCD problem and using the same polynomials as in Example 5, computing $gcd(f, g)$ is equivalent to finding the least squares zero of the function

$$F(p_1, p_2, p_3) \;=\; [\mathbf{r}^*\mathbf{p}_1 - 1,\;\; p_1 \cdot p_2 - f,\;\; p_1 \cdot p_3 - g]. \qquad (4.6)$$

whose domain and range are both lists of polynomials. To prepare for applying `GaussNewton`, we write a simple Maple procedure for the function F in (4.6) in a straightforward translation:

```
> GcdPolynFunc := proc( p, x, f, g, r )      # procedure for the gcd function
    return [ PolynomialDotProduct(r,p[1],x) - 1,
             expand(p[1]*p[2])-f, expand(p[1]*p[3])-g ]
  end proc:
```

We then set the input items and call `GaussNewton` in an intuitive manner:

```
> f, g := x^5-x^3+5*x^2-6*x+10, 2*x^4 + x^2 - 6:  # define input polynomial pair
> r := .4 + .2*x^2:                               # scaling polynomial
> u, v, w := 1.99 + 1.01*x^2,                     # set initial guesses
             4.99 - 3.01*x +0.99*x^3,
            -3.01 + 1.99*x^2:
> s,res := GaussNewton(GcdPolynFunc,[u,v,w],[[x],f,g,r],[1e-12,9,true])

          Gauss-Newton step   0,  residual =   1.34e-01
          Gauss-Newton step   1,  residual =   2.44e-04
          Gauss-Newton step   2,  residual =   2.66e-09
          Gauss-Newton step   3,  residual =   3.75e-15

> s[1]/lcoeff(s[1]); s[2]/lcoeff(s[2]); 2*s[3]/lcoeff(s[3]);  # show result

                  2.000000000000000 + 1.000000000000000 x^2

      4.999999999999999 - 2.999999999999999 x + 0.9999999999999997 x^3

                 -3.000000000000000 + 2.000000000000000 x^2
```

This feature of `GaussNewton` enables direct manipulation of polynomials and avoids the tedious task of translation between polynomials and vectors. □

Remark. Gauss-Newton iteration locally converges to a point satisfying (4.3) which is the *necessary* condition for a *local* minimum in (4.2). A *global* minimum is not guaranteed in general. As a matter of fact, a global minimum is difficult to verify, let alone to compute with any certainty. Even though a proposed method in [15] is claimed to have the capability of finding the global minimum in polynomial time, that claim is neither proved rigorously nor backed up with an implementation. On the other hand, the backward accuracy of the computing result can be verified

by the actual backward nearness whose measurement is a by-product of the Gauss-Newton iteration (i.e. the output `res` of `GaussNewton`). Consequently, the lack of guarantee in finding global minimum does not appear to be a serious concern in practical computation, or should not be until such an application arises. Furthermore, it can be proved that the minimum is indeed global if the given (inexact) problem is sufficiently close to the underlying exact problem [30]. □

5. `ApaTools` overview. We give an itemized overview of our package `ApaTools` on its functionality currently available.

1. **Base level tools**
   - `LinearTransformMatrix`: A generic function for constructing the matrix associated with a given linear transformation between vector spaces of polynomials, as described in §3.
   - `GaussNewton`: A generic function for carrying out the Gauss-Newton iteration for a given overdetermined system of nonlinear equations as described in §4.
2. **Matrix computation tools**
   - `NulVector`: The function for computing the smallest singular value and the corresponding singular vectors. In many cases, it is desirable to determine if a matrix is approximately rank-deficient without computing a full-blown singular value decomposition. Function `NulVector` is designed for this purpose. It also serves as a subroutine for `ApproxiRank` that computes the approximate rank and approximate kernel.
   - `ApproxiRank`: The function for computing the approximate rank $\mathrm{rank}_\theta(A)$ and approximate kernel $\mathcal{K}_\theta(A)$ of a matrix A within a given threshold θ. Here the approximate rank is defined as

$$\mathrm{rank}_\theta(A) = \min_{\|B-A\|_2<\theta} \mathrm{rank}(B)$$

     and the approximate kernel $\mathcal{K}_\theta(A) = \mathcal{K}(B)$ for

$$\|B-A\|_2 = \min_{\mathrm{rank}(C)=\mathrm{rank}_\theta(A)} \|C-A\|_2.$$

     Function `ApproxiRank` is efficient when the nullity of A is low. The standard singular value decomposition (SVD) may be more suitable if the approximate rank of A is around one half of the column dimension.
   - `ApproxiJCF`: The function for computing the approximate Jordan Canonical Form (JCF) of a given matrix A within a threshold ε. Computing the exact Jordan Canonical Form is

an ill-posed problem that requires exact data with symbolic computation. The approximate JCF is defined according to the same "three strikes" principle and computed with a two-staged algorithm elaborated in §2. As a result, `ApproxiJCF` is capable of computing the Jordan canonical decomposition accurately even if the matrix is perturbed [36].

3. **Univariate polynomial tools**:

   - `uvGCD`: The function for computing the approximate greatest common divisor, denoted by $gcd_\theta(f,g)$, of univariate polynomial pair (f,g) within a given threshold θ with definition as follows [30].

     Let $\mathcal{P}$ be the vector space of polynomial pairs (p,q) satisfying $\deg(p) \le \deg(f)$ and $\deg(q) \le \deg(g)$. Then $\Pi_j = \{(p,q) \in \mathcal{P} \mid \deg(gcd(p,q)) = j\}$ is a manifold of codimension j in $\mathcal{P}$ associated with the metric topology induced by the vector 2-norm for $j = 0, 1, \cdots$. Let Π_k be the highest codimension manifold among $\Pi_0, \Pi_1, \cdots$ containing a polynomial pair within distance θ of (f,g). Then the approximate GCD $gcd_\theta(f,g)$ is the exact GCD of pair (p,q), where (p,q) is the nearest polynomial pair on the manifold Π_k to the given pair (f,g).

     Function `uvGCD` accepts input polynomial pair (f,g) and threshold θ. The user can choose to omit providing θ value in Matlab, or set it to be zero in Maple, to invoke the default θ that is cube root of the square of the machine epsilon. The output consists of $u = gcd_\theta(f,g)$, cofactors v and w along with the residual $\|(f,g) - (uv, uw)\|$, where the norm $\|\cdot\|$ is either the vector 2-norm chosen by users or the default weighted 2-norm. The input polynomials are not required to be monic and will be taken "as is" without scaling. The output GCD and cofactors are generally not monic for avoiding unbalanced scaling to the computing result.

     There are several proposed algorithms in the literature for computing the univariate approximate GCD, such as [5, 9, 13, 15, 23]. Our `uvGCD` is significantly more accurate and robust than existing implementation due to the our employment of the iterative refinement at Stage II.

   - `ExtendGCD`: The function for computing a polynomial pair (p,q) such that $pf + qg = gcd_\theta(f,g)$ for given polynomial f and g using the output of `uvGCD`. The extended GCD can be applied to transforming matrices of polynomial entries, and in particular, to computing the Smith Normal Form [29, 35].

- `uvFactor`: The function for computing an approximate factorization

$$p_0(x-z_1)^{m_1}(x-z_2)^{m_2}\cdots(x-z_k)^{m_k}$$

for a given polynomial $p(x)$, as shown in Example 2. Excluding certain pathologically ill-conditioned cases, the factors of polynomials with nontrivial multiplicities $m_j > 1$ can be calculated accurately by `uvFactor` without extending the machine precision even if the coefficients of $p(x)$ are perturbed to certain extent [33].

4. **Multivariate polynomial tools**

- `mvGCD`: The function for computing the approximate GCD of a given pair (f,g) of multivariate polynomials. The formulation of the multivariate approximate GCD is similar to the univariate case. The computation requires applying `uvGCD` repeatedly on the univariate polynomial pairs projected from (f,g) in determining the GCD structure before calling `GaussNewton` for solving a least squares problem that minimizes the distance from the given polynomial pair to a GCD manifold. For details, see [34].

- `SquarefreeFactor`: The function for computing an approximate squarefree factorization of a given multivariate polynomial p. Here, again, the notion of the *approximate squarefree factorization* is formulated using the "three-strikes" principle. Function `SquarefreeFactor` produces two types of squarefree factorizations: a staircase squarefree factorization

$$p = (p_1)^1(p_2)^2\cdots(p_k)^k$$

where $p_1,\cdots,p_k$ are coprime, or a flat-type squarefree factorization

$$p = f_1\cdot f_2\cdots f_k$$

where $f_j = p_j p_{j+1}\cdots p_k$ for $j=1,\cdots,k$, and f_i divides f_j for all $i>j$. Each p_i or f_j is "squarefree", namely it has no repeated nontrivial factors of its own.

- `MultiplicityStructure`: The function for computing the multiplicity structure of a given polynomial system

$$\begin{cases} f_1(x_1,\cdots,x_s)=0 \\ \quad\vdots \\ f_t(x_1,\cdots,x_s)=0 \end{cases} \tag{5.1}$$

at a given zero $\mathbf{x}^* = (x_1^*, \cdots, x_s^*)$.

Let $\partial_{\mathbf{j}}$ denote a differential monomial

$$\partial_{\mathbf{j}} \equiv \frac{1}{j_1! \cdots j_s!} \frac{\partial^{j_1+\cdots+j_s}}{\partial x_1^{j_1} \cdots \partial x_s^{j_s}} \quad \text{for} \quad \mathbf{j} = [j_1, \cdots, j_s] \in \mathbb{N}^s.$$

A vector of complex numbers $\mathbf{a} = [\alpha_{\mathbf{j}} \mid \mathbf{j} \in \mathbb{N}^s]$ corresponds to a differential functional $a[\mathbf{x}^*]$ defined as

$$a[\mathbf{x}^*](f) = \sum_{\mathbf{j} \in \mathbb{N}^s} \alpha_{\mathbf{j}} \partial_{\mathbf{j}} f(\mathbf{x}^*)$$

for any polynomial f in the ideal $\mathcal{I} = \langle f_1, \cdots, f_t \rangle$. The vector space

$$\mathcal{D}_{\mathbf{x}^*}(\mathcal{I}) \equiv \{a[\mathbf{x}^*] \mid a[\mathbf{x}^*](f) = 0 \quad \text{for all} \quad f \in \langle f_1, \cdots, f_t \rangle\}$$

is called the dual space of the ideal $\mathcal{I}$ at $\mathbf{x}^*$. The dimension of the dual space $\mathcal{D}_{\mathbf{x}^*}(\mathcal{I})$ is the multiplicity of $\mathbf{x}^*$ as a zero to the system (5.1). The dual space itself constitutes the multiplicity structure of the system (5.1) at zero $\mathbf{x}^*$ [27].

The function `MultiplicityStructure` calculates the multiplicity, a basis for the dual space along with other invariants such as breadth, depth and Hilbert function [7] even if the system and zero are inexact.

For closely related algorithms, see [2, 20].

- `PolynomialEliminate`: The function for computing polynomials p, q and h such that $h = pf + qg$ belongs to a specified elimination ideal generated by polynomials f and g. In other words, `PolynomialEliminate` eliminates a specified variable x_j in $h = pf + qg$ for given f and g in variables $x_1, \cdots, x_s$, by solving the differential equation

$$\frac{\partial}{\partial x_j}(pf + qg) = 0$$

for p and q via rank-revealing tool `ApproxiKernel` on a sequence of matrices generated by the matrix building tool `LinearTransformMatrix`. Combined with `mvGCD`, this elimination tool is particularly useful in solving polynomial systems whose solutions contain nonzero dimensional components [31].

6. A brief comparison to existing software packages. For a similar purpose, a software package `SNAP` (Symbolic-Numeric Algorithms for Polynomials) [12] is included in recent releases of Maple. The scope

of ApaTools is much broader than current SNAP in functionality. In short, SNAP consists of only ten functions with three univariate GCD procedures (EpsilonGCD, QuasiGCD and QRGCD) with nearly identical objectives, along with seven utility routines. In comparison, ApaTools' matrix rank-revealing, univariate factorization, multivariate GCD, squarefree factorization, multiplicity structure, polynomial elimination, etc. are original developments. For the univariate GCD function that is included in both ApaTools and SNAP, our uvGCD is more advanced in robustness and accuracy due to the iterative refinement at Stage II. A detailed comparison on univariate GCD computation can be found in [30]. Our approach of building on matrix building and least squares solving is also novel in software design.

On Matlab platform, there appear to be no other packages that are comparable to Apalab in functionality and comprehensiveness.

7. Future development. ApaTools is an on-going project. While continuing to refine the existing functions, we plan to expand the package by developing more algorithms and their implementations for various problems in approximate polynomial algebra. For instance, a project is underway for developing a numerical algorithm for computing the approximate irreducible factorization of multivariate polynomials. The "three-strikes" principle is again applied to formulate the notion of the approximate irreducible factorization, and the algorithm follows the two-staged strategy. Moreover, the algorithm and its implementation will be built on top of the existing uvGCD, mvGCD, SquarefreeFactor, and ApproxiJCF.

Acknowledgements. This work is originated and partially completed during the author's six week participation in the Thematic Year on Application of Algebraic Geometry at the Institute for Mathematics and Its Applications (IMA) in 2006. The author is grateful to IMA for its refreshing research environment.

REFERENCES

[1] D. Bates, J.D. Hauenstern, A.J. Sommese, and C.W. Wampler, *Bertini: Software for Nmerical Algebraic Geometry.* http://www.nd.edu/~sommese/bertini, 2006.

[2] D.J. Bates, C. Peterson, and A.J. Sommese, *A numerical-symbolic algorithm for computing the multiplicity of a component of an algebraic set*, J. of Complexity, **22** (2006), pp. 475–489.

[3] L. Busé, I. Emiris, and B. Mourrain, Multires. http://www-sop.inria.fr/galaad/mourrain/multires.html.

[4] G. Chèze and A. Galligo, *Four lessons on polynomial absolute factorization*, Solving Polynomial Equations: Foundations, Algorithms, and Applications, A. Dickenstein and I.Z. Emiris, eds, Algorithms and Computation in Mathematics 14, Springer-Verlag (2005), pp. 339–392.

[5] R.M. Corless, S.M. Watt, and L. Zhi, *QR factoring to compute the GCD of univariate approximate polynomials*, IEEE Trans. Signal Processing, **52** (2003), pp. 3394–3402.

[6] B. DATTA AND K. DATTA, *Toeplitz algorithms for controllability, GCD and Cauchy index.* Proceedings of the American Control Conference.

[7] B. DAYTON AND Z. ZENG, *Computing the multiplicity structure in solving polynomial systems.* Proceedings of ISSAC '05, ACM Press, pp. 116–123, 2005.

[8] I.Z. EMIRIS, E.D. FRITZILAS, AND D. MANOCHA, *Algebraic algorithms for determining structure in biological chemistry*, Internatinal J. Quantum Chemistry, **106** (2006), pp. 190–210.

[9] I.Z. EMIRIS, A. GALLIGO, AND H. LOMBARDI, *Certified approximate univariate GCDs*, J. Pure Appl. Algebra, **117/118** (1997), pp. 229–251.

[10] S. GAO, E. KALTOFEN, J. MAY, Z. YANG, AND L. ZHI, *Approximate factorization of multivariate polynomials via differential equations.* Proc. ISSAC '04, ACM Press, pp. 167–174, 2004.

[11] T. GAO AND T.-Y. LI, *MixedVol: A software package for mixed volume computation*, ACM Trans. Math. Software, **31** (2005), pp. 555–560.

[12] C.-P. JEANNEROD AND G. LABAHN, *The SNAP package for arithemetic with numeric polynomials.* In International Congress of Mathematical Software, World Scientific, pp. 61–71, 2002.

[13] E. KALTOFEN, J. MAY, Z. YANG, AND L. ZHI, *Structured low rank approximation of Sylvester matrix*, in Symbolic-Numeric Computation, Trends in Mathematics, D. Wang and L. Zhi, editors, Birkhäuser Verlag, Basel, Switzerland (2007), pp. 69–83.

[14] N. KARCANIAS AND M. MITROULI, *A matrix pencil based numerical method for the computation of the GCD of polynomials*, IEEE Trans. on Automatic Control, **39** (1994), pp. 977–981.

[15] N.K. KARMARKAR AND Y.N. LAKSHMAN, *On approximate polynomial greatest common divisors*, J. Symb. Comput., **26** (1998), pp. 653–666.

[16] T.-L. LEE, T.Y. LI, AND Z. ZENG, *A rank-revealing method with updating, downdating and applications, Part II.* submitted, 2006.

[17] T.-Y. LI, *Solving polynomial systems by the homotopy continuation method*, Handbook of Numerical Analysis, XI, edited by P.G. Ciarlet, North-Holand, Amsterdam (2003), pp. 209–304.

[18] T.Y. LI AND Z. ZENG, *A rank-revealing method with updating, downdating and applications*, SIAM J. Matrix Anal. Appl., **26** (2005), pp. 918–946.

[19] J.P. MERLET, *Singular configuration of parallel manipulators and Grassman geometry*, Int. J. Robitics Research, **8** (1989), pp. 45–59.

[20] B. MOURRAIN, *Isolated points, duality and residues*, J. of Pure and Applied Algebra, **117** & **118** (1996), pp. 469–493. Special issue for the Proc. of the 4th Int. Symp. on Effective Methods in Algebraic Geometry (MEGA).

[21] H.-T. PAI, A. BOVIK, AND B. EVANS, *Multi-channel blind image restoration*, TUBITAK Elektrik J. of Electrical Eng. and Comput. Sci., **5** (1997), pp. 79–97.

[22] S. PILLAI AND B. LIANG, *Blind image deconvolution using GCD approach*, IEEE Trans. Image Processing, **8** (1999), pp. 202–219.

[23] D. RUPPRECHT, *An algorithm for computing certified approximate GCD of n univariate polynomials*, J. Pure and Appl. Alg., **139** (1999), pp. 255–284.

[24] A.J. SOMMESE, J. VERSCHELDE, AND C.W. WAMPLER, *Numerical factorization of multivariate complex polynomials*, Theoretical Computer Science, **315** (2003), pp. 651–669.

[25] A.J. SOMMESE AND C.W. WAMPLER, *The Numerical Solution of Systems of Polynomials*, World Scientific Pub., Hackensack, NJ, 2005.

[26] B.M. ST-ONGE AND C.M. GOSSELIN, *Singularity analysis and representation of general Gough-Stewart platform*, Int. J. Robitics Research, **19** (2000), pp. 271–288.

[27] H.J. STETTER, *Numerical Polynomial Algebra*, SIAM, 2004.

[28] J. VERSCHELDE, *Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation*, ACM Trans. Math. Software (1999), pp. 251–276.
[29] J. VERSCHELDE AND Y. WANG, *Computing dynamic output feedback laws*, IEEE Trans. Automatic Control (2004), pp. 1552–1571.
[30] Z. ZENG, *The approximate GCD of inexact polynomials. Part I.* Preprint, 2007.
[31] Z. ZENG, *A polynomial elimination method for symbolic and numerical computation.* Preprint, 2007.
[32] Z. ZENG, *Algorithm 835: Multroot – A Matlab package for computing polynomial roots and multiplicities*, ACM Trans. Math. Software, **30** (2004), pp. 218–235.
[33] ———, *Computing multiple roots of inexact polynomials*, Math. Comp., **74** (2005), pp. 869–903.
[34] Z. ZENG AND B. DAYTON, *The approximate GCD of inexact polynomials. II: A multivariate algorithm.* Proceedings of ISSAC'04, ACM Press, pp. 320–327 (2006).
[35] ———, *The approximate GCD of inexact polynomials. Part II.* Preprint, 2007.
[36] Z. ZENG AND T.Y. LI, *A numerical method for computing the Jordan Canonical Form.* Preprint, 2007.

LIST OF WORKSHOP PARTICIPANTS

- Douglas N. Arnold, Institute for Mathematics and its Applications, University of Minnesota Twin Cities
- Donald G. Aronson, Institute for Mathematics and its Applications, University of Minnesota Twin Cities
- Daniel J. Bates, Institute for Mathematics and its Applications, University of Minnesota Twin Cities
- Gian Mario Besana, Department of Computer Science-Telecommunications, DePaul University
- Anna M. Bigatti, Dipartimento di Matematica, Università di Genova
- Rachelle Bouchat, Department of Mathematics, University of Kentucky
- Massimo Caboara, Dipartimento di Matematica, Università di Genova
- Ionut Ciocan-Fontanine, Institute for Mathematics and its Applications, University of Minnesota Twin Cities
- Barry H. Dayton, Department of Mathematics, Northeastern Illinois University
- Wolfram Decker, Fachrichtung Mathematik, Universität des Saarlandes
- Alicia Dickenstein, Departamento de Matemática, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires
- Kenneth R. Driessel, Department of Mathematics, Iowa State University
- Makan Fardad, Department of Electrical and Computer Engineering, University of Minnesota Twin Cities
- Xuhong Gao, Department of Mathematical Sciences, Clemson University
- Jürgen Gerhard, Department of Research and Development, Maplesoft
- Jason E. Gower, Institute for Mathematics and its Applications, University of Minnesota Twin Cities,
- Daniel R. Grayson, Department of Mathematics, University of Illinois at Urbana-Champaign
- Genhua Guan, Department of Mathematics, Clemson University
- Yun Guan, Department of Mathematics, Statistics and Computer Science, University of Illinois at Chicago
- Tatsuyoshi Hamada, Department of Applied Mathematics, Fukuoka University
- Marshall Hampton, Department of Mathematics and Statistics, University of Minnesota

- Gloria Haro Ortega, Institute for Mathematics and its Applications, University of Minnesota Twin Cities
- Jonathan D. Hauenstein, Department of Mathematics, University of Notre Dame
- Raymond Hemmecke, Otto-von-Guericke-Universität Magdeburg
- Milena Hering, Institute for Mathematics and its Applications, University of Minnesota Twin Cities
- Benjamin J. Howard, Institute for Mathematics and its Applications, University of Minnesota Twin Cities
- Evelyne Hubert, Project CAFE, Institut National de Recherche en Informatique Automatique (INRIA)
- Farhad Jafari, Department of Mathematics, University of Wyoming
- Abdul Salam Jarrah, Virginia Bioinformatics Institute, Virginia Polytechnic Institute and State University
- Anders Nedergaard Jensen, Institut für Mathematik, Technische Universität Berlin
- Gabriela Jeronimo, Departamento de Matematica - FCEyN, University of Buenos Aires
- Masakazu Kojima, Department of Mathematical and Computing Sciences, Tokyo Institute of Technology
- Song-Hwa Kwon, Institute for Mathematics and its Applications, University of Minnesota Twin Cities
- Niels Lauritzen, Institut for Matematiske Fag, Aarhus University
- Grégoire Lecerf, Laboratoire de Mathematiques, Universit Versailles/Saint Quentin-en-Yvelines
- Anton Leykin, Institute for Mathematics and its Applications, University of Minnesota Twin Cities
- Hstau Y. Liao, Institute for Mathematics and its Applications, University of Minnesota Twin Cities
- Gennady Lyubeznik, School of Mathematics, University of Minnesota Twin Cities
- Diane Maclagan, Department of Mathematics, Rutgers University
- Peter Nicholas Malkin, CORE, Centre for Operations Research and Econometrics, Universit Catholique de Louvain
- Hannah Markwig, Institute for Mathematics and its Applications, University of Minnesota Twin Cities
- Thomas Markwig, Department of Mathematics, Universität Kaiserslautern
- Andrew A. McLennan, Department of Economics, University of Minnesota Twin Cities
- Richard B. Moeckel, School of Mathematics, University of Minnesota Twin Cities
- Bernard Mourrain, Project GALAAD, Institut National de Recherche en Informatique Automatique (INRIA)

- Uwe Nagel, Department of Mathematics, University of Kentucky
- Jiawang Nie, Institute of Mathematics and its Applications, University of Minnesota Twin Cities
- Michael E. O'Sullivan, Department of Mathematics and Statistics, San Diego State University
- Rohit Pandita, Department of Aerospace Engineering and Mechanics, University of Minnesota Twin Cities
- Antonis Papachristodoulou, Engineering Science University of Oxford
- Pablo A. Parrilo, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology
- Chris Peterson, Department of Mathematics, Colorado State University
- Sonja Petrovic, Department of Mathematics, University of Kentucky
- Kathy Wei Piret, Department of Mathematics, Statistics and Computer Science, University of Illinois
- Sorin Popescu, Department of Mathematics, SUNY
- Jacob Quant, University of Minnesota Twin Cities
- Gregory J. Reid, Department of Applied Mathematics, University of Western Ontario
- Victor Reiner, School of Mathematics, University of Minnesota Twin Cities
- Joel Roberts, School of Mathematics, University of Minnesota Twin Cities
- Fabrice Rouillier, Projet SALSA, Institut National de Recherche en Informatique Automatique (INRIA)
- Bjarke Hammersholt Roune, Department of Mathematics, Aarhus University
- David Rusin, Department of Mathematical Sciences, Northern Illinois University
- Arnd Scheel, Institute for Mathematics and its Applications, University of Minnesota Twin Cities
- Hans Schöenemann, Department of Mathematics Universität Kaiserslautern Eric Schost, LIX, École Polytechnique
- Mathias Schulze, Department of Mathematics, Oklahoma State University
- Robin Scott, Department of Applied Mathematics, University of Western Ontario
- Chehrzad Shakiban, Institute of Mathematics and its Applications, University of Minnesota Twin Cities
- Donald H. Singley, 3M
- Andrew J. Sommese, Department of Mathematics, University of Notre Dame

- Steven Sperber, School of Mathematics, University of Minnesota Twin Cities
- Dumitru Stamate, School of Mathematics, University of Minnesota Twin Cities
- William Stein, Department of Mathematics, University of Washington
- Brandilyn Stigler, Mathematical Biosciences Institute
- Michael E. Stillman, Department of Mathematics, Cornell University
- Erik Stokes, University of Kentucky
- Nobuki Takayama, Department of Mathematics, Kobe University
- Akiko Takeda, Department of Mathematical and Computing Sciences, Tokyo Institute of Technology
- Amelia Taylor, Colorado College Enrique Augusto Tobis, Computer Science and Mathematics, University of Buenos Aires
- Carl Toews, Institute for Mathematics and its Applications, University of Minnesota Twin Cities
- Elias P. Tsigaridas, project GALAAD, INRIA Sophia Antipolis
- Theodore L. Turocy, Department of Economics, Texas A&M University
- Balint Vanek, Department of Aerospace Engineering and Mechanics, University of Minnesota Twin Cities
- Jan Verschelde, Department of Mathematics, Statistics and Computer Science, University of Illinois at Chicago
- John Voight, Institute for Mathematics and its Applications, University of Minnesota Twin Cities
- Charles W. Wampler II, General Motors Research and Development
- Stephen M. Watt, Department of Computer Science, University of Western Ontario
- Thomas Wolf, Department of Mathematics, Brock University
- Wenyuan Wu, Department of Applied Mathematics, University of Western Ontario
- Zhonggang Zeng, Department of Mathematics, Northeastern Illinois University
- Ailing Zhao, Department of Mathematics, Statistics and Computer Science, University of Illinois
- Ke Zhou, Electrical and Computer Engineering Department, University of Minnesota Twin Cities
- Yan Zhuang, Department of Mathematics, Statistics and Computer Science, University of Illinois

1997–1998 Emerging Applications of Dynamical Systems
1998–1999 Mathematics in Biology
1999–2000 Reactive Flows and Transport Phenomena
2000–2001 Mathematics in Multimedia
2001–2002 Mathematics in the Geosciences
2002–2003 Optimization
2003–2004 Probability and Statistics in Complex Systems: Genomics, Networks, and Financial Engineering
2004–2005 Mathematics of Materials and Macromolecules: Multiple Scales, Disorder, and Singularities
2005-2006 Imaging
2006-2007 Applications of Algebraic Geometry
2007-2008 Mathematics of Molecular and Cellular Biology
2008-2009 Mathematics and Chemistry
2009-2010 Complex Fluids and Complex Flows

IMA SUMMER PROGRAMS

1987 Robotics
1988 Signal Processing
1989 Robust Statistics and Diagnostics
1990 Radar and Sonar (June 18–29)
New Directions in Time Series Analysis (July 2–27)
1991 Semiconductors
1992 Environmental Studies: Mathematical, Computational, and Statistical Analysis
1993 Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations
1994 Molecular Biology
1995 Large Scale Optimizations with Applications to Inverse Problems, Optimal Control and Design, and Molecular and Structural Optimization
1996 Emerging Applications of Number Theory (July 15–26)
Theory of Random Sets (August 22–24)
1997 Statistics in the Health Sciences
1998 Coding and Cryptography (July 6–18)
Mathematical Modeling in Industry (July 22–31)
1999 Codes, Systems, and Graphical Models (August 2–13, 1999)
2000 Mathematical Modeling in Industry: A Workshop for Graduate Students (July 19–28)
2001 Geometric Methods in Inverse Problems and PDE Control (July 16–27)
2002 Special Functions in the Digital Age (July 22–August 2)

2003 Probability and Partial Differential Equations in Modern Applied Mathematics (July 21–August 1)
2004 n-Categories: Foundations and Applications (June 7–18)
2005 Wireless Communications (June 22–July 1)
2006 Symmetries and Overdetermined Systems of Partial Differential Equations (July 17–August 4)
2007 Classical and Quantum Approaches in Molecular Modeling (July 23-August 3)
2008 Geometrical Singularities and Singular Geometries (July 14-25)

IMA "HOT TOPICS" WORKSHOPS

- Challenges and Opportunities in Genomics: Production, Storage, Mining and Use, April 24–27, 1999
- Decision Making Under Uncertainty: Energy and Environmental Models, July 20–24, 1999
- Analysis and Modeling of Optical Devices, September 9–10, 1999
- Decision Making under Uncertainty: Assessment of the Reliability of Mathematical Models, September 16–17, 1999
- Scaling Phenomena in Communication Networks, October 22–24, 1999
- Text Mining, April 17–18, 2000
- Mathematical Challenges in Global Positioning Systems (GPS), August 16–18, 2000
- Modeling and Analysis of Noise in Integrated Circuits and Systems, August 29–30, 2000
- Mathematics of the Internet: E-Auction and Markets, December 3–5, 2000
- Analysis and Modeling of Industrial Jetting Processes, January 10–13, 2001
- Special Workshop: Mathematical Opportunities in Large-Scale Network Dynamics, August 6–7, 2001
- Wireless Networks, August 8–10 2001
- Numerical Relativity, June 24–29, 2002
- Operational Modeling and Biodefense: Problems, Techniques, and Opportunities, September 28, 2002
- Data-driven Control and Optimization, December 4–6, 2002
- Agent Based Modeling and Simulation, November 3–6, 2003
- Enhancing the Search of Mathematics, April 26-27, 2004
- Compatible Spatial Discretizations for Partial Differential Equations, May 11-15, 2004
- Adaptive Sensing and Multimode Data Inversion, June 27–30, 2004
- Mixed Integer Programming, July 25–29, 2005
- New Directions in Probability Theory, August 5–6, 2005
- Negative Index Materials, October 2-4, 2006

- The Evolution of Mathematical Communication in the Age of Digital Libraries, December 8-9, 2006
- Math is Cool! and Who Wants to Be a Mathematician?, November 3, 2006
- Special Workshop: Blackwell-Tapia Conference, November 3-4, 2006
- Stochastic Models for Intracellular Reaction Networks, May 11-13, 2008

SPRINGER LECTURE NOTES FROM THE IMA:

The Mathematics and Physics of Disordered Media
Editors: Barry Hughes and Barry Ninham
(Lecture Notes in Math., Volume 1035, 1983)

Orienting Polymers
Editor: J.L. Ericksen
(Lecture Notes in Math., Volume 1063, 1984)

New Perspectives in Thermodynamics
Editor: James Serrin
(Springer-Verlag, 1986)

Models of Economic Dynamics
Editor: Hugo Sonnenschein
(Lecture Notes in Econ., Volume 264, 1986)

The IMA Volumes in Mathematics and its Applications

1 **Volume 142: Compatible Spatial Discretizations**
Editors: Douglas N. Arnold, Pavel B. Bochev, Richard B. Lehoucq, Roy A. Nicolaides, and Mikhail Shashkov

2 **Volume 143: Wireless Communications**
Editors: Prathima Agrawal, Daniel Matthew Andrews, Philip J. Fleming, George Yin, and Lisa Zhang

3 **Volume 144: Symmetries and Overdetermined Systems of Partial Differential Equations**
Editors: Michael Eastwood and Willard Miller, Jr.

4 **Volume 145: Topics in Stochastic Analysis and Nonparametric Estimation**
Editors: Pao-Liu Chow, Boris Mordukhovich, and George Yin

5 **Volume 146: Algorithms in Algebraic Geometry**
Editors: Alicia Dickenstein, Frank-Olaf Schreyer, and Andrew Sommese

6 **Volume 147: Symmetric Functionals on Random Matrices and Random Matchings Problems**
by Grzegorz A. Rempała and Jacek Wesołowski

7 **Volume 148: Software for Algebraic Geometry**
Editors: Michael E. Stillman, Nobuki Takayama, and Jan Verschelde

The full list of IMA books can be found at the Web site of Institute for Mathematics and its Applications:
http://www.ima.umn.edu/springer/volumes.html