SNC'11

Proceedings of the 2011 International Workshop on Symbolic-Numeric Computation

June 7-9, 2011, San Jose , California, USA

Association for Computing Machinery

Advancing Computing as a Science & Profession

San Jose, California, USA
June 7-9, 2011

SNC'11

Proceedings of the 2011 Internation Workshop on

Symbolic-Numeric Computation

Supported by:

Numerical Algorithms Group, Maplesoft, The University of Western Ontario, Wilfrid Laurier University

Sponsored by:

SIGSAM

Edited by:

Marc Moreno Maza

Proceedings of SNC 2011: Symbolic-Numeric Computation 2011
Editor: Marc Moreno Maza
©SNC 2011
ISBN 978-1-4503-0515-0

Foreword to the SNC 2011 Conference

Algorithms that combine techniques from symbolic and numeric computation have been of increasing importance and interest over the past decade. The necessity to work reliably with imprecise and noisy data, and for speed and accuracy within algebraic and hybrid-numerical problems, has encouraged a new synergy between the numerical and symbolic computing fields. Novel and exciting problems from industrial, mathematical and computational domains are now being explored and solved.

The goal of the present workshop is to support the interaction and integration of symbolic and numeric computing. Earlier meetings in this series include the SNAP 96 Workshop, held in Sophia Antipolis, France, the SNC 2005 meeting, held in Xi'an, China, SNC 2007 which was held in London, Canada, and SNC 2009, held in Kyoto, Japan. The 4th International Workshops on Symbolic-Numeric Computation will be held on June $7-9$ at San Jose, California, as a member of the ACM Federated Computing Research Conference. SNC 2011 is affiliated with the 2011 International Symposium on Symbolic and Algebraic Computation (ISSAC 2011) which will be held on June $8-11$, 2011 as another member of the ACM FCRC.

The SNC 2011 Call For Papers from all areas of symbolic-numeric computing, including:

- Hybrid symbolic-numeric algorithms in linear, polynomial and differential algebra
- Approximate polynomial GCD and factorization
- Symbolic-numeric methods for solving polynomial systems
- Resultants and structured matrices for symbolic-numeric computation
- Differential equations for symbolic-numeric computation
- Symbolic-numeric methods for geometric computation
- Symbolic-numeric algorithms in algebraic geometry
- Symbolic-numeric algorithms for nonlinear optimization
- Implementation of symbolic-numeric algorithms
- Model construction by approximate algebraic algorithms
- Applications of symbolic-numeric computation: global optimization, verification, etc.

We received 36 submissions: 10 extended abstracts and 26 full papers. Based on 99 referee reports, we accepted 8 extended abstracts and 18 full papers. We are grateful to all who contributed to the success of our meeting: the invited speakers: Jonathan Borwein, James Demmel, Stephen Watt; the authors of full papers and extended abstracts; our hard-working program committee; and many anonymous reviewers. We wish to thank our publicity chair Erik Postma, webmaster Guillaume Moroz and treasurer Werner Krandick. We would also like to thank Eric Schost and Ioannis Z. Emiris for their support. Finally, we wish to express our sincere gratitude to the following sponsors:

- Maplesoft
- Wilfrid Laurier University
- Numerical Algorithms Group
- The Ontario Research Centre for Computer Algebra
- The Association for Computing Machinery

Ilias Kotsireas
SNC 2011 General Chair
Wilfrid Laurier University
Canada

Marc Moreno Maza
SNC 2011 Proceedings Editor
University of Western Ontario
Canada

Lihong Zhi
SNC 2011 Program Committee Chair
Academy of Mathematics and Systems Science
China

Contents

Mahler measures, short walks and log-sine integrals: A case study in hybrid computation

[Extended Abstract]

Jonathan Borwein
Professor Laureate, Director
Centre for Computer Assisted Research Mathematics and its Applications (CARMA)
School of Mathematical and Physical Sciences
University of Newcastle
Callaghan NSW 2308, Australia
jon.borwein@gmail.com

Categories and Subject Descriptors

G.1.9 [**Mathematics of Computing**]: Numerical AnalysisIntegral Equations; I.1.2 [**Computing Methodologies**]: Symbolic and algebraic manipulationAlgorithms

General Terms

Theory, Experimentation

Keywords

Mahler measures, short walks, log-sine integral

The Mahler measure of a polynomial of several variables has been a subject of much study over the past thirty years. Very few closed forms are proven but many more are conjectured.

We provide systematic evaluations of various higher and multiple Mahler measures using moments of random walks and values of log-sine integrals. We also explore related generating functions for the log-sine integrals and their generalizations.

This work would be impossible without very extensive symbolic and numeric computations. It also makes frequent use of the new NIST Handbook of Mathematical Functions.

My intention is to show off the interplay between numeric and symbolic computing while exploring the three topics in title.

Accurate and efficient expression evaluation and linear algebra, or Why it can be easier to compute accurate eigenvalues of a Vandermonde matrix than the accurate sum of 3 numbers

[Extended Abstract]

James Demmel
Department of Mathematics (Computer Science Division)
University of California at Berkeley
Berkeley CA 94720-1776
demmel@cs.berkeley.edu

Categories and Subject Descriptors

G.1.3 [**Numerical Linear Algebra**]:

General Terms

Performance

Keywords

Eigenvalues, accurate numerical linear algebra, structured matrices

We survey and unify recent results on the existence of accurate algorithms for evaluating multivariate polynomials, and more generally for accurate numerical linear algebra with structured matrices. By "accurate" we mean the computed answer has relative error less than 1, i.e. has some leading digits correct. We also address efficiency, by which we mean algorithms that run in polynomial time in the size of the input. Our results will depend strongly on the model of arithmetic: Most of our results will use what we call the *Traditional Model (TM)*, that the computed result of $op(a, b)$, a binary operation like $a + b$, is given by $op(a, b) * (1 + \delta)$ where all we know is that $|\delta| \leq \varepsilon \ll 1$. Here ε is a constant also known as machine epsilon.

We will see a common reason that the following disparate problems all permit accurate and efficient algorithms using only the four basic arithmetic operations: finding the eigenvalues of a suitably discretized scalar elliptic PDE, finding eigenvalues of arbitrary products, inverses, or Schur complements of totally nonnegative matrices (such as Cauchy and Vandermonde), and evaluating the Motzkin polynomial. Furthermore, in all these cases the high accuracy is "deserved", i.e. the answer is determined much more accurately by the data than the conventional condition number would suggest.

In contrast, we will see that evaluating even the simple polynomial $x + y + z$ accurately is impossible in the TM, using only the basic arithmetic operations. We give a set of necessary and sufficient conditions to decide whether a high accuracy algorithm exists in the TM, and describe progress toward a decision procedure that will take any problem and either provide a high accuracy algorithm or a proof that none exists.

When no accurate algorithm exists in the TM, it is natural to extend the set of available accurate operations $op()$ by a library of additional operations, such as $x + y + z$, dot products, or indeed any enumerable set which could then be used to build further accurate algorithms. We show how our accurate algorithms and decision procedure for finding them can be extended in this case.

Finally, we address other models of arithmetic, and address the relationship between (im)possibility in the TM with (in)efficient algorithms operating on numbers represented as bit strings.

Polynomial Approximation in Handwriting Recognition

[Extended Abstract]

Stephen M. Watt
Dept of Computer Science
University of Western Ontario
London, Ontario, Canada
Stephen.Watt@uwo.ca

ABSTRACT

Considering digital ink traces as plane curves provides a useful framework for handwriting recognition. Characters may be represented as parametric curves approximated by certain truncated orthogonal series, mapping symbols to the low-dimensional vector space of series coefficients. Many useful properties are obtained in this representation, allowing fast recognition based on small training sets. The beauty of this framework is that a single, coherent view leads to highly efficient methods with a high recognition rate. Furthermore, these truncated orthogonal series are subject to all the geometric techniques of symbolic-numeric polynomial algorithms.

Categories and Subject Descriptors

I.5.1 [**Pattern Recognition**]: Models; I.7.5 [**Document and Text Processing**]: Document Capture—*Optical character recognition*; G.1.2 [**Numerical Analysis**]: Approximation

General Terms

Algorithms

Keywords

Handwriting recognition, orthogonal series, approximation

1. INTRODUCTION

Machine-based handwriting recognition has been studied now for more than a century. In 1910, Hyman Goldberg proposed recognizing handwriting using electically conducting ink [1]. Since then, the subject of handwriting recognition has grown and flourished. Handwriting recognition is essential to major economic activities, such as cheque processing and mail sorting, and is a standard feature on many mobile electronic devices.

There is by now a vast literature on the subject of handwriting recognition by computer, divided between "off-line" and "on-line" recognition. Off-line recognition takes a static image of some handwriting and produces text. The input is typically an image which may involve background noise, digitization artifacts and distortion. On-line recognition takes motions and other events, such as button presses, pen up and pen down, and produces text. A variety of capture devices may be used, including digitizing tablets, screen overlays or cameras. The captured pen movements and related events may be called "digital ink" regardless of the source, and which may be stored and transmitted in a number of ways, including InkML [2]. On-line recognition is often regarded as an easier problem because the writing order is given, the identification of the input is evident and mis-recognitions can be corrected. On the other hand, processing time becomes a constraint and there is no forward context.

A problem of particular interest is that of mathematical handwriting recognition. High quality mathematical handwriting recognition would be useful for expression entry and editing in both document processing systems and mathematical software, such as computer algebra systems. We are therefore interested in on-line methods. The usual techniques used for natural languages cannot, however, be applied to written mathematics. There are a number of difficulties: First, a dizzying array of symbols from many different alphabets are used at once. Second, many similar characters, which must be distinguished, are written with just a few strokes. For example, Figure 1 shows a progression of symbols with similar features all written with one stroke. Third, the lay out is two dimensional and relative positioning matters. Moreover the symbols are typically in several sizes leading to ambiguous juxtapositions, as shown in Figure 2. (Are the first two symbols a^p or aP?) Fourth, in mathematical handwriting, there is not a useful fixed dictionary of words that may be used to rule out senseless letter combinations. Almost any sequence of symbols potentially has meaning. On the other hand, symbols tend to be well separated with a known orientation.

With so many symbols, each with variants, and new ones frequently added, the usual techniques of symbol recognition are difficult to apply. It is impractical do develop hand-tuned heuristics to recognize specific features for each symbol. Matching against a comprehensive database of models is too time consuming. Neural nets require massive amounts of training data not readily available for rarely used symbols. Instead, in a series of papers, we have developed a framework suitable for this setting, allowing high recognition rates at high speed with only modest amounts of training data.

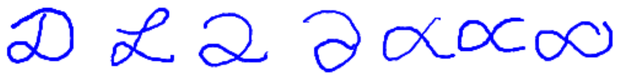


Figure 1: Similar single-stroke characters

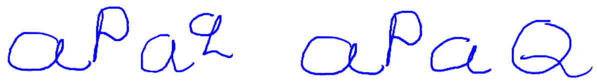


Figure 2: Juxtaposition ambiguity

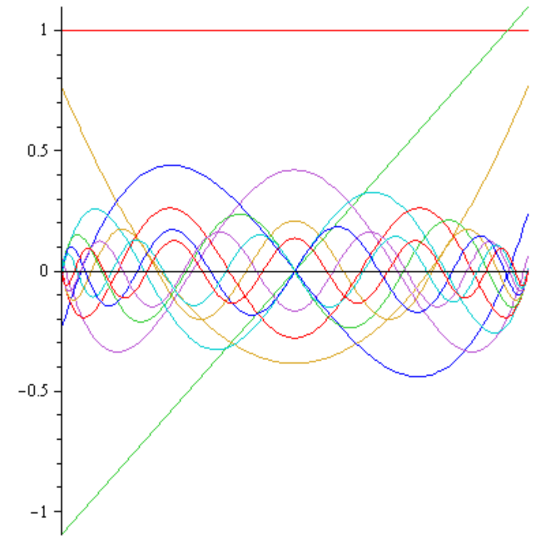


Figure 3: L-S polynomials on $[0,1]$ for $\mu = 1/8$

2. FRAMEWORK

One of the main difficulties we find with other recognition methods is that the digital ink is thought of as a set of points, and subject to various *ad hoc* treatments, such as smoothing to eliminate device jitter and "re-sampling", i.e. interpolation to add or remove or equalize the distance between points. This is followed by various numerical heuristics to try to identify features such as cusps, self intersections, etc. The problem with this is that we are not treating the ink traces as what they really are, curves.

We consider an ink trace to be a segment of a plane curve $(x(\lambda), y(\lambda))\,, \lambda \in [0, L]$. Various parameterizations are possible, including parameterization by time (as the curve was traced), by arc length or by affine arc length. We have found over the course of various experiments that arc length is the most robust parameterization in most cases, which makes intuitive sense since this gives curves that look the same the same parameterization.

The next step is to realize that curves that are "almost the same" should be recognized as the the same symbol. We may therefore work with approximations in a finite dimensional function space:

$$x(\lambda) \approx \sum_{i=0}^{d} x_i B_i(\lambda), \qquad y(\lambda) \approx \sum_{i=0}^{d} y_i B_i(\lambda).$$

By appropriate choice of basis functions B_i, $i = 0, \ldots, d$, the approximations can be made arbitrarily close to x and y. After normalizing for position and scale, the coefficients represent the curve as a $(2d-1)$-dimensional point. If the basis functions are orthogonal with respect to a functional inner product, then we can obtain the coefficients (x_i, y_i) by numerical integration. One of the most appealing aspects to this is that this completely captures the curves in a manner independent of the resolution of the device and allows us to ask geometrically meaningful questions.

The first step in this approach used Chebyshev polynomials as the basis functions [3]. The non-linearity of the weight function $1/\sqrt{1-\lambda^2}$ required first capturing the entire curve, normalizing it, then computing the series coefficients. It was then shown that a Legendre polynomial basis allowed the coefficients to be calculated instantly on pen-up from moments of $x(\lambda)$ and $y(\lambda)$ integrated as the curve is written [4]. This real-time property is preserved using a Legendre-Sobolev basis, orthogonal with respect to the inner product

$$\langle f, g \rangle = \int_a^b f(\lambda)g(\lambda)d\lambda + \mu \int_a^b f'(\lambda)g'(\lambda)d\lambda,$$

for which orthogonal series approximate both the function and its derivative well [5]. An orthogonal polynomial basis may be obtained for this or any desired inner product by Gram-Schmidt orthogonalization of the monomial basis $\{1, \lambda, \lambda^2, ...\}$. The Legendre-Sobolev polynomials of degrees 0 to 12 for $a = 0, b = 1, \mu = 1/8$ are shown in Figure 3. As the representation by series coefficients is space efficient, it may be used for digital ink compression [6].

3. DISTANCE-BASED RECOGNITION

Representing handwritten mathematical symbols as points in such an inner product space has a number of pleasant properties. One property of using orthonormal series is that distances in the function space become Euclidean distances in the coefficient space:

$$||f - g|| \approx \sqrt{\sum_{i=0}^{d} (f_i - g_i)^2}$$

as the integrals of the $B_i B_{j \neq i}$ cross terms vanish by orthogonality. This allows the variational integrals approximated by elastic matching to be calculated extremely quickly.

We find that classes of like symbols form clusters, and that the classes are convex even at low dimension [7, 8]. This implies that linear homotopies between points in the same class should have intermediate points that remain in the same class. Indeed this is observed, as illustrated in Figure 4. This property allows us to classify points based on distances to convex hulls of sets of points, rather than to particular point or their averages, which is more robust when the training sets are small. A comparison of recognition methods using elastic matching with dynamic time warping (re-parameterization), and Euclidean and Manhattan distances in the orthogonal series coefficient space [9] shows that for many choices of dimension d and coefficient size,

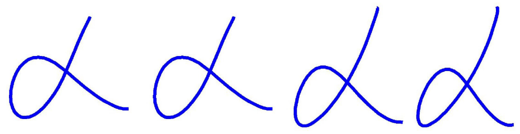


Figure 4: Linear homotopy stays within the class

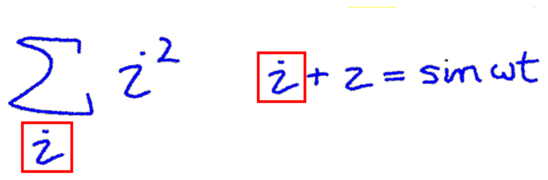


Figure 5: Context dependency: i or $\dot{z}$

the coefficient-based methods give similar results to elastic matching, but may be computed much faster. Alternatively to convex hulls of the known points, one may use the polyhedra bordered by the planes of SVM ensembles [10] and these may additionally make use of other features [11].

Even with the best individual character recognition, ambiguities remain. Consider the two expressions shown in Figure 5. Even though the symbol shown in the red box is exactly the same in both cases, in one case it should be recognized as "i" and in the other as "$\dot{z}$". It is therefore useful to be able to use our distance-based methods in conjunction with contextual information, such as frequency information for mathematical n-grams [12, 13, 14]. In this regard, a useful distance-based confidence measure may be obtained for classification using either distances to convex hulls of classes or to separating planes in SVM ensembles [15].

To this point, we have discussed classification based on functional approximation of the coordinate curves $x(\lambda)$ and $y(\lambda)$. If the orientation of the characters is not certain, or if they are deformed in other ways, then it is natural to seek methods that are invariant under these transformations. We have seen that similar methods may be applied to integral invariants of the coordinate curves to classify symbols with unknown rotation or shear [16, 17].

4. ON TO APPROXIMATE POLYNOMIALS

The objects with which we are working, these truncated series, are polynomials with approximate coefficients in a non-monomial basis. Given the vast body of work in approximate polynomial algebra, e.g. [18, 19, 20, 21, 22], we ask what ideas may be brought to bear from this community. This points to several directions for investigation.

It is easy in this representation to find all the critical points of digital ink traces. Many of these, such as self intersection, number of local maxima, etc, have long been used as features for recognition. The usual methods for detecting these features depend significantly on device resolution. With rapidly evolving technology, this means that newly adjusted algorithms become necessary, and these cannot use archival data directly. Instead, finding these critical points from the polynomial approximations is robust against changes in device resolution.

We give an example. Consider the digital ink trace of a lower case letter d, shown in Figure 6(a). The trace consists of about 300 data points sampling the x and y coordinates and pressure at a uniform frequency. Figure 6(b) shows an approximation in parametric form $(x(\lambda), y(\lambda))$ for $\lambda \in [0, 1]$, with x and y being Legendre Sobolev series over $\lambda \in [0, 1]$ with $\mu = 1/8$ up to degree 12. Figure 6(c) shows the critical points found by solving $x'(\lambda) = 0$ and $y'(\lambda) = 0$. This is achieved by univariate polynomial root finding, retaining real roots in the interval $[0, 1]$.

The key point is that it is meaningful to perform useful analysis efficiently on the traces as curves, rather than considering the trace as a collection of discrete points. To illustrate this concretely, consider the critical point on the top of the body of the letter d. This is the second critical point from the beginning of the trace. Finding the critical point from the polynomial approximation can be done by a fast Newton iteration. This takes into account the effect of all the sample points on the local behaviour. In contrast, consider trying to find this local maximum from the discrete sample points. A portion of the trace data is shown in Figure 7, magnified more vertically than horizontally. We see that the local maximum should occur somewhere near the five sample points with approximately equal y value, but have to construct heuristics to compute a value. For example, in deciding which points are at the maximum, what error tolerance should be used? If several points are within the error tolerance of being the maximum, whereabouts in that point set should the maximum be taken? Should the maximum be taken as the maximum value achieved by one of the sample points, or should the maximum of a local spline be used? All this is avoided by working with curves instead of points.

Some operations can be more natural on implicit curve models. This is obtained directly from the parametric representation as $Resultant(X - x(\lambda), Y - y(\lambda), \lambda)$. For the example, the implicit polynomial obtained this way to approximate the example trace is plotted in Figure 6(d). This polynomial of degree 12 in X and Y has 91 terms. (A plotting artefact loses part of the tail.)

5. FUTURE DIRECTIONS

There appears to be a fertile ground for further work in this area. It should be useful to maintain the perspective that perturbations are to be minimized in the Legendre-Sobolev space instead of with respect to polynomial norms in a monomial basis. Thus we would want to compute resultants, SVDs, etc, in this basis, rather than perform ill-conditioned conversions. Some new results are required in this area in order to proceed.

Two of the main ideas in symbolic-numeric algorithms for polynomials are that of backward error and semi-definite programming. With backward error, we may ask questions such as whether there is a near by polynomial (i.e. requiring perturbation by less than a given bound) that has certain properties, e.g. singularity, factorization, etc. With semi-definite programming we can ask questions about the least perturbations required.

These tools provide a most useful opening for symbolic-numeric computation in handwriting recognition. Rather than *ad hoc* numerical techniques based on sample points, we have a framework in which to ask well-posed questions about nearby curves and have a systematic, meaningful approach to answering them.

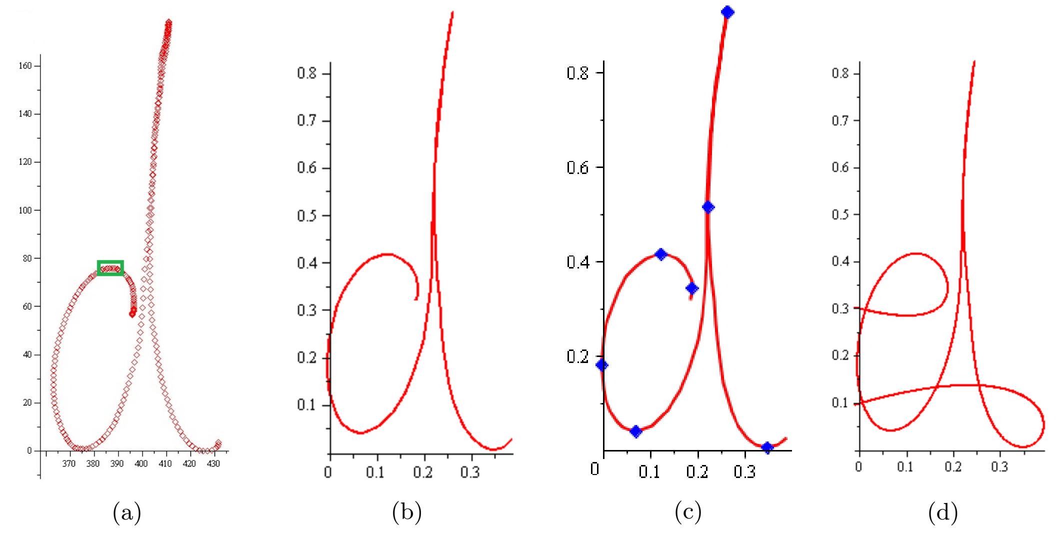


Figure 6: Analysis of an input symbol:
(a) Trace data (green box magnified in Figure 7),
(b) Parametric approximation $(x(\lambda), y(\lambda))$ $\lambda \in [0,1]$,
(c) Critical points computed from $(x(\lambda), y(\lambda))$,
(d) Implicit approximation $P(X,Y) = 0$.

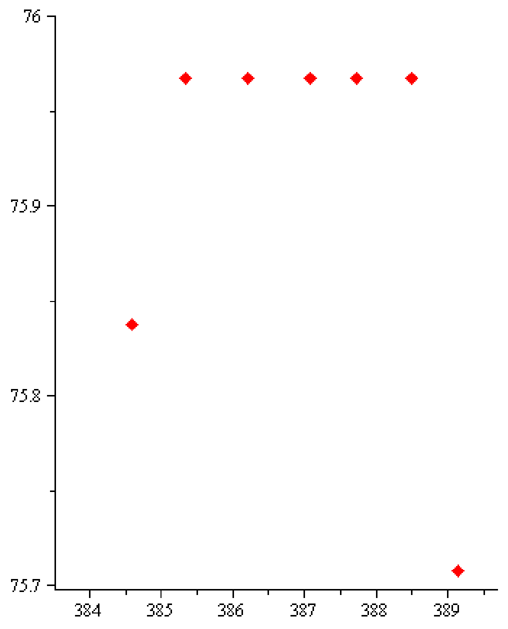


Figure 7: Top of d body, magnified more vertically

6. REFERENCES

[1] Goldberg, H.E: US Patent No. 1,117,184. Filed November 23, 1910. Granted November17, 1914.

[2] Watt, S.M. and Underhill, T. (eds): Ink Markup Language (InkML). W3C Proposed Recommendation 10 May 2011 `http://www.w3.org/TR/2011/PR-InkML-20110510` .

[3] Char, B.W. and Watt, S.M.: Representing and Characterizing Handwritten Mathematical Symbols Through Succinct Functional Approximation. pp. 1198–1202, Proc. Int'l Conf. Document Analysis and Recognition (ICDAR), IEEE Press 2007.

[4] Golubitsky, O. and Watt, S.M.: Online Stroke Modeling for Handwriting Recognition. pp. 72-80, Proc. 18th Annual Int'l Conf. Computer Science and Software Engineering (CASCON 2008), IBM Canada 2008.

[5] Golubitsky, O. and Watt, S.M.: Online Computation of Similarity between Handwritten Characters. pp. C1-C10, Proc. Document Recognition and Retrieval XVI (DRR XVI), SPIE and IS&T 2009.

[6] Mazalov, V. and Watt, S.M.: Digital Ink Compression via Functional Approximation. pp. 688-694, Proc. Proc. 12th Int'l Conf. Frontiers in Handwriting Recognition (ICFHR 2010), IEEE Computer Society 2010.

[7] Golubitsky, O. and Watt, S.M.: Online Recognition of Multi-Stroke Symbols with Orthogonal Series. pp. 1265-1269, Proc. 10th Int'l Conf. Document Analysis and Recognition (ICDAR 2009), IEEE Computer Society 2009.

[8] Golubitsky, O. and Watt, S.M.: Improved Classification through Runoff Elections. pp. 59–64, Proc. Int'l Workshop on Document Analysis Systems (DAS 2010), ACM 2010.

[9] Golubitsky, O. and Watt, S.M.: Distance-Based Classification of Handwritten Symbols. Int'l Journal on Document Analysis and Recognition, Vol. 13, No. 2, pp. 133-146, Springer 2010.

[10] Keshari, B. and Watt, S.M.: Online Mathematical Symbol Recognition using SVMs with Features from Functional Approximation. Electronic Proc. Mathematical User-Interfaces Workshop 2008 (MathUI 2008), Activemath.org 2008.

[11] Keshari, B. and Watt, S.M.: Hybrid Mathematical Symbol Recognition Using Support Vector Machines. pp. 859–863, Proc. Int'l Conf. Document Analysis and Recognition (ICDAR), IEEE Press 2007.

[12] So, Clare M. and Watt, S.M.: Determining Empirical Properties of Mathematical Expression Use. pp. 361-375, Proc. Fourth Int'l Conf. Mathematical Knowledge Management (MKM 2005), LNCS 3863, Springer 2005.

[13] Watt, S.M.: An Empirical Measure on the Set of Symbols Occurring in Engineering Mathematics Texts. pp. 557-564, Proc. 8th IAPR Int'l Workshop on Document Analysis Systems (DAS 2008), IEEE Computer Society 2008.

[14] Smirnova, E. and Watt, S.M.: Context-Sensitive Mathematical Character Recognition. pp. 604–610, Proc. IAPR Int'l Conf. Frontiers in Handwriting Recognition (ICFHR 2008), Cenparmi, Concordia University 2008, ISBN 1-895193-03-6.

[15] Golubitsky, O. and Watt, S.M.: Confidence Measures in Recognizing Handwritten Mathematical Symbols. pp. 460-466, Proc. Conf.s on Intelligent Computer Mathematics 2009, LNAI 5625, Springer 2009.

[16] Golubitsky, O., Mazalov, V. and Watt, S.M.: Orientation-Independent Recognition of Handwritten Characters with Integral Invariants. pp. 252-261, Proc. Joint Conf. of ASCM 2009 and MACIS 2009 (ASCM 2009), COE Lecture Note Vol. 22, Kyushu U. ISSN 1881-4042.

[17] Golubitsky, O., Mazalov, V. and Watt, S.M.: Towards Affine Recognition of Handwritten Mathematical Characters. pp. 35–42, Proc. Int'l Workshop on Document Analysis Systems (DAS 2010), ACM 2010.

[18] Watt, S.M. and Stetter, H.J. (eds): Symbolic-Numeric Algebra for Polynomials. J. Symbolic Computation Special Issue, Vol. 26 No 6. 1998.

[19] Wang, Dongming and Zhi, Lihong (eds): Symbolic-Numeric Computation (SNC 2005). Birkhäuser 2007.

[20] Watt, S. and Verschelde, J.: Proc. 2007 Conf. Symbolic Numeric Computation (SNC 2007). ACM 2007.

[21] Kai, Hiroshi and Sekigawa, Hiroshi (eds): Proc. 2009 Conf. Symbolic Numeric Computation (SNC 2009). ACM 2009.

[22] Moreno Maza, M. (ed): Proc. 2011 Conf. Symbolic Numeric Computation (SNC 2011). ACM 2011.

Numerical Calculation of H-bases for Positive Dimensional Varieties

Barry H Dayton
Department of Mathematics
Northeastern Illinois University
Chicago, IL 60625, USA
bhdayton@neiu.edu

ABSTRACT

A symbolic-numeric method for calculating an H-basis for the ideal of a positive dimensional complex affine algebraic variety, possibly defined numerically, is given. H-bases for ideals $\mathcal{I}$ in $\mathbb{C}[\mathbf{x}] = \mathbb{C}[x_1, \dots, x_s]$, introduced by Macaulay and later studied by Möller and Sauer, are an analog of Gröbner bases with respect to a global degree ordering: f is an element of $\mathcal{I}$ of total degree n if and only f is a $\mathbb{C}$-linear combination of polynomials of total degree n or less which are monomial multiples of members of the H-basis. The method uses the interplay of local and global duality.

Applications include factoring multivariable polynomials, analyzing singular curves, finding equations for the union of varieties, and, most importantly, finding equations for components of reducible varieties given numerically.

Categories and Subject Descriptors

G.1.5 [**Roots of Nonlinear Equations**]: Systems of Equations

General Terms

Algorithms, Theory

Keywords

H-basis, Dual Functionals, Macaulay Array, Sylvester Array, Positive dimensional varieties, Symbolic Numeric Computation

1. INTRODUCTION

We introduce a heuristic symbolic-numeric method to find a H-basis [14, 15] for an ideal $\mathcal{I}$ in $\mathbb{C}[\mathbf{x}] = \mathbb{C}[x_1, \dots, x_s]$ provided the zero set $V(\mathcal{I})$ is positive dimensional or has only a few isolated zeros. This method, which makes use of both the original equations which locally define $V(\mathcal{I})$ and numerically approximate points, generally works better than interpolation and requires fewer points. Although the method was developed to handle systems given numerically, experiments on exact systems show that, using machine precision (about 17 digits) numerical calculation, one can generally approximate exact solutions to 9 or more digits for moderate sized systems.

These algorithms make use of the interplay between local and global duality via Macaulay and Sylvester matrices. The H-basis will generate the Sylvester matrices of the ideal for all orders which, like Gröbner bases, will provide a decision process for deciding if a polynomial is in the ideal and, if so, write it in terms of the basis, see Lemma 1. Möller and Sauer [15] show how, given an inner product on $\mathbb{C}[\mathbf{x}]$, normal forms may be calculated.

When finding the ideal of a component of a reducible variety straight forward interpolation methods work poorly in the positive dimensional case and there are many solutions; therefore picking a good generating system is not realistic. Sommese *et. al.* have suggested algorithms in [18, 2]. These require numerical irreducible decomposition and may return only a set theoretic basis whereas our method will generally give an ideal theoretic basis. They recently have proposed an algorithm [3] that will return a system with integer coefficients, when available, using the LLL or similar algorithms. However this method does not apply to some of the systems discussed in this paper.

H-bases are specifically introduced to handle affine systems and are numerical friendly analogs of Gröbner bases given from global degree orderings, for example the degree reverse lexicographical `dp` ordering of [9, §1.2]. Example 1 shows that in general H-bases are larger than arbitrary ideal generating sets but are often smaller than Gröbner bases. If the ideal of the system is homogeneous, then any homogeneous basis for the ideal is an H-basis which may explain the use of the letter H.

The algorithms are implemented using Macaulay arrays and SVD based numerical linear algebra to calculate the dual vectors. Various authors [13, 16, 21, 22] have used other methods to avoid the large matrices. These strategies may be useful in the calculation of §5.1. In this paper we avoid these strategies for the following reasons: 1) the complexity of these methods is an unnecessary distraction from the subject at hand, 2) we want to emphasize the interplay of Macaulay and Sylvester arrays and 3) for most of our examples using sparse matrices makes the Macaulay method fast enough that, from a practical point view, the more sophisticated methods are not necessary and may actually be slower.

The method here is inherently numerical, there is no exact analog. For instance in Example 4 we factor a multivariate polynomial using a single numerical point on its variety as opposed to a-priori knowledge, which would be required by any exact method, of the extension field over which the factors are defined .

All computations in this paper were done with Mathematica. Timings were taken using both Mathematica 6 running on a Linux system and Mathematica 7 running on Windows. To be conservative the slower Mathematica 6 times are reported.

2. SYLVESTER AND MACAULAY ARRAYS

Let $F = [f_1, f_2, \dots, f_t]$ be a list of t polynomial functions in s variables $x_1, \dots, x_s$ and $\hat{\mathbf{x}} = (\hat{x}_1, \dots, \hat{x}_s)$ a point in $\mathbb{C}^s$.

For a non-negative integer list $\mathbf{j} = [j_1, \dots, j_s]$ write $|\mathbf{j}| = j_1 + j_2 + \cdots + j_s$ and then $\mathbf{x}^{\mathbf{j}} = x_1^{j_1} x_2^{j_2} \dots x_s^{j_s}$ is a monomial in $\mathbb{C}[\mathbf{x}] = \mathbb{C}[x_1, \dots, x_s]$ of *total degree* $|\mathbf{j}|$. The *total degree* of a polynomial in these variables is the total degree of the largest monomial. Let $(\mathbf{x} - \hat{\mathbf{x}})^{\mathbf{i}}$ denote $(x_1 - \hat{x}_1)^{i_1} \dots (x_s - \hat{x}_s)^{i_s}$. Assume a global degree ordering is given on the monomials of $\mathbb{C}[\mathbf{x}]$. As in [19, 7, 6] we use the differentiation operator

$$\partial_{\mathbf{x}^{\mathbf{j}}} \equiv \partial_{x_1^{j_1} \cdots x_s^{j_s}} \equiv \frac{1}{j_1! \cdots j_s!} \frac{\partial^{j_1 + \cdots + j_s}}{\partial x_1^{j_1} \cdots \partial x_s^{j_s}}. \qquad (1)$$

where we write $\partial_{\mathbf{x}^{\mathbf{j}}}[\hat{\mathbf{x}}](f)$ to indicate that we have applied the operator to function f and evaluated at point $\hat{\mathbf{x}}$.

The *Macaulay array of degree n at $\hat{\mathbf{x}}$*, $\mathbf{M}(F, k, \hat{\mathbf{x}})$ is the $t\binom{n+s-1}{s} \times \binom{n+s}{s}$ matrix with columns indexed by the differentials $\partial_{\mathbf{x}^{\mathbf{j}}}$ for $|\mathbf{j}| \le n$ or, more commonly, just by the $\mathbf{x}^{\mathbf{j}}$ ordered by a global degree ordering, we write $\mathcal{X}_n$ for the ordered list of monomials of total degree n or less. In particular, the left hand column has index 1 for the evaluation functional. The rows will be indexed by the functions $\mathbf{x}^{\mathbf{i}} f_\alpha$ for $|\mathbf{i}| < k$, $\alpha = 1, \dots, t$. Again these will be grouped by degree $|\mathbf{i}|$ and by monomial $\mathbf{x}^{\mathbf{i}}$ in our ordering. In particular the first t rows are indexed by $f_1, \dots, f_t$.

The entry in the row indexed by $\mathbf{x}^{\mathbf{i}} f_\alpha$ and column indexed by $\mathbf{x}^{\mathbf{j}}$ is $\partial_{\mathbf{x}^{\mathbf{j}}}[\hat{\mathbf{x}}]((\mathbf{x} - \hat{\mathbf{x}})^{\mathbf{i}} f_\alpha)$.

The *Sylvester Array of degree k*, $\mathbf{S}(F, k)$ of the list $F = [f_1, \dots, f_t]$ is the submatrix of $\mathbf{M}(F, k, \hat{\mathbf{0}})$, where $\hat{\mathbf{0}}$ is the origin, consisting of rows so that the total degree of the index polynomial $\mathbf{x}^{\mathbf{i}} f_\alpha$ is less than or equal to k. The entry in the row indexed by $\mathbf{x}^{\mathbf{i}} f_\alpha$ and column indexed by $\mathbf{x}^{\mathbf{j}}$ is the coefficient of the monomial $\mathbf{x}^{\mathbf{j}}$ in the polynomial $\mathbf{x}^{\mathbf{i}} f_\alpha$. In the Macaulay array the polynomials may be truncated, for the Sylvester array there is no truncation since only polynomials of total degree k or less are used.

Denote $\mathcal{I} = \langle f_1, \dots, f_t \rangle$ to be the ideal generated by the f_i in $\mathbb{C}[\mathbf{x}]$ and put $A = \mathbb{C}[\mathbf{x}]/I$. Then the row spaces of Macaulay arrays can be identified with a filtration of the maximal ideal of the local ring $\mathcal{O}_{\hat{\mathbf{x}}}(A)$, [4], while the row spaces of the Sylvester arrays filter the ideal $\mathcal{I}$ so are associated with the global structure. But note while every element of $\mathcal{I}$ is of the form of a finite sum $g = \sum c_{\mathbf{i}} x^{\mathbf{i}} f_\alpha$ the total degrees of some of the terms in this sum may be larger than the total degree of g due to some cancellation of terms. So not all the polynomials of total degree k or less may be in the sub-vector space of $\mathcal{I}$ associated with $\mathbf{S}(F, k)$, see Example 1 below. If we want to include such polynomials we will use the notation $\mathbf{S}(\mathcal{I}, k)$ for the Sylvester array of some set G so that the rowspace $\mathbf{S}(G, k)$ contains the rowspace of $\mathbf{S}(\{f\}, k)$ for all polynomials f of degree k or less in $\mathcal{I}$. Note that $\mathbf{S}(\mathcal{I}, k)$ is only defined up to row space. This leads to a restatement of our main definition:

Definition 1. *Let $\mathcal{I}$ be an ideal of $\mathbb{C}[\mathbf{x}]$. If $\mathcal{B}$ is a set of polynomials in $\mathcal{I}$ such that $\mathbf{S}(\mathcal{B}, k) = \mathbf{S}(\mathcal{I}, k)$ for all $k \ge 1$ then $\mathcal{B}$ will be called an H-basis for $\mathcal{I}$. If no proper subset of $\mathcal{B}$ is an H-basis we will call $\mathcal{B}$ a minimal H-basis.*

For a polynomial f borrowing notation from [9] we write $\mathrm{jet}(f, n)$ for the polynomial consisting of all terms of f of degree n or less. We write $\mathrm{vjet}(f, n)$ for the vector consisting of all coefficients of $\mathrm{jet}(f, n)$ in the ordering given by $\mathcal{X}_n$, i.e. $\mathrm{vjet}(f, n)\mathcal{X}_n = \mathrm{jet}(f, n)$. In particular $\mathrm{vjet}(f, n)$ would be the row indexed by f in $\mathbf{M}(\{f\}, n, \mathbf{0})$ so $\mathbf{S}(\mathcal{I}, n)$ could be described by the property that $\mathrm{vjet}(f, n)$ is in the rowspace of $\mathbf{S}(\mathcal{I}, n)$ for all $f \in \mathcal{I}$ of total degree n or less.

The existence and main property of a minimal H-basis are given by

Lemma 1. *Let $\mathcal{I}$ be an ideal of $\mathbb{C}[x_1, \dots, x_s]$. Then*

i) A finite H-basis for $\mathcal{I}$ exists.

ii) If $\mathcal{B}$ is an H-basis for $\mathcal{I}$ then given $f \in \mathbb{C}[\mathbf{x}]$ of total degree n, $f \in \mathcal{I}$ if and only if $\mathrm{vjet}(f, n)$ *is in the rowspace of* $\mathbf{S}(\mathcal{B}, n)$.

Proof: For i) note that any Gröbner basis with respect to a global degree ordering [9, §1.2] contains an H-basis. Or see [15, Theorem 2.3 (ii)]. ii) follows from the definition of $\mathbf{S}(\mathcal{I}, n)$.

From Part i) finite minimal H-bases exist for each each ideal of $\mathbb{C}[\mathbf{x}]$. Part ii) of Lemma 1 gives the recognition criteria for membership in $\mathcal{I}$ and writing $f \in \mathcal{I}$ in terms of the H-basis is then a matter of linear algebra.

Example 1: Consider the ideal $\mathcal{I}$ of $\mathbb{C}[x, y, u, v]$ generated by $F = [x - y + uy + vy, -x + ux + vx + y, -1 + u + v + xy]$. Then rank $\mathbf{S}(F, 2) = 3$ but rank $\mathbf{S}(\mathcal{I}, 2) = 5$ because of the additional degree 2 linearly independent polynomials $-x^2 + y^2, 1 - 2u + u^2 - 2v + 2uv + v^2 - x^2$. An H-basis consists of these 5 polynomials. The Gröbner bases given by Mathematica using the DegreeReverseLexicograpic ordering or by Singular using the dp ordering require also the cubic $x^3 - x$. However $\mathrm{vjet}(x^3 - x, 3)$ is already in the rowspace of the Sylvester array of degree 3 given by the H-basis above so is not needed in the H-basis.

3. THE DUAL SPACE OF DIFFERENTIAL FORMS

3.1 Local and Global Duality

Dual spaces on polynomial ideals have been studied by [14, 10, 8, 16] and also by [7, 6, 19], however the former use a global point of view while the latter take a local point of view with the exception of [19] who attempts to combine the two ideas. Since the main object of study by both groups are zero-dimensional rings the difference may not be apparent, however we are applying these to positive dimensional rings and the important distinction needs to be noted. We begin by briefly reviewing the two concepts of dual spaces in the papers cited above.

For the global point of view a *global differential functional* on $\mathbb{C}[\mathbf{x}] = \mathbb{C}[x_1, \dots, x_s]$ is a formal infinite sum $\mathbf{D} = \sum_{\mathbf{k}} c_{\mathbf{k}} \mathbf{X}^{\mathbf{k}}$ where $\mathbf{k}$ runs over all lists $\mathbf{k} = [k_1, \dots, k_s]$ where the k_i are non-negative integers. The $\mathbf{X^k}$ are the functionals defined for each monomial $\mathbf{x^j}$ by

$$\mathbf{X^k}(\mathbf{x^j}) = \begin{cases} 1 & \text{if} \quad \mathbf{j} = \mathbf{k}, \\ 0 & \text{if} \quad \mathbf{j} \neq \mathbf{k}. \end{cases}$$

This extends $\mathbf{D}$ to a linear functional over all polynomials in $\mathbb{C}[\mathbf{x}]$. If $\mathcal{I}$ is an ideal of $\mathbb{C}[\mathbf{x}]$ we say $\mathbf{D}$ is in the *dual space* of $\mathcal{I}$ if $\mathbf{D}(f) = 0$ for all $f \in \mathcal{I}$. This makes D a linear functional on $\mathbb{C}[\mathbf{x}]/\mathcal{I}$ By abuse of language we sometimes say $\mathbf{D}$ is a *a linear functional on $\mathcal{I}$.*

If $\mathbf{D} = \sum_{\mathbf{k}} c_{\mathbf{k}} \mathbf{X^k}$ we write $\text{jet}(\mathbf{D}, p) = \sum_{|\mathbf{k}| \leq p} c_{\mathbf{k}} \mathbf{X^k}$. Then $\text{vjet}(\mathbf{D}, p)$ will be the vector of coefficients $c_{\mathbf{k}}$ indexed as in the Macaulay and Sylvester matrices. We have

Lemma 2. *Suppose $F = [f_1, \ldots, f_t]$ and $\mathcal{I}$ is the ideal $\mathbb{C}[\mathbf{x}]\,F$ generated by the f_i. Let $\mathbf{D} = \sum_{\mathbf{k}} c_{\mathbf{k}} \mathbf{X^k}$ be a differential functional. The following are equivalent.*

i) $\mathbf{D}$ is a differential functional on $\mathcal{I}$.

ii) For all $n > 0$ $\text{vjet}(\mathbf{D}, n)$ is in the nullspace of $\mathbf{S}(F, n)$.

iii) For all $n > 0$ $\text{vjet}(\mathbf{D}, n)$ is in the nullspace of $\mathbf{S}(\mathcal{I}, n)$.

Proof: From the definitions i) is equivalent to iii) and iii) implies ii). But if $g \in \mathcal{I}$ then the coefficient vector of g is contained in the row space of $\mathbf{S}(F, d)$ for some d and hence $\mathbf{D}(g) = 0$. Thus ii) implies i).

On the other hand a *local differential functional* on $\mathbb{C}[\mathbf{x}]$ is a finite sum $\mathbf{d} = \sum_{\mathbf{x^j}} c_{\mathbf{x^j}} \partial_{\mathbf{x^j}}[\hat{\mathbf{x}}]$ where $\mathbf{x^j}$ runs over a finite set monomials in $\mathbb{C}[\mathbf{x}]$, each $c_{\mathbf{x^j}}$ is a complex number and $\partial_{\mathbf{x^j}}[\hat{\mathbf{x}}]$ was defined in (1). Local differential functionals are the differential functionals used in [19, 7, 6]. $\hat{\mathbf{x}}$ is a point we will call the *center*, this point will act as a local origin, and local differential functionals will act on analytic functions as well as polynomials [6], a fact we will not use.

Given $\hat{\mathbf{x}}$ in $V(\mathcal{I})$, the *local dual space at $\hat{\mathbf{x}}$* is the vector space of all local differentials $\mathbf{d}$ centered at $\hat{\mathbf{x}}$ such that $\mathbf{d}(g) = 0$ for all $g \in \mathcal{I}$. A member of the local dual space at $\hat{\mathbf{x}}$ will also be called a local differential functional on $\mathbb{C}[\mathbf{x}]/\mathcal{I}$, or on $\mathcal{O}_{\hat{\mathbf{x}}}(\mathbb{C}[\mathbf{x}]/\mathcal{I})$ or on $\mathcal{I}$ for an ideal $\mathcal{I}$

The *order* of a local differential functional $\mathbf{d}$, written $d = \text{ord}(\mathbf{d})$ is the largest integer d so that $c_{\mathbf{x^j}} \neq 0$ for some $|\mathbf{j}| = d$. Again we can write $\text{vjet}(\mathbf{d}, p)$ for the vector of coefficients $c_{\mathbf{x^j}}$ where $|\mathbf{j}| \leq p$, if $p > \text{ord}(\mathbf{d})$ filling with 0's so that there is an entry for each index $\mathbf{j}$ with $|\mathbf{j}| \leq p$. In contrast to Lemma 2, we have from [7, 6]

Lemma 3. *Suppose $F = [f_1, \ldots, f_t]$ is a list of polynomials and $\mathcal{I}$ is the ideal of $\mathbb{C}[\mathbf{x}]$ generated by the f_i. Then $\mathbf{d}$ is a local differential functional at $\hat{\mathbf{x}}$ if and only if $\text{vjet}(\mathbf{d}, d)$ is in the nullspace of $\mathbf{M}(F, d, \hat{\mathbf{x}})$ where $d = \text{ord}(\mathbf{d})$. Moreover, $\text{vjet}(\mathbf{d}, p)$ will then be in the nullspace of $\mathbf{M}(F, p, \hat{\mathbf{x}})$ for all $p \geq d$.*

We note that although local differentials are finite sums, the vector space of all local differentials on $\mathcal{I}$ at $\hat{\mathbf{x}}$ may be infinite dimensional. In fact [6, Theorem 1] a point $\hat{\mathbf{x}}$ in $V(\mathcal{I})$ is an *isolated point* of $V(\mathcal{I})$ if and only if the local dual space of $\mathcal{I}$ is finite dimensional. On the other hand the global result, eg. [8, 16, 19], is that $V(\mathcal{I})$ is finite if and only the global dual space of $\mathcal{I}$ is finite dimensional.

It is instructive to compare our distinction of local and global with the monomial orderings of [9]. Local orderings have notation such as `ls, ds` where the "s" stands for series while global orderings are denoted by `lp, dp` etc. where "p" stands for polynomial. The local dual differential functionals here make sense in the context of analytic functions or series [6] while the global dual differential functionals only apply to polynomials.

To sum up, local forms act at a point, global forms act on the whole variety, local forms are dual to the Macaulay arrays while global forms are dual to the Sylvester matrices. These differences become most apparent working with positive dimensional algebraic sets.

3.2 Local to Global

A key feature for polynomials is that we can convert local differentials to global. Given a local differential at a point $\hat{\mathbf{x}} \in \mathbb{C}^s$ we have the following *change of center* formula:

$$\partial_{\mathbf{x^j}}[\hat{\mathbf{x}}] = \sum_{\mathbf{i} \geq \mathbf{j}} \tbinom{i_1}{j_1} \hat{x}_1^{i_1 - j_1} \cdots \tbinom{i_s}{j_s} \hat{x}_s^{i_s - j_s} \mathbf{X^i} \tag{2}$$

where $\hat{\mathbf{x}} = [\hat{x}_1, \ldots, \hat{x}_s], \mathbf{i} = [i_1, \ldots, i_s]$ and $\mathbf{j} = [j_1, \ldots, j_s]$, $\mathbf{i} \geq \mathbf{j}$ means $i_\alpha \geq j_\alpha$ for all $1 \leq \alpha \leq s$ and, for this formula, $x_i^0 = 1$ even if $x_i = 0$. Further, $\hat{\mathbf{0}}$ always denotes the actual origin in that all coordinates of $\hat{\mathbf{0}}$ are 0.

Note that the right hand side of (2) is a formal infinite sum so we do not get a local differential. But there are only finitely many terms of any degree so applying this to any polynomial gives only finitely many non-zero terms. If $\mathbf{d} = \sum c_{\mathbf{j}} \partial_{\mathbf{x^j}}$ linearity gives a global differential $\mathbf{D} = \sum \xi_{\mathbf{i}} \mathbf{X^i}[\hat{\mathbf{0}}]$. But both give the same result when acting on any polynomial $f \in \mathbb{C}[\mathbf{x}]$. So if $\mathbf{d}$ is in the local dual space to $\mathcal{I}$ at $\hat{\mathbf{x}}$, then $\mathbf{D}$ is in the global dual space to $\mathcal{I}$. We will write $\mathbf{D} = \Gamma(\mathbf{d})$ and call Γ the *globalization operator.*

If $\text{vjet}(\mathbf{d}, n)$ is in the nullspace of the Macaulay matrix $\mathbf{M}(F, n, \hat{\mathbf{x}})$ then $\text{vjet}(\mathbf{D}, n)$ is in the nullspace of the Sylvester matrix $\mathbf{S}(F, n)$ for $\mathbf{d} = \Gamma(d)$. In fact, if $\mathcal{I}$ is generated by F then Lemma 2 says that $\text{vjet}(\mathbf{D}, d)$ is in the nullspace of $\mathbf{S}(\mathcal{I}, d)$.

Conversely if $\mathbf{D}$ is in the nullspace of $\mathbf{S}(F, n)$ it need not be in the nullspace of $\mathbf{M}(F, n, \hat{\mathbf{0}})$. In particular a local differential on $\mathcal{I}$ is global but not conversely. So there is no global to local principle.

Note that (2) extends to a linear transformation on the local dual space at $\hat{\mathbf{x}}$ to the global dual space of $\mathcal{I}$. Thus, given suitable bases for the space of all differentials at $\hat{\mathbf{x}}$ and the global differentials, then there are matrices which give these transformations taking $\text{vjet}(\mathbf{d}, n)$ to $\text{vjet}(\mathbf{D}, n)$. For example if $s = 2$ and $\hat{\mathbf{x}} = (1, 2)$ then, with the monomial ordering $\mathcal{X}_3 = \{1, x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3\}$, this is the 10×10 lower triangular unit diagonal matrix in Figure 1.

$$\gamma_{\hat{\mathbf{x}}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 2 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 4 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 3 & 0 & 3 & 0 & 0 & 1 & 0 & 0 & 0 \\ 2 & 4 & 1 & 2 & 2 & 0 & 0 & 1 & 0 & 0 \\ 4 & 4 & 4 & 0 & 4 & 1 & 0 & 0 & 1 & 0 \\ 8 & 0 & 12 & 0 & 0 & 6 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 1: Change of Center Matrix

3.3 The Main Theorem

If $\mathcal{I}$ is an ideal in $\mathbb{C}[\mathbf{x}]$ we write $\mathrm{jet}(\mathcal{I}, n)$ for the $\mathbb{C}$ vector subspace of $\mathcal{I}$ of all polynomials of total degree at most n. Our main theoretical result is

Theorem 1. *Let $\mathcal{I}$ be an ideal of $\mathbb{C}[\mathbf{x}]$ and integer $n > 0$ be given. Then there exist finitely many points $\hat{\mathbf{p}}_j \in V(\mathcal{I}), j = 1, \ldots, k$ such that for $f \in \mathbb{C}[\mathbf{x}]$ of total degree no more than n, $f \in \mathrm{jet}(\mathcal{I}, n)$ if and only if $\mathbf{D}(f) = 0$ for each global vector $\mathbf{D} = \Gamma(\mathbf{d})$ where $\mathbf{d}$ is a local vector of order n or less at some $\hat{\mathbf{p}}_j$.*

Proof: Since $f \in \mathcal{I}$ if and only if f lies in the local ideal $\mathcal{I}_{\hat{\mathbf{x}}}$ for all $\hat{\mathbf{x}} \in V(\mathcal{I})$ then strict local duality given in [8, 16] shows $f \in \mathcal{I}_{\hat{\mathbf{x}}}$ if and only if $\mathbf{d}(f) = 0$ for all local differentials at $\hat{\mathbf{x}}$. Thus $f \in \mathcal{I}$ if and only if $\Gamma(\mathbf{d})(f) = 0$ for all local differentials at all points of $V(\mathcal{I})$. Moreover if the total degree of f is no more than n, $f \in \mathrm{jet}(\mathcal{I}, n)$ if and only if $\Gamma(\mathbf{d})(f) = 0$ for all $\mathbf{d}$ of order n or less. But since $\mathrm{jet}(\mathcal{I}, n) \subseteq \mathrm{jet}(\mathbb{C}[\mathbf{x}], n)$ which is finite dimensional the restriction of the dual space to dual vectors operating on $\mathrm{jet}(\mathbb{C}[\mathbf{x}], n)$ is finite dimensional. Thus it has a basis consisting of finitely many $\Gamma(\mathbf{d})$, and these can be derived from finitely many points of $V(\mathcal{I})$.

If $P = \{\hat{\mathbf{p}}_1, \ldots, \hat{\mathbf{p}}_k\}$ is a finite set of points such that the conclusion of Theorem 1 holds for $\mathcal{I}$ and N we say P *satisifies Theorem 1 for $\mathcal{I}, N$.*

From a matrix point of view Theorem 1 says that if one constructs the column matrix $\mathcal{D}_n = \mathcal{D}_n(\{\hat{\mathbf{p}}_1, \ldots, \hat{\mathbf{p}}_k\})$ of all $\mathrm{vjet}\Gamma(\mathbf{d})$ of all local dual vectors $\mathbf{d}$ of degree n or less at the points $\hat{\mathbf{p}}_j$ then the Sylvester matrix $\mathbf{S}(\mathcal{I}, k)$ is row equivalent to the left nullspace of $\mathcal{D}_n$. If $N > n$ then if f is a polynomial in $\mathcal{I}$ of degree n then $\mathrm{vjet}(f, N)$ is the vector of length $\binom{N+s}{s}$ which agrees with $\mathrm{vjet}(f, n)$ in the first $\binom{n+s}{s}$ places and is 0 in the remaining places. By Theorem 1 $\mathrm{vjet} f, N$ is then in the left null space of $\mathcal{D}_n = \mathcal{D}_N(\{\hat{\mathbf{p}}_1, \ldots, \hat{\mathbf{p}}_k\})$ for any set of points $(\{\hat{\mathbf{p}}_1, \ldots, \hat{\mathbf{p}}_k\}$ of $V(\mathcal{I})$. But because of the zeros above degree n it is seen that $\mathrm{vjet}(f, n)$ is in the left nullspace of $\mathrm{vjet}(\mathcal{D}_N, n)$, the submatrix of $\mathcal{D}_N$ consisting of the first $\binom{n+s}{s}$ rows. Conversely any vector in the left nullspace of $\mathrm{vjet}(\mathcal{D}_N, n)$ extends, by adding 0's at the end to a vector in the left nullspace of $\mathcal{D}_N$ so corresponds to a polynomial of degree n in $\mathcal{I}$ assuming $\{\hat{\mathbf{p}}_1, \ldots, \hat{\mathbf{p}}_k\}$ satisfies Theorem 1 for degree N. Thus we have shown

Corollary 1. *Suppose $\{\hat{\mathbf{p}}_1, \ldots, \hat{\mathbf{p}}_k\}$ satisfy Theorem 1 for N and $n < N$. Then the row space of the Sylvester matrix $\mathbf{S}(\mathcal{I}, n)$ is the left nullspace of $\mathrm{vjet}(\mathcal{D}_N(\{\hat{\mathbf{p}}_1, \ldots, \hat{\mathbf{p}}_k\}), n)$.*

This corollary will provide the theoretical justification of the algorithms in this paper. How many points are needed cannot be easily quantified because all points of $V(\mathcal{I})$ may not be equally good. To get a full H-basis for $\mathcal{I}$ it can be seen that it is necessary to have at least one point from each component of $V(\mathcal{I})$. For fixed n as N grows the number of points needed per component decreases, in principle one point per component should be enough but computing costs grow quickly with n while adding new points is cheaper, so this idea is not pursued here.

Often if the degree of $V(\mathcal{I})$ is d then d points are sufficient and intersecting $V(\mathcal{I})$ by a random hyperplane gives general enough points. The result can be checked on additional random points and/or by numerical irreducible decomposition to see that in fact $V(\mathcal{I})$ is defined, at least as a set theoretical intersection, by the resulting system.

4. THE HILBERT FUNCTION

Kreuzer and Robbiano define an affine Hilbert function in [12, §5.6] which is easy to describe using our notation. We will call it the *global Hilbert function* $\mathrm{GHF}(n)$ to distinguish it from our *local Hilbert function* in [7, 6]. Given an ideal $\mathcal{I}$ in $\mathbb{C}[\mathbf{x}] = \mathbb{C}[x_1, \ldots, x_s]$

$$\mathrm{GHF}(n) = \binom{n+s}{s} - \mathrm{rank}\ \mathbf{S}(\mathcal{I}, n), \ n > 0 \tag{3}$$

Note that $\mathrm{GHF}(0) = 1$ always. Since we will be using Corollary 1 to calculate $\mathbf{S}(\mathcal{I}, n)$ a more direct calculation of $\mathrm{GHF}(n)$ is given by

Corollary 2. *If $\{\hat{\mathbf{p}}_1, \ldots, \hat{\mathbf{p}}_k\} \subseteq V(\mathcal{I})$ is a set of points for which the conclusion of Theorem 1 is true for N then for $n \leq N$*

$$\mathrm{GHF}(n) = \mathrm{rank}\ \mathrm{vjet}(\mathcal{D}_N(\{\hat{\mathbf{p}}_1, \ldots, \hat{\mathbf{p}}_k\}), n)$$

Example 2 [12, Example 5.6.2]: Consider $\mathcal{I} = \langle xy, x^2 - y\rangle \subseteq \mathbb{C}[x, y]$. Since $V(\mathcal{I}) = \{(0, 0)\}$ it can be seen that Theorem 1 holds for $\{(0, 0)\}$. Thus the global dual space is the local dual space spanned by $\partial_1, \partial_x, \partial_y + \partial_x^2$. In [7, 6] this gives local Hilbert function $1, 1, 1, 0, 0, \ldots$. The global duals are $1, X, Y + X^2$ so $\mathrm{vjet}(\mathcal{D}_N(\{(0, 0)\}, 2)$ is the matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}^{\top}$$

for $N \geq 2$ where the rows are indexed by $1, X, Y, X^2, XY, Y^2, \ldots$. So $\mathrm{vjet}(\mathcal{D}_N(\{(0, 0)\}, n)$ has rank 3 for $n \geq 1$ and hence GHF is $1, 3, 3, 3, \ldots$.

Further it is noted in [12] that $\mathbb{C}[x, y]/\mathcal{I} \cong \mathbb{C}[x]/\langle x^3\rangle$ which has GHF $1, 2, 3, 3, \ldots$. Thus for ideals $\mathcal{J}$ in $\mathbb{C}[\mathbf{x}]$ the global Hilbert function is an invariant of $\mathcal{J}$ but not of the ring $\mathbb{C}[\mathbf{x}]/\mathcal{J}$.

We remark that there is a polynomial $\mathrm{GHP}(n)$ with rational coefficients which agrees with GHF for large n known as the *affine Hilbert polynomial.* The degree d of this polynomial is the Krull dimension of $\mathbb{C}[x]/\mathcal{I}$, see again [12] for details. If c_d is the leading coefficient of GHP we will call the integer $d!\, c_d$ the *degree* of $V(I)$ which agrees with the usual definition of degree of an irreducible variety.

5. THE ALGORITHMS

5.1 Calculating $\mathcal{D}_N(\hat{\mathbf{p}}_1, \ldots, \hat{\mathbf{p}}_k)$

Let polynomials $f_1, \ldots, f_t$ be given, as in §3 we will let $\mathcal{D}_N = \mathcal{D}_N(\hat{\mathbf{p}}_1, \ldots, \hat{\mathbf{p}}_k)$ be the column matrix of all $\mathrm{vjet}(\Gamma(\mathbf{d}))$ where $\mathbf{d}$ is of order N or less in the dual of the ideal $\mathcal{I} = \langle f_1, \ldots, f_t\rangle$ at some $\hat{\mathbf{p}}_j$. $\mathcal{D}_N$ can be calculated by Macaulay matrices as in Lemma 2 and §3.2 or using strategies such as in [13, 16, 21, 22] and then multiplying the vjet vectors at $\hat{\mathbf{x}}$ by $\gamma_{\hat{\mathbf{x}}}$.

Computations in this paper are done primarily by SVD based numerical linear algebra. Matrix ranks are given by the number of singular values larger than a specified tolerance. In satisfying the conclusion of Theorem 1 picking a correct tolerance is a further complicating factor.

Experiment 3: The most difficult example encountered is finding the positive dimensional locus of the well known cyclic-4 example:

$$\begin{gathered} x_1 + x_2 + x_3 + x_4 \\ x_1x_2 + x_2x_3 + x_3x_4 + x_4x_1 \\ x_1x_2x_3 + x_2x_3x_4 + x_3x_4x_1 + x_4x_1x_2 \\ x_1x_2x_3x_4 - 1 \end{gathered}$$

Let $\mathcal{I}$ be the ideal generated by the above, then $V(\mathcal{I})$ contains a curve of degree 4 with global Hilbert function $1, 3, 6, 10, 14, 18, 22, \ldots$ and the remaining points are isolated. It is relatively easy to find points on the curve by adding a random linear function to the system above as the augmented system will have 4 solution points. This latter information implies the highest degree term of the global Hilbert polynomial will be $4t$. As mentioned above it is tempting to use these 4 points and $N = 6$ to satisfy the conclusion of Theorem 1.

One can test by calculating a global Hilbert function using Corollary 2. As an experiment we tested the terms of the global Hilbert function up to degree 6. Ten trials each were done using MATHEMATICA 6 and MATHEMATICA 7, 3 different tolerances and both 4 and 5 points. For each trial with 4 points the points were chosen by intersecting the Cyclic 4 with a random hyperplane. For the trials with 5 points 4 were chosen as before and the fifth came from a different random hyperplane intersection. For each of the 12 possibilities 10 trials were run. The following table gives the number of successful trials out of ten.

MATHEMATICA	6.0.3.0		7.0.1	
tolerance	4 pts	5 pts	4 pts	5 pts
10^{-12}	5	9	3	6
10^{-10}	9	9	5	6
10^{-8}	7	7	10	8

Overall it seems that with tight tolerance the extra point helped, but with loose tolerance the extra point was not needed. It should be pointed out that often, even if one or more terms in the Hilbert function was incorrect, the correct leading term of the Hilbert Polynomial, $4t$, could still be inferred from the last 3 terms. This happened 107 times out of the 120 total trials or 89% of the time.

When the correct Hilbert function is obtained then using the same tolerance and the two algorithms HBasis, MBasis below give a complex H-basis. Comparing Sylvester matrices we see that this complex H-basis is equivalent to the known H-basis $\{x_1 + x_3,\ x_2 + x_4,\ -1 + x_1^2x_2^2\}$. With the correct number of points and tolerance this entire calculation should take less than a minute of computer time.

Thus given the present lack of a definitive criteria for satisfing Theorem 1 with a given tolerance we must regard this step, only, as heuristic. If the Hilbert function is not known a-priori then one should run several trials with different random points, different numbers of points and different tolerances. One is looking for the smallest global Hilbert function with the correct highest degree term, easily found, for the Hilbert polynomial.

5.2 Computing $\mathbf{S}(\mathcal{I}, k)$

Our main algorithm is the following, here $\mathcal{X}_n$ is a vector of column indices of $\mathbf{S}(\mathcal{I}, n)$.

Algorithm HBasis: Calculation of Sylvester Matrices.

- **Input:** Positive integers $n \le N$, polynomials $f_1, \ldots, f_t$ defining $V(\mathcal{I})$ and points $\hat{\mathbf{p}}_1, \ldots, \hat{\mathbf{p}}_j$ in $V(\mathcal{I})$.
- Calculate $\mathcal{D}_N = \mathcal{D}_N(\hat{\mathbf{p}}_1, \ldots, \hat{\mathbf{p}}_k)$ as in §5.1.
- Truncate this matrix to order n, i.e. take the first $\binom{n+s}{s}$ rows vjet$(\mathcal{D}_N, n)$ of $\mathcal{D}_N$.
- Find the nullspace of vjet$(\mathcal{D}_N, n)$ as a row matrix S.
- **Output:** The matrix S or the set of polynomials $S\mathcal{X}_n$.

If $\mathcal{I}$ and the points $\hat{\mathbf{p}}_1, \ldots, \hat{\mathbf{p}}_k$ satisfy Theorem 1 for order N then $S = \mathbf{S}(\mathcal{I}, n)$, and, if N is sufficiently large, $\mathbf{S}(\mathcal{I}, N)$ will be a H-basis for $\mathcal{I}$.

Even when $\mathcal{D}_N(\hat{\mathbf{p}}_1, \ldots, \hat{\mathbf{p}}_k)$ does not have the correct Hilbert function the algorithm HBasis may give useful information.

Example 4: Consider the polynomial

$$\begin{aligned} f = 2 + 3x^2 - 5x^4 + x^6 - 3y^2 - 16x^2y^2 + 8x^4y^2 - 5y^4 + \\ 16x^2y^4 + 7y^6 + 9xz - 4x^3z - 2x^5z - 22xy^2z + \\ 11xy^4z + 7x^2z^2 - 4x^4z^2 - 11x^2y^2z^2 + x^3z^3 \end{aligned}$$

This polynomial can be factored exactly over a complicated complex finite extension field of the rationals to give 3 real quadratic factors. However we merely have MATHEMATICA pick random complex values, approximately, $\hat{y} = -0.59133 - 0.126784\imath$, $\hat{x} = -0.2477 + 0.897805\imath$ which we substitute into f and solve the resulting one variable polynomial for $\hat{z}$. For $\hat{\mathbf{p}}_1 = (\hat{x}, \hat{y}, 1.03993 + 5.73923\imath)$, approximating the 17 digit machine numbers, we construct $\mathcal{D}_7(\{\hat{\mathbf{p}}_1\})$ taking 4.3 seconds. The Hilbert function with tolerance 10^{-9} is $1, 4, 9, 16, 24, 30, 34, 36$ which shows that Theorem 1 is not satisfied for $\{\hat{\mathbf{p}}_1\}$ and $N = 7$ but that the Sylvester matrix of order 2 obtained by the algorithm HBasis will contain the unique row vjet$(g_1, 2)$ for a quadratic g_1. The time reported for this computation is zero since we are merely taking the nullspace of a tiny 36×10 matrix. Thus from this single point $\hat{\mathbf{p}}_1$ we obtain a potential factor of f in under 5 seconds. We repeat this for points $\hat{\mathbf{p}}_2, \hat{\mathbf{p}}_3$ with the same $\hat{x}, \hat{y}$ but $\hat{z} = 0.232135 - 0.184349\imath,\ 0.395344 + 1.01247\imath$ respectively. Thus we obtain, after about 15 seconds, the following 3 factors with real coefficients, the small imaginary terms dropped, and coefficients accurate to 9 digits

$$\begin{aligned} g1 =& -0.447561736 + 0.364721029x^2 \\ & + 0.812282765y^2 - 0.082840707xz \\ g2 =& -0.210422444 - 0.537639032x^2 \\ & - 0.327216587y^2 - 0.748061476xz \\ g3 =& 0.561390307 - 0.134811931x^2 \\ & - 0.696202238y^2 + 0.426578376xz \end{aligned}$$

Multiplying these as reported above, then norming the result to have constant term 2, we obtain a polynomial whose difference from the exact f is about $5 * 10^{-8}$ in the 2-norm.

5.3 Minimal H-Basis

Let $\mathcal{X}_n$ be the vector of column indices of $\mathbf{S}(\mathcal{I}, n)$ for an ideal of $\mathbb{C}[x_1, \ldots, x_s]$. In theory it should hold that

$$\text{rowspace}\big(\mathbf{S}(\mathbf{S}(\mathcal{I}, n)\mathcal{X}_n, n+1)\big) \subseteq \text{rowspace}\big(\mathbf{S}(\mathcal{I}, n+1)\big) \quad (4)$$

From the point of view of numerical linear algebra this inclusion of row spaces could be tested by

$$\text{rank } \mathbf{S}(\mathcal{I}, n+1) = \text{rank } \begin{bmatrix} \mathbf{S}(\mathbf{S}(\mathcal{I}, n)\mathcal{X}_n, n+1) \\ \mathbf{S}(\mathcal{I}, n+1) \end{bmatrix} \quad (5)$$

where the second matrix is a block matrix and rank is measured by number of singular values above the current working tolerance. However using the algorithm HBasis of the last section (4) may not precisely hold. There are various strategies that may be used to overcome this issue, but in this paper we will use the simple, although not totally satisfying, strategy of loosening the working tolerance so that (5) holds. Since (5) almost holds the extra singular values of the test block matrix are still small so we can reset the tolerance to be slightly larger than the largest of these.

Given a matrix A with rowspace a subset of the rowspace of B there are many numerical algorithms which, to a tolerance ϵ, will produce a submatrix B' of B such that all of the rows of B' are numerically independent of the rowspace of A and the block matrix $\left[\begin{smallmatrix} A \\ B' \end{smallmatrix}\right]$ has the same rowspace the same rowspace as B. We let the reader choose her favorite and call the result B' a *complimentary matrix to A in B*.

Algorithm MBasis: Calculation of Minimal H-basis.

```
• Input: $\mathcal{D}_N(\hat{\mathbf{p}}_1, \ldots, \hat{\mathbf{p}}_k)$ as calculated in §5.1.
• Use HBasis algorithm to successively calculate $\mathbf{S}(\mathcal{I}, 1)$,
  $\mathbf{S}(\mathcal{I}, 2), \ldots$ until one finds a non-empty Sylvester matrix
  $\mathbf{S}(\mathcal{I}, n_0)$, initialize $\mathcal{B} = \mathbf{S}(\mathcal{I}, n_0)\mathcal{X}_{n_0}$
• For $n = n_0, \ldots, N-1$ do
    – Calculate $\mathbf{S}(\mathcal{I}, n+1)$ and check that (5) holds for
      $n$, if not increase $\epsilon$ so that it does hold. If the new
      $\epsilon$ is unacceptably large then quit, algorithm fails.
    – Let $C$ be the complimentary matrix to
      $\mathbf{S}(\mathbf{S}(\mathcal{I}, n)\mathcal{X}_n)$ in $\mathbf{S}(\mathcal{I}, n+1)$. Append $C\mathcal{X}_{n+1}$ to
      $\mathcal{B}$.
• Output $\mathcal{B}$.
```

Example 5: Consider the plane curve defined by $f = 8x^3+x^4+12x^2y-20xy^2-x^2y^2+4y^3+y^4$ which clearly has a singularity at the origin. We wish to desingularize this curve and use the information to find the tangent directions of the branches, which are irrational numbers. Since apriori we do not know the answer we use a random quadratic, approximately $g = -.292846x + .999554y - .763056xz - .063694yz$. At least locally the desingularized curve will be the component of the system f, g obtained by removing the unwanted multiple component $x = y = 0$. To illustrate algorithm MBasis we will find a minimal H-basis describing this component.

Intersecting $V(\langle f, g, \rangle)$ with a random hyperplane we obtain 8 points, three on the component $x = y = 0$. We calculate $\mathcal{D}_6(\{\hat{\mathbf{p}}_1, \ldots, \hat{\mathbf{p}}_5\})$, starting with tolerance 10^{-12}, which takes 6 seconds. The Hilbert function confirms we have a curve of degree 5, so Theorem 1 is satisfied. By HBasis $\mathbf{S}(\mathcal{I}, 1)$ is empty, where $\mathcal{I}$ will be the ideal of the desingularization. $\mathbf{S}(\mathcal{I}, 2)$ has rank 1, essentially the vjet of quadratic g, so we initialize $B = \{g\}$. Now $\mathbf{S}(B, 3)$ has rank 4, and HBasis finds $\mathbf{S}(\mathcal{I}, 3)$ also to be of rank 4 with condition (5) satisfied, so $\mathcal{B}$ will contain nothing of degree 3. $\mathbf{S}(B, 4)$ has rank 10 while $\mathbf{S}(\mathcal{I}, 4)$ has rank 14. Condition (5) is not satisfied so we raise tolerance to $6*10^{-10}$ to make it true. We add the 4 new functions to $\mathcal{B}$. There is nothing new in $\mathbf{S}(\mathcal{I}, 5)$ or $\mathbf{S}(\mathcal{I}, 6)$ but we must raise the final tolerance to $3*10^{-6}$ to keep (5) valid. Along the way we have calculated the Hilbert function of the ideal generated by $\mathcal{B}$ using (3) and we appear to have our curve of degree 5 so we are satisfied that we have the correct $\mathcal{B}$.

Substituting $x = y = 0$ into the 5 members of $\mathcal{B}$ we find that all 5 have solutions $\{-1.7727, 0.479915, 0.810186\}$ for z so we now know the points over $(0, 0)$ in the curve $V(\mathcal{B})$, we can check that these points are all non-singular by checking the Jacobians, each is of rank 2. Projecting the null vectors of these Jacobians down to the plane we find the tangent lines to be $y + .391382x,\ y - 1.22713x,\ y - 4.16425x$.

5.4 Alternate Minimal Basis Computation

The algorithm MBasis above returns a numeric minimal H-basis. Not only do these bases contain polynomials with many terms but the coefficients are floating point complex numbers which makes these bases awkward to display. In this section we consider an alternate computation of a minimal H-basis which often gives nicer results to display. When there is an exact minimal H-basis this algorithm often finds a good approximation of it. This algorithm is, however, less robust and may not work for large systems.

A matrix M is in RRREF form, i.e. *reverse reduced row echelon form*, if the rightmost non-zero entry of each row is a 1, say in the i_0^{th} row and j_0^{th} column and each entry in the i^{th} row and j^{th} column for $i < i_0,\ j \geq j_0$ is zero. In s variables the number of monomials of total degree n or less is $\binom{n+s}{n}$. So if F is set of s-variable polynomials and $S(F, n)$ is put in RRREF form the rows with leading (rightmost) 1 in position $\binom{k+s-1}{k-1} + 1$ to $\binom{k+s}{k}$ correspond to polynomials with total degree k. By a mild abuse of language we will say the *degree* of that row is k and write $RS(F, n)$ for the RRREF reduction of $S(F, n)$ and $RS(F, n)_k$ for the submatrix of $RS(F, n)$ consisting of rows of degree k or less. The following specialization of Lemma 1 is then easily proved:

Lemma 4. *Let $f \in \mathcal{I}$ be of total degree $k \leq n$. Then* vjet(f, k) *is in the rowspace of $RS(\mathcal{I}, n)_k$.*

In [4] the author introduced a *approximate reverse row echelon form* algorithm ARRREF which creates an approximate reverse row echelon form. The approximate method is similar to an approximate RREF introduced about the same time in [17]. The author's implementation also outputs a list b of the pivot columns and we will assume below that the ARRREF algorithm used does this.

Then we have the alternate minimal H-basis algorithm:

Algorithm MBasis2: Minimal S-basis via the ARRREF algorithm

```
• Input: Sylvester Matrix $\mathbf{S}(\mathcal{I}, n)$ for large $n$ so that
  $\mathbf{S}(\mathcal{I}, n)$ contains an H-basis.
• Apply ARRREF to $\mathbf{S}(\mathcal{I}, n)$ to obtain a matrix $RS(\mathcal{I}, n)$
  which is row equivalent to $\mathbf{S}(\mathcal{I}, n)$ but in reverse re-
  duced row echelon form. Analyze the list of pivot
  indices to obtain a list $k_1 \leq k_2 \leq \cdots \leq k_n$ where
  $k_i = \dim RS(\mathcal{I}, n)_i$.
```

- Let i be the smallest index where $k_i > 0$. Multiply $RS(\mathcal{I}, n)_i$ by the column vector of indices $\mathbf{x}^{\mathbf{j}}, |\mathbf{j}| \leq k_i$ to obtain the corresponding polynomials and let $\mathcal{B}$ be this set of polynomials.
- Let $j = i + 1$, while $j \leq n$ do
  - *a.* Calculate $\mathbf{S}(\mathcal{B}, j)$, if rank $\mathbf{S}(\mathcal{B}, j) = k_j$ increment $j = j + 1$ and repeat step *a*.
  - *b.* Find the complementary matrix of $\mathbf{S}(\mathcal{B}, j)$ and the first $\binom{s+j}{s}$ columns of $RS(\mathcal{I}, n)_j$. Extract the last $k_j -$ rank $\mathbf{S}(\mathcal{B}, j)$ rows of this complementary matrix and multiply by the appropriate vector of indices $\mathbf{x}^{\mathbf{j}}$ to get polynomials. Append these polynomials to $\mathcal{B}$. Increment $j = j + 1$ and go back to step *a*.
- **Output** $\mathcal{B}$.

6. SUBVARIETIES AND UNIONS OF VARIETIES

In both Examples 4 and 5 we illustrated the application that motivated this paper, finding a system of equations defining a subvariety of a reducible variety. In Example 4 we were fortunate that one point was enough, in Example 5 we were fortunate that many points on the desired component satisfied a simple condition, $x \neq 0$ or $y \neq 0$. Another fortuitous situation is finding the top dimension locus of $V(\mathcal{I})$ using MATHEMATICA's NSolve which, when presented with a non-zero dimensional system returns a set of points in the top dimensional locus of the variety. We illustrated this with Example 3.

More generally one could use homotopy continuation numerical algebraic geometry software such as PHCPACK [20] or BERTINI [1] the latter of which automatically gives degree $V(\mathcal{J})$ *witness points* on the component $V(\mathcal{J})$. If this is not enough BERTINI also has an automatic option to give more. PHCPACK can also give this information but requires a bit more effort from the user.

An alternative is to note that each point on a positive dimensional component of $V(\mathcal{I})$ has infinitely many local dual differentials. These can be interpreted as tangency conditions. Thus while a function that that is dual to just a few low order differentials at one point $\hat{\mathbf{p}}$ need not be in the ideal of the component it should have a small residue near $\hat{\mathbf{p}}$.

Conversely, Suppose $\mathcal{I}_1, \mathcal{I}_2$ are ideals in $\mathbb{C}[\mathbf{x}]$. If $\mathcal{D}^1 = \mathcal{D}^1_N(\hat{\mathbf{p}}_1, \ldots, \hat{\mathbf{p}}_j), \mathcal{D}^2 = \mathcal{D}^2_N(\hat{\mathbf{q}}_1, \ldots, \hat{\mathbf{q}}_k)$ are the dual matrices for the two ideals both satisfying Theorem 1 for the same N then any $f \in \mathcal{J} = \mathcal{I}_1 \bigcap \mathcal{I}_2$ will be dual to both, that is dual to the block matrix $[\mathcal{D}^1|\mathcal{D}^2]$, and conversely. Thus the algorithms HBasis, MBasis will calculate a minimal S-basis for $V(\mathcal{I}_1) \bigcup V(\mathcal{I}_2)$. Note that [5] gives a different approach that works for homogeneous ideals only.

Here is one final example illustrating both of the above ideas.

Example 6 Let $F = [x^3 + y^3 + z^3 - 1, x^2y + xy^2 + xz + yz]$. It is checked that $V(\langle F \rangle)$ is the union of 9 lines in the non-singular cubic surface $x^3 + y^3 + z^3 = 1$. For more information on this see [5, 11]. We will first use MBASIS2 to find equations for 4 of these lines and then find the ideal of the union of these 4 lines.

Intersecting $V(\langle F \rangle)$ with a random plane gives 9 points, 4 of which are, approximately

$$\begin{aligned}
\hat{\mathbf{p}}_1 =& (-0.5 + 0.866025\imath, 0.422331 + 0.471534\imath,\\
& 0.619526 - 0.129982\imath)\\
\hat{\mathbf{p}}_2 =& (-0.5 - 0.866025\imath, 0.422331 - 0.471534\imath,\\
& 0.619526 + 0.129982\imath)\\
\hat{\mathbf{p}}_3 =& (0.311069 + 0.406982\imath, -0.5 + 0.866025\imath,\\
& 0.507991 - 0.0659029\imath)\\
\hat{\mathbf{p}}_4 =& (0.311069 - 0.406982\imath, -0.5 - 0.866025\imath,\\
& 0.507991 + 0.0659029\imath)
\end{aligned}$$

For each of these points we calculate $\mathcal{D}_3(\hat{\mathbf{p}}_j)$ as in §5.1 and then use HBasis to find $\mathbf{S}(\mathcal{J}_j, 1)$ where $\mathcal{J}_j$ is the ideal of the line through $\hat{\mathbf{p}}_j$. Then apply MBasis2 to get a nice system. The total MATHEMATICA 6 time for all three steps is 0.05 seconds. The lines are given by the following exact systems which differ from the numeric output by about $3 * 10^{-16}$ for each equation. Let λ be cube root of unity $(-1 + \sqrt{3}\imath)/2$.

$$\begin{aligned}
\ell_1 &= [-\lambda + x, \lambda y + z]\\
\ell_2 &= [-\lambda^{-1} + x, \lambda^{-1} y + z]\\
\ell_3 &= [-\lambda + y, \lambda x + z]\\
\ell_4 &= [-\lambda^{-1} + y, \lambda^{-1} x + z]
\end{aligned}$$

It is seen that the configuration of these for lines is a 2×2 i.e. each line intersects two of the others and is skew from the third, see [5], which is one of the few subconfigurations of lines in the cubic which is known to be a complete intersection. We will find a minimal H-basis and verify this last fact.

By concatenating the 20×4 matrix $\mathcal{D}_3(\hat{\mathbf{p}}_1)$ with $\mathcal{D}_3(\hat{\mathbf{p}}_2)$ we get the 20×8 matrix $\mathcal{D}_3(\hat{\mathbf{p}}_1, \hat{\mathbf{p}}_2)$. Applying HBasis to get $\mathbf{S}(\mathcal{I}_{12}, 3)$ and then HBasis we get a minimal H-basis for $\mathcal{I}_{12}$ the ideal of the union $\ell_1 \cup \ell_2$. This numerical minimal H-basis is again very close to the exact

$$\mathcal{I}_{12} = \langle 1 + x + x^2, xy + z, -y + z + xz, y^2 - yz + z^2 \rangle$$

which is not a complete intersection.

Likewise

$$\begin{aligned}
\mathcal{I}_{123} =& \langle xy + z, \lambda + \lambda x + \lambda x^2 - y + z + xz,\\
& -\lambda^{-1} - \lambda^{-1}x - \lambda^{-1}x^2 + y^2 - yz + z^2 \rangle\\
\mathcal{I} =& \langle xy + z, 1 + x + x^2 + y + y^2 - z - xz - yz + z^2 \rangle.
\end{aligned}$$

where $\mathcal{I}_{123}$ is the ideal of $\ell_1 \cup \ell_2 \cup \ell_3$ and $\mathcal{I}$ is the ideal of the union of all 4 lines. Note that $\mathcal{I}$ is given by two generators as claimed.

7. CONCLUSION

By carefully distinguishing between local and global duals and exploiting the connection between the two we can work with positive dimensional ideals. Although further analysis and experience working with these heuristic methods is necessary one can be optimistic that these methods will be useful working with algebraic sets given numerically and accurately.

Acknowledgements

The author thanks an anonymous referee for pointing out the crucial reference [15].

Access to MATHEMATICA was provided by Oakton Community College.

8. REFERENCES

[1] D. J. Bates, J. D. Hauenstein, A. J Sommese and C. W. Wampler II, BERTINI*: Software for numerical algebraic geometry*, available at www.nd.edu/∼sommese/bertini, Version 1.2, Nov. 2009.

[2] D. J. BATES, J. D. HAUENSTEIN,C. PETERSON AND A. J. SOMMESE *Numerical Decomposition of the Rank-Deficiency Set of a Matrix of Multivariate Polynomials*, in L. Robbiano, J. Abbott (eds.) *Approximate Commutative Algebra*, Texts Monogr. Symbol. Comput., Springer-Verlag, 2009.

[3] D. J. BATES, J. D. HAUENSTEIN,T. M. MCCOY, C. PETERSON AND A. J. SOMMESE *Recovering exactness from numerical data in algebraic geometry*, preprint.

[4] B. H. DAYTON, *Numerical Local Rings and Local Solution of Nonlinear Systems*, Proceedings of SNC '07, ACM Press, pp. 79–86, 2007.

[5] B. H. DAYTON, *Ideals of Numeric Realizations of Configurations of Lines*, in *Interactions of Classical and Numerical Algebraic Geometry*, D. J. Bates, G. Besana, S. Di Rocco and C. W. Wampler Editors, Contemp. Math. 496, AMS, 2009.

[6] B. H. DAYTON, T. Y. LI AND Z. ZENG, *Multiple zeros of nonlinear systems*, to appear in Mathematics of Computation. Posted on Math. Comp. website `www.ams.org/journals/mcom` February 2011.

[7] B. H. DAYTON AND Z. ZENG,*Computing the Multiplicity Structure in Solving Polynomial systems.*, Proc. of ISSAC '05, ACM Press, pp. 116–123.

[8] J. EMSALEM, *Géométrie des points éspais*, Bull. Soc. Math. France, 106 (1978), pp. 399–416.

[9] G.-M. GREUEL AND G. PFISTER, *A Singular Introduction to Commutative Algebra*. Springer Verlag, 2002.

[10] W. GRÖBNER, *Algebrishe Geometrie II*, vol. 737 of Bib. Inst. Mannheim, Hochschultaschenbücher, 1970.

[11] D. HILBERT AND S. COHN-VOSSEN, *Geometry and the Imagination*, translated from the 1932 *Anschauliche Geometrie* by P. Nemenyi 1952. AMS Chelsea Publishing, 1999.

[12] M. KREUZER AND L. ROBBIANO, *Computational Commutative Algebra 2*, Springer, 2005.

[13] N. LI AND L. ZHI *Computing the multiplicity structure of an isolated singular solution: case of breadth one*, to appear in Journal of Symbolic Computation, 2010.

[14] F. S. MACAULAY, *The Algebraic Theory of Modular Systems*, Cambridge Univ. Press, 1916.

[15] H.M. MÖLLER AND T. SAUER, *H-bases for polynomial interpolation and system solving*, Adv. in Comp. Math. 12 (2000) pp. 335–362.

[16] B. MOURRAIN, *Isolated points, duality and residues*, J. of Pure and Applied Algebra, 117-118 (1996), pp. 469-493.

[17] R. SCOTT, *Approximate Gröbner bases – a backwards approach*. Master of Science Thesis, Greg Reid supervisor, Graduate Program in Applied Mathematics, University of Western Ontario, London, Ontario, Canada, 2006.

[18] A. J. SOMMESE, J. VERSCHELDE AND C. W. WAMPLER, *Numerical decomposition of the solution sets of polynomial systems into irreducible components*, SIAM Journal on Numerical Analysis 38 (2001), pp. 2022-2046.

[19] H. J. STETTER, *Numerical Polynomial Algebra*. SIAM, 2004.

[20] J. VERSCHELDE, *Algorithm 795: PHCpack: a general-purpose solver for polynomial systems by homotopy continuation*, ACM Transactions on Mathematical Software 25 (1999), pp. 251-276. Available at `www.math.uic.edu/∼jan`.

[21] X. WU AND L. ZHI, *Computing the multiplicity structure from geometric involutive form.* Proc of ISSAC 2008, ACM Press, pp. 325–332.

[22] Z. ZENG, *The closedness subspace method for computing the multiplicity structure of a polynomial system*, in *Interactions of Classical and Numerical Algebraic Geometry*, D. J. Bates, G. Besana, S. Di Rocco and C. W. Wampler Editors, Contemp. Math. 496, AMS, 2009, pp. 347–362.

Interval Function and Its Linear Least-Squares Approximation

Chenyi Hu
University of Central Arkansas
Computer Science Department
Conway, AR 72035
chu@uca.edu

ABSTRACT

This paper reports an interval least-squares (ILS) algorithm that computationally approximates an interval function, in which both dependent and independent variables are interval valued. An initial version of this algorithm and its implementation were developed as a computational scheme for the stock market variability forecast with significantly improved results reported in [16] and [18]. The computational scheme then applied to predict mortgage rate [12], crude oil price [31], and network bandwidth [23]. In all of these applications, the interval least squares approach improved computational results significantly in terms of much higher accuracy and stability comparing against that of traditional confidence interval forecasts with point-valued ordinary least squares.

In this paper, we modify the scheme as a general purpose ILS algorithm for interval function approximation with clearly defined concepts and quantitative quality assessment. A summary of its application on a real world multi-variable time series is also included.

Categories and Subject Descriptors

G.1.0 [**Numerical Analysis**]: General–Interval arithmetic; G.1.2 [**Approximation**]: Least Squares Approximation

General Terms

Algorithms, Theory

Keywords

Interval function, interval least squares, error and accuracy of interval estimation, uncertainty, variability forecast.

1. INTRODUCTION

1.1 Interval function and the problem we address

In this paper, we use boldfaced letters to specify an interval or an interval vector. We use an under- and an over- lines to denote the lower and upper bounds of an interval, respectively. For example, $\boldsymbol{x}$ is an interval (or interval vector), and its lower and upper bounds are $\underline{x}$ and $\overline{x}$, respectively.

We repeat the definition of an *interval function* in [14]:

Definition 1: Let X and Y be the sets of intervals $X = \{\boldsymbol{x} : \boldsymbol{x} \subset \mathbb{R}^n, \boldsymbol{x} \text{ is an interval vector}\}$ and $Y = \{\boldsymbol{y} : \boldsymbol{y} \subset \mathbb{R}, \boldsymbol{y} \text{ is an interval}\}$. If for any $\boldsymbol{x} \in X$ there is a unique interval $\boldsymbol{y} \in Y$ correspondent to $\boldsymbol{x}$, then we say that X and Y define an interval function $\boldsymbol{y} = f(\boldsymbol{x})$.

In applications, the analytic form of an interval function $\boldsymbol{y} = f(\boldsymbol{x})$ may not be provided explicitly. Instead, one may only be able to work on a set of interval-valued observations $(\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{x} \subset \mathbb{R}^n$ and $\boldsymbol{y} \subset \mathbb{R}$.

The problem we address in this paper is to find an interval function that best fits the observations in a least squares sense.

1.2 Motivation of this study

The motivation of this study comes from the following observations mainly.

First, data observed and recorded in finite precision are often imprecise. Using intervals to contain uncertainty as well as observational and rounding errors, one may probe the reliability of computational results of applications including function approximation. In 1998, Hu *et al* reported their initial investigation on interval polynomial interpolation and its Lagrange solution of univariate functions [14]. Their numerical examples evident that even with a very small percentage of error, from $\mp 0.1\%$ to $\mp 0.5\%$, for only five or six point valued nodes, the interval Lagrange interpolation could vary widely. This implies unreliability and instability of computational results.

Second, in studying variability of natural and/or social phenomena within a specified time period, the typical objective is not to project the instantaneous value at a particular time spot but rather to estimate the range of the function within the time period. For example, the annual precipitation in a city during the decade 2010-2019 would be predicted more reasonably as an interval rather than a point. Traditionally, people have applied statistics and probability theory to generate confidence interval predictions. However, the confidence interval approach does not work well in many cases. For example, in forecasting the stock market annual variation from 1940-2004, the 95% confidence interval forecasts reaches the average accuracy ratio[1] only about 13%.

[1]See Definitions 5 and 6 in section 3.5 of this paper.

But, the interval function approach with the same dataset the same economical model reaches the average accuracy above 64% and achieves much higher stability in terms of less standard deviation [11], [16], and [18]. In [15], Hladik and Cerny also discussed finding interval regression parameters.

Finally, many real-world phenomena, even at the atomic level, are better described as intervals. The International Union of Pure and Applied Chemistry has very recently adopted as a standard that atomic mass, one of the most fundamental scientific concepts, as an interval rather than as a point. For instance, the atomic mass of chlorine is now [35.446, 35.457], not 35 (or 35.453) .

Again, in many real world applications, the explicit form of an interval function is unavailable. The best one can do is probably to find a reasonable estimation from a collection of interval-valued pairs $(\boldsymbol{x}, \boldsymbol{y})$. The linear least squares principle has been arguably most broadly applied in function approximation. Therefore, we study interval least squares approximation in this paper.

1.3 Organization of the rest of this paper

The rest of this paper is organized as the follows. We define the interval least squares problem in section 2 and present a practical algorithm with quantitative quality assessment in section 3. We report a successful real-world applications as a sample in section 4. We conclude the paper with possible future studies in section 5.

2. INTERVAL LEAST SQUARES (ILS)

2.1 The linear ordinary least squares

The interval least squares (ILS) is closely related to traditional linear ordinary least squares (OLS). Therefore, we briefly review the point-valued OLS first. A more detailed summary can be found in [24].

With a selected set of base functions $\Phi = \{\varphi_0, \varphi_1, \ldots, \varphi_m\}$ and N data points (x_i, y_i), where $i \in \{1, 2, \ldots, N\}$, the objective of linear OLS is to find a vector $\alpha = (\alpha_0, \alpha_1, \ldots, \alpha_m)$ that minimizes

$$\sum_{i=1}^{N} \left(y_i - \sum_{0 \le j \le m} \alpha_j \varphi_j(x_i) \right)^2 .$$

To computationally determine the coefficient vector α, one needs to evaluate the basis functions $\varphi_j(x)$ at x_i for all $1 \le i \le N$ and $1 \le j \le m$ and then to form the design matrix

$$X = \begin{pmatrix} \varphi_0(x_1) & \varphi_1(x_1) & \cdots & \varphi_m(x_1) \\ \varphi_0(x_2) & \varphi_1(x_2) & \cdots & \varphi_m(x_2) \\ \cdots & \cdots & \cdots & \cdots \\ \varphi_0(x_i) & \varphi_1(x_i) & \cdots & \varphi_m(x_i) \\ \cdots & \cdots & \cdots & \cdots \\ \varphi_0(x_N) & \varphi_1(x_N) & \cdots & \varphi_m(x_N) \end{pmatrix} \tag{1}$$

A classical approach to obtain the α is to solve the linear system of equations, the normal equations,

$$X^T X \alpha = X^T y,$$

where $y = (y_1, y_2, \ldots, y_N)^T$.

Instead of solving the normal equations, a more recent computational approach [24] applies a sequence of Householder transformations to both sides of the system $X\alpha \approx y$ aimed to transform X to an upper triangular matrix R. One can then determine α to avoid possible ill-conditioning of the matrix $X^T X$ in the normal equations.

Using the α obtained with OLS, one may can approximate y_i with $\sum_{0 \le j \le m} \alpha_j \varphi_j(x_i)$. The average and standard deviation of the absolute error, $|y_i - \sum_{0 \le j \le m} \alpha_j \varphi_j(x_i)|$, can then be used to form confidence interval. Such that, one may have certain statistic and probabilistic confidence on that, for a provided x, the y will be within the confidence interval.

We emphasize it again, in traditional OLS confidence interval approach, point-values are used in entire calculation. On at the end of computation, a certain percentages of standard deviation (or error) are subtracted from and added to the point estimation to form intervals.

2.2 Interval least squares

Different from OLS, the concept of interval least-squares (ILS) derives an interval-valued approximation starting from interval valued input at the very beginning. In other words, an ILS interval estimates an interval function.

One of the primary objectives in previous studies on ILS, including [1], [8], and others, is to bound the norm of $\boldsymbol{X}\alpha - \boldsymbol{y}$. However, our goal in this paper is to practically minimize the overall absolute error of an approximation of an interval function.

Prior to our discussion on computational approaches to estimate the α in ILS, we need to define the meaning of interval least squares in this paper. Let us define the absolute error of interval estimation first.

Definition 2: Let interval $\boldsymbol{y}_{\text{est}} = [\underline{y}_{\text{est}}, \overline{y}_{\text{est}}]$ be an estimation of an interval $\boldsymbol{y} = [\underline{y}, \overline{y}]$. The *l*eft and *right* absolute errors are $E_L = |\underline{y}_{\text{est}} - \underline{y}|$ and $E_R = |\overline{y}_{\text{est}} - \overline{y}|$, respectively. The *absolute error* of the estimation is the sum of the left and right absolute errors, that is, $E = E_L + E_R = |\underline{y}_{\text{est}} - \underline{y}| + |\overline{y}_{\text{est}} - \overline{y}|$, respectively.

For example, if one uses $[-1.02, 1.95]$ to estimate the interval $[-1.0, 2.0]$, then the left and right absolute errors of the estimation are $E_L = |(-1.02) - (-1.0)| = 0.02$ and $E_R = |1.95 - 2.0| = 0.05$, respectively. The absolute error of the estimation is $E_L + E_R = 0.07$.

Using the concept of absolute error of interval estimation, we define linear interval least squares as the follow:

Definition 3: Let S be a set of N interval valued observations of an interval function $\boldsymbol{y} = f(\boldsymbol{x})$, i. e., $S = \{(\boldsymbol{x}, \boldsymbol{y}) : \boldsymbol{x} \subset \mathbb{R}^n, \boldsymbol{y} \subset \mathbb{R}, \text{both } \boldsymbol{x} \text{ and } \boldsymbol{y} \text{ are compact}\}$. Provided $\Phi = \{\varphi_0, \varphi_1, \ldots, \varphi_m\}$ be a set of basis functions. We say that a linear combination of the basis functions, $\sum_{0 \le j \le m} \alpha_j \varphi_j(\boldsymbol{x})$, is an interval least squares approximation of $f(\boldsymbol{x})$ if the linear combination minimizes: $\sum_{i=1}^{N} E_L^2 + \sum_{i=1}^{N} E_R^2$, where

$$\sum_{i=1}^{N} E_L^2 = \sum_{i=1}^{N} \left(\underline{y}_i - \sum_{0 \le j \le m} \underline{\alpha_j \varphi_j(\boldsymbol{x}_i)} \right)^2 , \tag{2}$$

and

$$\sum_{i=1}^{N} E_R^2 = \sum_{i=1}^{N} \left(\overline{y}_i - \sum_{0 \le j \le m} \overline{\alpha_j \varphi_j(\boldsymbol{x}_i)} \right)^2 . \tag{3}$$

2.3 Interval arithmetic

Similar to the OLS, it is required to find the vector α that minimizes the sum of (2) and (3) in solving an ILS. A significant difference between OLS and ILS is that, the ILS approximates an interval function $\boldsymbol{y} = f(\boldsymbol{x})$ but the OLS estimates a point-valued function. With interval valued data, we need to use interval arithmetic as defined in [25]:

> Let $\boldsymbol{a} = [\underline{a}, \overline{a}]$ and $\boldsymbol{b} = [\underline{b}, \overline{b}]$ be two nonempty intervals and op be an operator $(+, -, *, \div)$. Then, $\boldsymbol{a}$ op $\boldsymbol{b} = \{a \text{ op } b : a \in \boldsymbol{a}, b \in \boldsymbol{b}\}$. If op is $\div$, then $0 \notin \boldsymbol{b}$.

That is:
$\boldsymbol{a} + \boldsymbol{b} = [\underline{a} + \underline{b}, \overline{a} + \overline{b}]$,
$\boldsymbol{a} - \boldsymbol{b} = [\underline{a} - \overline{b}, \overline{a} - \underline{b}]$,
$\boldsymbol{a} * \boldsymbol{b} = [\min\{\underline{a}*\underline{b}, \underline{a}*\overline{b}, \overline{a}*\underline{b}, \overline{a}*\overline{b}\}, \max\{\underline{a}*\underline{b}, \underline{a}*\overline{b}, \overline{a}*\underline{b}, \overline{a}*\overline{b}\}]$, and
$\boldsymbol{a} \div \boldsymbol{b} = [\min\{\underline{a}/\underline{b}, \underline{a}/\overline{b}, \overline{a}/\underline{b}, \overline{a}/\overline{b}\}, \max\{\underline{a}/\underline{b}, \underline{a}/\overline{b}, \overline{a}/\underline{b}, \overline{a}/\overline{b}\}]$, provided $0 \notin \boldsymbol{b}$.

There are numerous well tested interval software tools that can perform interval arithmetic reliably and evaluate interval fundamental functions. These tools are available in mainstream programming languages such as C [21], C++ [2], [22], [27], Fortran [20], Python [19], as well as in integrated computational environment like the interval toolbox [30] in MATLAB.

3. A PRACTICAL ILS ALGORITHM

3.1 Interval valued design matrix

With preselected base functions Φ and interval data, it is straightforward to evaluate (1) and to obtain the interval valued design matrix $\boldsymbol{X}$ by using available interval software. However, it is a computational challenge to find the α such that $\boldsymbol{X}\alpha \approx \boldsymbol{y}$. Because neither directly solving the interval normal equations $\boldsymbol{X}^T\boldsymbol{X}\alpha = \boldsymbol{y}$ nor performing the QR-factorization on the interval design matrix is easy.

Reported in the §4.3 of [17], an interval linear equation $\boldsymbol{Az} = \boldsymbol{b}$ may not have an *exact solution* at all (i.e., for a given interval matrix $\boldsymbol{A}$ and an interval vector $\boldsymbol{b}$, there may not exist an interval vector $\boldsymbol{z}$ such that $\boldsymbol{Az} = \boldsymbol{b}$.) Most available interval linear solvers, if not all, try to find a $\boldsymbol{z}$ such that $\boldsymbol{Az} \supset \boldsymbol{b}$. The solution set of an interval linear system of equations is mostly irregularly shaped and nonconvex [26]. A naive application of interval linear solver to bound the solution vector α, in the interval normal equations $\boldsymbol{X}^T\boldsymbol{X}\alpha = \boldsymbol{y}$, may cause severe overestimation. Hence, the interval least squares approximation could become practically meaningless.

It is much harder to apply the QR approach on the system $\boldsymbol{X}\alpha \approx \boldsymbol{y}$. A non-trivial interval subtract itself, according to the definition of interval arithmetic, is no longer zero. Hence, to the best of our knowledge, there is not interval Householder transformation that can transform a non-zero interval vector to the form of $[\boldsymbol{a}, 0, \ldots, 0]^T$.

Moreover, even if one can find the α vector from the interval valued design matrix $\boldsymbol{X}$, there is no theory to ensure that the α so computed minimizes the sum of (2) and (3). Therefore, we try to construct a practical approach to solve an ILS computationally.

3.2 Initial solutions

Our approach begins with the interval normal equations. Traditionally, the solution set to $\mathbf{Az} = \mathbf{b}$ is defined as $\{z | \, \exists A \in \mathbf{A}, \exists b \in \mathbf{b} : Az = b\}$. Using this approach, one may obtain an interval vector $\mathbf{z}$ that is over-sized. Instead, we apply the computational method reported in the §4.3 of [17].

When an interval linear system of equations does not have an exact solution, one can find a satisfactory approximated solution to meet our needs. Applying the concept of *volume* of an interval vector $\boldsymbol{z} = (\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_n)$, which is $v(\boldsymbol{z}) = \prod_{1 \le i \le n} (\overline{z}_i - \underline{z}_i)$, the notion *ratio of estimation* for an approximated solution of interval linear systems $\boldsymbol{Az} = \boldsymbol{b}$ is defined in [17] as:

Definition 4: The ratio of estimation of an approximated solution of $\boldsymbol{Az} = \boldsymbol{b}$ is:

$$r = \begin{cases} 0\% & \text{if } \boldsymbol{Az} \cap \boldsymbol{b} = \emptyset; \\ 100\% & \text{if } \boldsymbol{Az} = \boldsymbol{b}; \\ \dfrac{v(\boldsymbol{b})}{v(\boldsymbol{Az})} & \text{if } \boldsymbol{Az} \supset \boldsymbol{b}; \\ \dfrac{v(\boldsymbol{Az})}{v(\boldsymbol{b})} & \text{if } \boldsymbol{Az} \subset \boldsymbol{b}; \\ \dfrac{v(\boldsymbol{Az} \cap \boldsymbol{b})}{v(\boldsymbol{Az} \cup \boldsymbol{b})} & \text{otherwise.} \end{cases} \tag{4}$$

The greater the ratio r is, the better an estimated solution is. Considering the interval normal equation $\boldsymbol{X}^T\boldsymbol{X}\alpha = \boldsymbol{X}^T\boldsymbol{y}$, we may try to find the α that maximizes the ratio of estimation. Although the α may not minimize the sum of (2) and (3), it would well fit the normal equation of ILS.

A computational approach is suggested in Example 4 of [17], in which an initial midpoint solution is calculated. Then an ϵ-inflation [29] is performed to enlarge the ratio of estimation. In the interval normal equation $\boldsymbol{X}^T\boldsymbol{X}\alpha = \boldsymbol{X}^T\boldsymbol{y}$, we try to find the midpoint vector of α first. This intuitively suggests matching the center of the two interval vectors $\boldsymbol{X}^T\boldsymbol{X}\alpha$ and $\boldsymbol{X}^T\boldsymbol{y}$.

Let $(X^TX)_{\text{mid}}$ and $(X^Ty)_{\text{mid}}$ be the midpoint matrix of $\boldsymbol{X}^T\boldsymbol{X}$ and the midpoint vector of $\boldsymbol{X}^T\boldsymbol{y}$, respectively. The linear system of equations $(X^TX)_{\text{mid}}\ \alpha = (X^Ty)_{\text{mid}}$ is a point (not interval) linear system. We call its point-valued solution α an *initial point solution.* Directly applying the initial point solution in $\boldsymbol{y} \approx \sum_{0 \le j \le m} \alpha_j \varphi_j(\boldsymbol{x})$, one may obtain interval valued results. Because of the independent variable $\boldsymbol{x}$ is interval-valued. The initial point solution can provide relative positions of $\boldsymbol{X}^T\boldsymbol{X}\alpha$ and $\boldsymbol{X}^T\boldsymbol{y}$ for one to calculate the ratio of estimation defined in (4).

There is an alternative way to obtain an initial point solution of α. Let X_{mid} and y_{mid} be the midpoint matrix of interval design matrix (1) and the midpoint vector of $\boldsymbol{y}$, respectively. By performing a sequence of Householder transformations on $X_{\text{mid}}\alpha = y_{\text{mid}}$ that makes X_{mid} right triangular, we can obtain an initial point solution of α to avoid possible ill-conditioned matrix $(X^TX)_{\text{mid}}$.

If we apply the sequence of Householder transformations to $X_{\text{mid}}\alpha = \boldsymbol{y}$, we get an *initial interval solution* of α. However, the initial interval solution of α can be severely overestimated because of the collapsing of the interval design matrix (1) to its mid-point without any adjustment on the interval vector $\boldsymbol{y}$.

The QR-factorization approach can produce both initial point and interval solutions. It is preferable to the normal equation approach.

3.3 Width adjustment

The initial point solution is probably too narrow because it collapse an interval valued α to a point. The initial interval solution may be too wide due to the midpoint collapsing of the interval design matrix (1). In either case, we need to adjust the width of α.

Width adjustments on initial point or interval solutions of α can be done through inflation (for the point one) or deflation(for the interval one) experimentally.

For the initial point solution, we may perform a sequence of ϵ-inflations, [17] and [29], to form various interval-valued α. For example, we may use a sequence of ϵ values of $1\%, 2\%, \ldots, 10\%$. By adding to or subtract from the point-valued α, we obtain multiple interval-valued α's. For each of them, we can calculate the ratio of estimation as defined in (4) or directly the sum of squares as in Definition 3 with provided data. Similarly, we can perform a sequence of deflations on the initial interval solution.

By comparing the sum of squares (or ratio of estimation), we can choose a preferable interval-valued α. One may also use a hybrid approach by combining both initial point and interval solutions in different weights to construct experiments.

Instead of adjusting the width of α, another approach is to approximate the interval function $\boldsymbol{y} = f(\boldsymbol{x})$ with the initial solutions of α directly in $\boldsymbol{y}_{\text{est}} \approx \sum\limits_{0 \le j \le m} \alpha_j \varphi_j(\boldsymbol{x})$ and then perform width adjustment on $\boldsymbol{y}_{\text{est}}$. We call this scheme *post adjustment of width.* In the application of the stock market interval forecasts reported in Section 4 of this paper, we have applied moving average of width within a time window to perform the post adjustment of width successfully.

3.4 A practical interval least squares algorithm

Summarizing the above discussion, we form an interval least squares algorithm for interval function approximation with the following main steps:

ALGORITHM 1. *Given a preselected set of base functions $\Phi = \{\varphi_0, \varphi_1, \ldots, \varphi_m\}$ and a set of interval valued pairs $(\boldsymbol{x}_i, \boldsymbol{y}_i)$.*

1. *Evaluate the design matrix $\boldsymbol{X}$, as defined in (1), with interval arithmetic.*
2. *Perform the QR-factorization[2] on the midpoint matrix, X_{mid}, of $\boldsymbol{X}$ such that $X_{mid} = Q * R$.*
3. *Calculate $\boldsymbol{z} = Q^T \boldsymbol{y}$.*
4. *Find an initial α with one or both of the followings:*
   - (a) *Compute an initial point solution of α by solving $R\alpha = z_{mid}$ with backward substitution.*
   - (b) *Compute an initial interval solution of α by solving $R\alpha = \boldsymbol{z}$ with interval arithmetic.*
5. *Perform width adjustments on the initial point and/or interval solutions, and select the α that achieves the least of the sums of (2) and (3) among experiments.*
6. *Apply $\boldsymbol{y}_{est} \approx \sum\limits_{0 \le j \le m} \alpha_j \varphi_j(\boldsymbol{x})$ to approximate $\boldsymbol{y} \approx f(\boldsymbol{x})$ and then perform the post adjustment of width.*

The algorithm above does provide a computational approach to experimentally find an ILS approximation of an interval function . Practically, it can probably produce meaningful computational solution for some applications as reported in the next section of this paper. However, it neither mathematically nor computationally guarantees the minimization of the sum of squares of (2) and (3).

3.5 Accuracy - a quantitative quality assessment of interval approximation

The steps 4 and 5 of Algorithm 1 suggest multiple values of α in $\boldsymbol{y}_{\text{est}} = \sum_{0 \le j \le m} \alpha_j \varphi_j(\boldsymbol{x})$. To positively assess the quality of an approximation of $\boldsymbol{y} = f(\boldsymbol{x})$, we may examine the overlap between $\boldsymbol{y}_{\text{est}}$ and $\boldsymbol{y}$. The larger the overlap between the two intervals, the better the approximation should be. By the same token, the less the non-overlap between the two intervals, the more accurate the forecast is. In addition, the accuracy of an interval estimation should be between 0% and 100%. By using the notion of interval width, which is the difference between the upper and lower bounds of an interval, we can measure the intersection and the union (or the convex hull) of the two intervals. The function $w(\ )$ returns the width of an interval. We define the concept, named the *accuracy ratio* of an interval approximation, as the follow.

Definition 5: Let $\boldsymbol{y}_{\text{est}}$ be an approximation for the interval $\boldsymbol{y}$. The accuracy ratio of the approximation is

$$acc(\boldsymbol{y}, \boldsymbol{y}_{\text{est}}) = \begin{cases} 100\% & \text{if } \boldsymbol{y} = \boldsymbol{y}_{\text{est}} \\ \dfrac{w(\boldsymbol{y} \cap \boldsymbol{y}_{\text{est}})}{w(\boldsymbol{y} \cup \boldsymbol{y}_{\text{est}})} & \text{if } (\boldsymbol{y} \cap \boldsymbol{y}_{\text{est}}) \neq \emptyset \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

For example, if we use $[-1.02, 1.95]$ to approximate the interval $[-1.0, 2.0]$, the accuracy ratio is

$$\frac{w([-1.02, 1.95] \cap [-1.0, 2.0])}{w([-1.02, 1.95] \cup [-1.0, 2.0])} = \frac{w([-1.0, 1.95])}{w([-1.02, 2.0])} = 97.68\%.$$

To quantitatively assess the quality of an interval function approximation on a set of N interval pairs $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, we may use the average accuracy ratio defined as the follow:

Definition 6: Let $\boldsymbol{y}_{\text{est}}(\boldsymbol{x}) = \sum_{0 \le j \le m} \alpha_j \varphi_j(\boldsymbol{x})$ be an approximation of an interval function $\boldsymbol{y} = f(\boldsymbol{x})$. For a provided set of N interval pairs $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, the average accuracy of the approximation is

$$acc_{\text{avg}} = \frac{\sum\limits_{i=1}^{N} acc(\boldsymbol{y}_i, \boldsymbol{y}_{\text{est}}(\boldsymbol{x}_i))}{N}. \tag{6}$$

The average accuracy ratio is a quality measurement in addition to the sum of squares of left and right errors as defined in (2) and (3). Maximizing the average accuracy ratio and minimizing the sum of squares are inter-connected. The higher the average accuracy ratio is, the smaller the sum and the better the approximation. An interval function approximation is more accurate than others in average if its mean accuracy ratio is higher and/or its mean absolute error is less than that of the rests. However, the two concepts are defined very differently.

[2]For computational efficiency, it is not necessary to calculate the Q matrix explicitly but performing the sequence of Householder transformations to accomplish both steps (3) and (4) of this algorithm.

In addition to average accuracy and total errors (in terms of least squares), we can calculate the standard deviation of the absolute errors and/or accuracy ratios on the N observations. The smaller the standard deviations of accuracy ratio and/or absolute error are, the stabler an approximation is. Use these indicators, we can practically assess the overall quality of an interval approximation quantitatively further. We report a real world application of the Algorithm 1 with quality comparisons in the following section.

4. AN APPLICATION ON TIME SERIES ANALYSIS

4.1 Time series analysis

A time series is a chronologically ordered sequence of observations of natural and social phenomenon. Time series analysis aims to understand, predict, and perhaps optimally control natural and social phenomena. The ordinary least squares algorithm has been commonly applied in the literature of time series analysis with point-valued data.

A typical approach of analyzing time series is to separate possible trends and random noise first. Then, by adding to and subtracted from the trend a factor of standard deviation, one can make predictions with some probabilistic confidence [7]. Though significant achievements have been made, scientists are very much aware of the limitations of this approach. In some cases, the confidence interval prediction can be very wide and meaningless. In some other cases, the actual event can be completely outside of predicted interval though its probabilistic confidence is considerably high. After reviewing the progress of time series forecasting during the past 25 years, Gooijer and Hyndman [9] concluded: "Clearly, the area of improved forecast intervals requires further research."

Our initial investigations on analyzing a time series with 75 years of macroeconomic data, from January 1930 to December 2004, reveal that the limitations are deeply rooted in using point data and point computing [16] and [12]. With the interval least squares approach, we are able to significantly improve the quality of forecast intervals of the stock market.

4.2 The time series and the model of forecast

The real world time series for total of 75 years of macroeconomic data is provided by a domain expert in computational finance. The dataset contains monthly observations of six macroeconomic parameters[3]:

- SP_t: the overall stock market value;
- IP_t: the growth rate variations of seasonally adjusted Industrial Production Index;
- DI_t: changes in expected inflation;
- UI_t: changes in unexpected inflation;
- DF_t default risk premiums; and
- TM_t: unexpected changes in interest rates.

[3]More detailed descriptions of these parameters can be found in [11] and [13]. The complete dataset is available at: http://sun0.cs.uca.edu/interval/projects/finance.html.

The objective is to forecast the annual variability of the overall stock market as SP_t intervals. This has both theoretical and practical importance.

In computational finance, the arbitrage pricing theory (APT) [28] provides a framework that identifies macroeconomic variables that significantly and systematically influence stock prices. By modeling the relationship between the stock market and relevant macroeconomic variables, one may forecast the overall level of the stock market. The model by Chen, Roll, and Ross [3] is broadly accepted in studying the financial market. According to the model, the changes in the overall stock market value SP_t are linearly determined by the five macroeconomic factors: IP_t, DI_t, UI_t, DF_t, and TM_t as

$$SP_t = \alpha_0 + \alpha_1 IP_t + \alpha_2 UI_t + \alpha_3 DI_t + \alpha_4 DF_t + \alpha_5 TM_t \quad (7)$$

In traditional point approach, one samples the data annually with 12-month interval, say every January (or other month, say July or December) and then fits (7) with ordinary least squares. By adding and subtracting a factor of the standard deviation, an annual confidence interval forecast will be produced.

However, computational experiments with this application suggest that the quality of confidence interval forecast is very poor. As an alternative, by using the annual minimum and maximum time series to forecast the annual lower and upper bounds of the stock market [11], respectively, some improvements have been made. But, the ILS algorithm achieves significantly improved quality of forecasts as reported in Tables 1 and 2 of this paper.

4.3 The ILS stock market interval forecasting

4.3.1 *Interval data formulation*

To perform an ILS annual interval forecast of the stock market, we need to approximate the interval function (7). The first step is to construct interval-valued input data from the monthly observed points. We take the following two approaches to form the annual interval for each of the six parameters.

The first one to use observed annual minimum and maximum within a given year as lower and upper bounds to form the annual interval for each of the six attributes. We call it min-max interval. The min-max interval has an advantage of inclusion of all monthly observations within a year, however, it does not consider randomness in the observations at all. The other approach is to form annual confidence interval for each of the six parameters. Using different confidence levels, we may form various interval-valued time series for the study and gain more information [18].

4.3.2 *Time window*

There is a general consensus in the financial literature that relationships between financial market and macroeconomic variables are time-varying [6]. This means that the relationship is valid only for a limited time period. Therefore, in applying function approximation on a time series, one should use only data inside an appropriate time window to estimate the relationship. Professor L. He, a domain expert, suggested we use a time window of ten years in our study. Further study in [4] and [5] have justified computationally that the window size of ten years is reasonable.

4.3.3 *In- and out-of- sample ILS interval forecasts*

Within the fixed time window of ten years, we perform the ILS algorithm to obtain the initial point solutionα in the equation (7). Using the α and the equation (7) with the annual interval of each parameter within the time window, we make an *in-sample forecast* of the annual SP interval for the last year in the time window. Using the α and the equation (7) with the annual interval of each parameter immediately follows the time-window, we make *out-of-sample forecast* of annual SP interval in the first year following the time window. We then apply the post width adjustment with the average width of SP intervals within the window. This is motivated by the auto regressive moving average method in time series forecast. We carry out the computation with C++ implementation of the ILS algorithm. The C++ interval BLAS package [27] is applied to perform interval computing.

4.4 Computational results and comparison

4.4.1 Computational results

By slicing the time window (also called "rolling"), reported in [6] and others, we obtain 66 in-sample and 65 out-of-sample annual ILS interval forecasts from 1939 and 1940 to 2004, respectively. For the purpose of comparison, we obtain annual interval forecasts of the SP with the following computational approaches:

- Ordinary Least Squares (OLS) 95% confidence interval forecast which is probably the most commonly applied one in the literature;
- OLS forecast of lower and upper bounds[4] with sequences of annual minimum and maximum;
- Interval Least Squares (ILS) forecast with annual min-max interval as input, and
- ILS forecast with annual 92% confidence interval (obtained with mean $\mp 1.75 *$ std.) as input.

Being able to compare the overall quality of the above computational approaches, we use the following quantitative indicators for both in-sample and out-of-sample forecasts of the stock annual variability.

- Absolute mean error:
$$E_{\text{mean}} = \frac{\sum_{i=1}^{N} E_i}{N},$$
where $E_i = E_{L_i} + E_{R_i}$ and the E_L and E_R are defined in Definition 2;
- Standard deviation of absolute forecast error:
$$\sigma_E = \sqrt{\frac{\sum_{i=1}^{N}(E_i - E_{\text{mean}})^2}{N-1}};$$
- Average accuracy ratio as defined in (6); and
- Number of forecasts with accuracy ratio 0% as defined in (5).

We summarize the statistics of these quality indicators, with each of these computational approaches in Tables 1 and 2 above, respectively.

[4]There are very few instances, in which the computed lower bound is greater than the upper bound. For them, we swap the numbers to form proper forecasting interval.

	Absolute mean err	Std of error	Avg. acc. ratio	Zero acc.
OLS 95%	0.243321	0.142537	0.245407	5
OLS low-up	0.064642	0.037364	0.469168	0
ILS min-max	0.043211	0.027064	0.686426	0
ILS 92% conf.	0.045594	0.027383	0.674752	0

Table 1: Quality comparison of the the stock market in-sample annual variability forecasts (1939-2004)

	Absolute mean err	Std of error	Avg. acc. ratio	Zero acc.
OLS 95%	0.723505	0.311973	0.125745	18
OLS low-up	0.066643	0.040998	0.461700	0
ILS min-max	0.051662	0.032238	0.641877	0
ILS 92% conf.	0.051206	0.029988	0.645526	0

Table 2: Quality comparison of the the stock market out-of-sample annual variability forecasts (1940-2004)

4.4.2 Comments on computational results

The above computational results show clear advantages of the ILS algorithm compare against traditional OLS-based approaches. The much lower mean absolute error of ILS forecasts on the stock market annual variability presents the overall better precision than that of OLS-based confidence interval forecasts. The relatively smaller standard deviation of the absolute error indicates the higher stability of the ILS forecasts than that of OLS-based calculation.

More significantly, it is reported [18] that, using confidence interval as input in the stock market annual variability least squares prediction, the forecast interval statistically preserves the same confidence level as that of the input.

5. CONCLUSION AND FUTURE WORK

Using the notion of interval functions, we can model a type of applications, in which the range of variability is more important than specific point value at a typical time spot in a specific location. In solving many real world applications, one may not have explicit analytical form of an interval function available but only a collection of interval valued observations. Therefore, approximating an interval function has practical importance.

In this paper, we defined related concepts for interval function approximation including error approximation, linear interval least squares, and accuracy ratio. A practical algorithm presented in this paper can computationally find interval function approximations with reasonable quality. The applications of the algorithm to the stock market annual variability forecasts have produced significantly improved accuracy and stability. We have also observed similar quality improvements in the forecasts of mortgage rates [12], network bandwidth variability prediction [23], and crude oil prices [31]. These experiments have demonstrated the potential of this ILS computational approach.

Future work in this direction may include, but not limited to, the followings. First, further theoretical investigations on interval function least squares approximation are needed.

The practical algorithm has produced significantly improved computational results. However, we should explore the theoretical insights and implications of the algorithm. Another direction is related to the algorithm refinement. Chen reported her initial studies on window size selection using discrete Fourier transformation and width adjustment with prediction in [4] and [5]. However, the results were inconclusive. Investigations on practical schemes of width adjustment may have the potential to further improve computational quality. Of course, one may try to apply the algorithm to solve more real world applications and obtain more empirical results for further investigation of this algorithm.

Acknowledgments:

This work is partially supported by the NSF through grants CISE/CCF-0202042 and CISE/CCF-0727798.

Dr. Ling T. He, Carmichael Professor of Finance at the University of Central Arkansas provided insightful domain knowledge and the financial data set used in the application.

The author also wants to express his sincere appreciations to the anonymous referees. Their constructive comments and editorial suggestions have helped the author to improve the quality of this article from the initial submission.

6. REFERENCES

[1] Bentbib, A. H.: Solving the full rank interval least squares problem. Appl. Numer. Math., 41(2): 283–294, 2002.

[2] Brönnimann, H., Melquiond, G., Pion, S.: A proposal to add interval arithmetic to the C++ Standard Library. Technical proposal N1843-05-0103, CIS, Brooklyn Polytechnic University, S Brooklyn (2005)

[3] Chen, N.F., Roll, R., Ross, S.A.: Economic forces and the stock market. Journal of Business 59(3), 383–403 (1986).

[4] Chen, G., Hu, C.: A Computational study on window-size selection in the stock market RILS interval forecasting. In the Proceedings of the 2009 World Congress on Computer Science and Information Engineering, IEEE Computer Society, Vol. 2, pp. 297-301, Los Angeles, CA. (2009).

[5] Chen, G.: Volatile Data Predictive Analysis with Interval Methods, Thesis of Master of Science in Applied Computing, University of Central Arkansas. (2009).

[6] Fama, E., French, K.: Industry costs of equity. Journal of Financial Economics 43, 153–193 (1997).

[7] Gardner, E.: A simple method of computing prediction intervals for time series forecasts. Journal of Management Science 34, 541–546 (1988).

[8] Gay, D. M.: Interval least squares - a diagnostic tool. Reliability in computing: the role of interval methods in scientific computing (ed. R. E. Moore). Academic Press, 183–205 (1988).

[9] Gooijer, J., Hyndman, R.: 25 years of time series forecasting. Journal of Forecasting 22, 443–473 (2006).

[10] He, L., Hu, C.: Impacts of interval measurement on studies of economic variability - evidence from stock market variability forecasting. Journal of Risk Finance 12, 489-507 (2007).

[11] He, L., Hu, C.: Impacts of interval computing on stock market variability forecasting. Journal of Computational Economics, 33(3), 263-276 (2009).

[12] He, L., Hu, C., and Casey, M.: Prediction of variability in mortgage rates - interval computing solutions, Journal of Risk Finance, 10(2), 142-154 (2009).

[13] He, L., Hu, C.: Midpoint method and accuracy of variability forecasting, Journal of Empirical Economics, 38, 705-715 (2010).

[14] Hu, C., Cardenas, A., Hoogendoorn, S., Selpulveda, P.: An interval polynomial interpolation problem and its Lagrange solution. Journal of Reliable Computing 4(1), 27-38 (1998).

[15] Hladik, M.and Cerny, M.:. New approach to interval linear regression. the 24th mini-EURO conference on continuous optimization and information-based technologies in the financial sector (eds. Kasimbeyli, Dincer, Ozpeynirci, and Sakalauskas), selected papers, 167 – 171 (2010).

[16] Hu, C., He, L.: An application of interval methods to the stock market forecasting. Journal of Reliable Computing 13, 423-434 (2007).

[17] Hu, C. et al: Knowledge Processing with Interval and Soft Computing, Springer (2008).

[18] Hu, C.: A note on probabilistic confidence of the stock market ILS interval forecasts. Journal of Risk Finance, 11 (4), 410–415 (2010).

[19] Interval arithmetic in Python, http://pyinterval.googlecode.com/svn/trunk/html/index.html

[20] Kearfott, R.B.: A Fortran 90 environment for research and prototyping of enclosure algorithms for nonlinear equations and global optimization. ACM Transactions on Mathematical Software 21(1), 63–78 (1995).

[21] Knüppel, O.: PROFIL/BIAS - A fast interval library. Computing 53(3–4), 277–287 (1994).

[22] Lerch, M., Tischler, G., Gudenberg, J.W.V., Hofschuster, W., Krämer, W.: FILIB++, a fast interval library supporting containment computations. ACM Transactions on Mathematical Software 32(2), 299–324 (2006).

[23] Marupally, P., Paruchuri, V., and Hu, C.: Bandwidth variability prediction with rolling interval least squares, manuscript being submitted for publication (2011).

[24] Moler, C.B.: Numerical Computing with MATLAB. Society for Industrial and Applied Mathematics, Philadelphia (2004).

[25] Moore, R.E.: Interval Analysis. Prentice–Hall, Upper Saddle River, NJ (1966).

[26] Neumaier, A.: Interval Methods for Systems of Equations, Encyclopedia of Mathematics and its Applications, Vol. 37. Cambridge University Press, Cambridge (1990).

[27] Nooner, M., Hu, C.: IntBox: An interval object-oriented interval computing software toolbox in C++, in Knowledge Processing with Interval and Soft Computing, Springer (2008).

[28] Ross, S.: The arbitrage theory of capital asset pricing. Journal of Economic Theory 13, 341–360 (1976).

[29] Rump, S.M.: A Note on epsilon-inflation. Reliable Computing 4, 371-375, (1998).

[30] Rump, S.M.: INTLAB - INTerval LABoratory,

http://www.ti3.tu-harburg.de/rump/intlab/
[31] Xu, S., Chen, X., Ai, H.: Interval forecasting of crude oil price. In Springer book series: Advances in Soft Computing, vol. 46, Interval/Probabilistic Uncertainty and Non-Classical Logics, 353-363 (2008).

On Calculating the Rate of Linear Convergence of Non-Linear Transformed Sequences

Johannes Grotendorst
Institute for Advanced Simulation
Forschungszentrum Jülich
52425 Jülich, Germany
and
Aachen University of Applied Sciences
j.grotendorst@fz-juelich.de

ABSTRACT

In this paper we calculate the rate of linear convergence of non-linear transformed sequences. Using symbolic computation new results are derived which quantify the convergence acceleration effects of the Aitken transformation S, the iterated Aitken transformation $S^{(2)}$ and the Shanks transformation S_2 for subclasses of linearly convergent sequences. The results are applied to linearly convergent fixed-point iteration methods, i.e. sequences $\{x_n\}_{n\geq 0}$ generated by $x_{n+1} = \phi(x_n)$, where the iteration function $\phi(x)$ possesses a certain number of continuous derivatives. We verify the findings by numerical examples.

Categories and Subject Descriptors

G.1.2 [**Numerical Analysis**]: Approximation—*rational approximation*; G.4 [**Mathematics of Computing**]: Mathematical Software—*algorithm analysis*; I.1.4 [**Symbolic and Algebraic Manipulation**]: Applications

General Terms

Algorithms, Performance, Theory

Keywords

Aitken transformation, iterated Aitken transformation, Shanks transformation, rate of linear convergence, convergence acceleration, fixed-point iteration methods, symbolic computation

1. INTRODUCTION

The speed at which a convergent sequence $\{x_n\}_{n\geq 0}$ approaches its limit is of great importance for practical computations. A basic concept to quantify the convergence speed is introduced in numerical analysis by the following

Definition. Consider a sequence $\{x_n\}_{n\geq 0}$ in $\mathbb{R}$ with $\lim\limits_{n\to\infty} x_n = \xi$ and $x_n \neq \xi$ for all $n \in \mathbb{N}$. The sequence is said to have convergence order $p \geq 1$ if there exists a constant $0 < \rho < \infty$ such that

$$\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|^p} = \rho.$$

Here $\{e_n\}$ is the sequence of errors $e_n = x_n - \xi$. ρ is called asymptotic error constant. If $p = 1$, then $0 < \rho < 1$ is required and $\{x_n\}_{n\geq 0}$ is said to converge linearly and ρ is the rate of linear convergence. For $p = 2$ the convergence is called quadratic. In the numerical literature the order p is called more precisely the Q-order of convergence. The Q stands for quotient, because the definition uses the quotient between two successive error terms [1]. Many numerical methods may be regarded as special cases of fixed-point iterations $x_{n+1} = \phi(x_n), n \geq 0$. The convergence order of fixed-point iterations can be determined if $\phi(x)$ is sufficiently many times continuously differentiable in a neighborhood of the fixed point.

Theorem 1. *Let $\{x_n\}_{n\geq 0}$ be a convergent iteration method, i.e. $x_{n+1} = \phi(x_n)$ and $\lim\limits_{n\to\infty} x_n = \xi$. Assume that the real iteration function $\phi(x)$ is p times continuously differentiable in a neighborhood of ξ with $\mathrm{D}^{(k)}(\phi)(\xi) = 0$, $k = 1 \ldots p-1$, $\mathrm{D}^{(p)}(\phi)(\xi) \neq 0$ ($|\mathrm{D}(\phi)(\xi)| < 1$ for $p = 1$) and $\phi(\xi) = \xi$. Then the iteration method is of order p and $\rho = \dfrac{\left|\mathrm{D}^{(p)}(\phi)(\xi)\right|}{p!}$.*

This Theorem follows directly from the Taylor series expansion of $\phi(x)$ in the neighborhood of ξ and the assumptions (see ref. [1], p. 623, Theorem 6.1.8). In 1926 Aitken introduced the famous Δ^2 process [2], a simple but effective convergence acceleration method. The Δ^2 process is a sequence transformation, i.e. a mapping $S : \{x_n\} \to \{x'_n\}$, where $x'_n = x_n - \dfrac{(\Delta\, x_n)^2}{\Delta^2\, x_n} = \dfrac{x_n\, x_{n+2} - x_{n+1}^2}{x_{n+2} - 2\, x_{n+1} + x_n}$. Here, the Δ operator generates the forward difference sequence $\Delta\, x_n = x_{n+1} - x_n$. Aitken's Δ^2 process (whose name comes from the Δ^2 operator in the denominator) combines three successive elements of the sequence $\{x_n\}$ in a non-linear way. A basic result due to Henrici [3] shows that the Aitken transformed sequence $\{x'_n\}$ derived from a linearly convergent sequence $\{x_n\}$ converges faster to the limit than the sequence $\{x_n\}$.

Theorem 2. *Let $\{x_n\}_{n\geq 0}$ be a linearly converging sequence in $\mathbb{R}$ with $\lim_{n\to\infty} x_n = \xi$, i.e. there exists a constant $0 < |\lambda| < 1$ and a sequence $\{\varepsilon_n\}$ with $e_{n+1} = (\lambda + \varepsilon_n)\cdot e_n$ and $\varepsilon_n \to 0$ for $n \to \infty$. Then the Aitken transformed sequence $\{x'_n\}$ converges faster to ξ than the sequence $\{x_n\}$ in the sense that $\frac{x'_n - \xi}{x_n - \xi} \to 0$ for $n \to \infty$.*

For a proof see ref. [3] (p. 73, Theorem 4.5). Convergence acceleration methods and in particular the Aitken transformation as perhaps the most popular one are studied extensively in monographs from Wimp [4], Weniger [5] , Brezinski and Zaglia [6], and Sidi [7]. Numerical comparisons of nonlinear convergence accelerators were published by Smith and Ford [8, 9]. In addition, several acceleration methods have been made available in computer algebra systems [10, 11, 12].

In the following we use the Maple software [11] for symbolic and numerical calculations, but the results could also be reproduced with other modern computer algebra systems. This paper also aims to demonstrate the capabilities of computer algebra systems for studying numerical methods in the sense of the recent articles of Gander et al. [13, 14]. Quantitative results become possible which are hard to find with paper and pencil only. In Maple we implement infinite sequences $\{x_n\}$ as functions on $\mathbb{N}$. Aitken's transformation S, the difference operator Δ and the composition operator Δ^2 are defined as functional operators:

```
> S:=x -> (n -> x(n) - Delta(x)(n)^2/
                          (Delta@@2)(x)(n));
```

$$S := x \mapsto \left(n \mapsto x(n) - \frac{\Delta(x)(n)^2}{\Delta^2(x)(n)}\right) \tag{1}$$

```
> Delta:=x -> (n->x(n+1) - x(n));
```

$$\Delta := x \mapsto (n \mapsto x(n+1) - x(n)) \tag{2}$$

```
> 'Delta(x)(n)' = Delta(x)(n);
```

$$\Delta(x)(n) = x(n+1) - x(n) \tag{3}$$

```
> '(Delta@@2)(x)(n)' = (Delta@@2)(x)(n);
```

$$\Delta^2(x)(n) = x(n+2) - 2\,x(n+1) + x(n) \tag{4}$$

Hence, the elements of the transformed sequence are computed by

```
> 'S(x)(n)'=normal(S(x)(n));
```

$$S(x)(n) = \frac{x(n)\,x(n+2) - x(n+1)^2}{x(n+2) - 2\,x(n+1) + x(n)} \tag{5}$$

In this paper we also use the more familiar notation $S(x_n)$ for the transformed sequence elements $S(x)(n)$.

2. RATE OF LINEAR CONVERGENCE OF AITKEN TRANSFORMED SEQUENCES

The following result quantifies the convergence acceleration effect of Aitken's Δ^2 process in terms of the rate of convergence of the given sequence $\{x_n\}$. We make an assumption as to how fast ε_n approaches to 0 in the error term of the linearly convergent sequence $\{x_n\}$.

Theorem 3. *Let $\{x_n\}_{n\geq 0}$ be a linearly convergent sequence in $\mathbb{R}$ with $\lim_{n\to\infty} x_n = \xi$, i.e. there exists a constant $0 < |\lambda| < 1$ and a sequence $\{\varepsilon_n\}$ with $e_{n+1} = (\lambda+\varepsilon_n)\cdot e_n$ and $\varepsilon_n \to 0$ for $n \to \infty$. If $\varepsilon_n = \omega_n \cdot e_n^{(p-1)}$ with $p \in \mathbb{N}, p > 1$, and $\omega_n \to \omega \neq 0$ for $n \to \infty$, then the rate of linear convergence of the transformed sequence $\{S(x_n)\}$ is $|\lambda|^p$.*

Proof. Firstly, we introduce two successive error terms of the transformed sequence:

```
> tau[n]:='S(x)(n)-xi';
```

$$\tau_n := S(x)(n) - \xi \tag{6}$$

```
> tau[n+1]:='S(x)(n+1)-xi';
```

$$\tau_{n+1} := S(x)(n+1) - \xi \tag{7}$$

Substituting $x_{n+i} = e_{n+i} + \xi$, $i = 0\ldots 3$, we get for the quotient $\frac{\tau_{n+1}}{\tau_n}$

```
> tau[n]:=subs(seq(x(n+i) = e[n+i]+xi,
               i=0..2),tau[n]):
> tau[n+1]:=subs(seq(x(n+i) = e[n+i]+xi,
               i=0..3),tau[n+1]):
> 'tau[n+1]/tau[n]' = tau[n+1]/tau[n];
```

$$\frac{\tau_{n+1}}{\tau_n} = \frac{e_{n+1} - \dfrac{(e_{n+2} - e_{n+1})^2}{e_{n+3} - 2\,e_{n+2} + e_{n+1}}}{e_n - \dfrac{(e_{n+1} - e_n)^2}{e_{n+2} - 2\,e_{n+1} + e_n}} \tag{8}$$

Next, the error terms e_{n+i}, $i = 1\ldots 3$, are replaced by

```
> for i from 1 to 3 do
    e[n+i]:=(lambda+epsilon[n+i-1])*e[n+i-1]:
  end do;
```

$$\begin{aligned} e_{n+1} &:= (\lambda + \varepsilon_n)\cdot e_n \\ e_{n+2} &:= (\lambda + \varepsilon_{n+1})(\lambda + \varepsilon_n)\cdot e_n \\ e_{n+3} &:= (\lambda + \varepsilon_{n+2})(\lambda + \varepsilon_{n+1})(\lambda + \varepsilon_n)\cdot e_n \end{aligned} \tag{9}$$

and we arrive at the following factored expression

```
> factor(rhs((8)));
```

$$\frac{(\varepsilon_{n+1} - \varepsilon_{n+2})}{(\varepsilon_n - \varepsilon_{n+1})} \times \frac{(\lambda + \varepsilon_{n+1})\left(\lambda^2 + \lambda\,\varepsilon_n + \lambda\,\varepsilon_{n+1} + \varepsilon_{n+1}\varepsilon_n - 2\,\lambda - 2\,\varepsilon_n + 1\right)}{(\lambda^2 + \lambda\,\varepsilon_{n+1} + \varepsilon_{n+2}\lambda + \varepsilon_{n+2}\varepsilon_{n+1} - 2\,\lambda - 2\,\varepsilon_{n+1} + 1)} \tag{10}$$

Now, the assumption $\varepsilon_n = \omega_n \cdot {e_n}^{(p-1)}$ is substituted recursively in expression (10), starting with ε_{n+2}:

```
> subs(seq(epsilon[n+i] = omega[n+i]*e[n+i]^(p-1),
          i=2..0,-1),(10));
```

$$\begin{aligned}
&\Big(\Big(\omega_{n+1}\big((\lambda+\omega_n {e_n}^{p-1})e_n\big)^{p-1} \\
&\quad -\omega_{n+2}\big((\lambda+\omega_{n+1}((\lambda+\omega_n {e_n}^{p-1})e_n)^{p-1}) \\
&\qquad (\lambda+\omega_n {e_n}^{p-1})e_n\big)^{p-1}\Big) \;\times \\
&\quad \Big(\lambda+\omega_{n+1}\big((\lambda+\omega_n {e_n}^{p-1})e_n\big)^{p-1}\Big) \;\times \\
&\quad \Big(\lambda^2+\lambda\,\omega_n {e_n}^{p-1}+\lambda\,\omega_{n+1}\big((\lambda+\omega_n {e_n}^{p-1})e_n\big)^{p-1} \\
&\quad +\omega_{n+1}\big((\lambda+\omega_n {e_n}^{p-1})e_n\big)^{p-1}\omega_n {e_n}^{p-1} \\
&\quad -2\,\lambda-2\,\omega_n {e_n}^{p-1}+1\Big)\Big)\Big/ \\
&\Big(\Big(\omega_n {e_n}^{p-1}-\omega_{n+1}((\lambda+\omega_n {e_n}^{p-1})e_n)^{p-1}\Big) \;\times \\
&\quad \Big(\lambda^2+\lambda\,\omega_{n+1}((\lambda+\omega_n {e_n}^{p-1})e_n)^{p-1} \\
&\quad +\omega_{n+2}((\lambda+\omega_{n+1}((\lambda+\omega_n {e_n}^{p-1})e_n)^{p-1}) \\
&\qquad (\lambda+\omega_n {e_n}^{p-1})e_n)^{p-1}\lambda \\
&\quad +\omega_{n+2}((\lambda+\omega_{n+1}((\lambda+\omega_n {e_n}^{p-1})e_n)^{p-1}) \\
&\qquad (\lambda+\omega_n {e_n}^{p-1})e_n)^{p-1}\omega_{n+1}((\lambda+\omega_n {e_n}^{p-1})e_n)^{p-1} \\
&\quad -2\;\lambda-2\,\omega_{n+1}((\lambda+\omega_n {e_n}^{p-1})e_n)^{p-1}+1\Big)\Big)
\end{aligned} \tag{11}$$

For $n \to \infty$ we have $e_n \to 0$ and $\omega_{n+i} \to \omega$, $i = 0 \ldots 2$, according to the assumption. A direct calculation of this multi-dimensional limit is too complex. The numerator and denominator of expression (11) are polynomials in e_n with coefficients depending on λ and $\omega_{n+i}, i = 0 \ldots 2$. In order to get non-vanishing constant terms we multiply the polynomials by $\frac{1}{{e_n}^{(p-1)}}$ and calculate the limit separately. Assuming $p \in \mathbb{N}$ and $p > 1$ we get:

```
> LN:=limit(numer((11))/e[n]^(p-1),e[n]=0)
      assuming p::integer,p>1:
> LN:=factor(expand(limit(LN,
      {seq(omega[n+i]=omega,i=0..2)})));
```

$$LN := \frac{\lambda^p\,\omega\,(\lambda-1)^2\,(\lambda^p-\lambda)}{\lambda} \tag{12}$$

```
> LD:=limit(denom((11))/e[n]^(p-1),e[n]=0)
      assuming p::integer,p>1:
> LD:=factor(expand(limit(LD,
      {seq(omega[n+i]=omega,i=0..2)})));
```

$$LD := \frac{\omega\,(\lambda-1)^2\,(\lambda^p-\lambda)}{\lambda} \tag{13}$$

With the assumptions $0 < |\lambda| < 1$ and $\omega \neq 0$ it follows for the limit:

```
> 'limit(tau[n+1]/tau[n],n=infinity)' = LN/LD;
```

$$\lim_{n\to\infty} \frac{\tau_{n+1}}{\tau_n} = \lambda^p \tag{14}$$

□

Next, we compare the error terms τ_n of the Aitken transformed sequence with the error terms e_n of the given sequence. Under the assumption of Theorem 3, i.e. with $\varepsilon_n = \omega_n \cdot {e_n}^{(p-1)}, p \in \mathbb{N}, p > 1$, the quotient $\frac{\tau_n}{{e_n}^p}$ converges to the following limit:

```
> tau[n]:=subs(seq(epsilon[n+i]=
      omega[n+i]*e[n+i]^(p-1),i=1..0,-1), tau[n]):
> Q:=limit(simplify(tau[n]/e[n]^p),e[n]=0)
      assuming p::integer,p>1:
> Q:=limit(Q,{seq(omega[n+i]=omega,i=0..1)}):
> 'limit(tau[n]/e[n]^p,n=infinity)'=factor(Q);
```

$$\lim_{n\to\infty} \frac{\tau_n}{{e_n}^p} = \frac{\omega\,(\lambda^p-\lambda)}{(\lambda-1)^2} \tag{15}$$

Eq. (15) generalizes a result of Traub ([15], p. 267) for $p = 2$:

```
> simplify(subs(p=2,'(15)'));
```

$$\lim_{n\to\infty} \frac{\tau_n}{{e_n}^2} = \frac{\lambda\,\omega}{\lambda-1} \tag{16}$$

Now, we apply Theorem 3 to linearly convergent fixed-point iteration methods.

Corollary 1. *Let $\{x_n\}_{n\geq 0}$ be a convergent iteration method, i.e. $x_{n+1} = \phi(x_n)$ and $\lim_{n\to\infty} x_n = \xi$. Assume that the real function $\phi(x)$ is p times continuously differentiable in a neighborhood of ξ with $0 < |\mathrm{D}(\phi)(\xi)| < 1, \mathrm{D}^{(k)}(\phi)(\xi) = 0$, $k = 2 \ldots p-1$, $\mathrm{D}^{(p)}(\phi)(\xi) \neq 0$ and $\phi(\xi) = \xi$. Then the rate of linear convergence of the transformed sequence $\{S(x_n)\}$ is $|\mathrm{D}(\phi)(\xi)|^p$.*

Proof. Using the Taylor series of $\phi(x)$ around ξ with expansion order p and the assumptions in Corollary 1 we get

$$\begin{aligned}
e_{n+1} &= x_{n+1} - \xi \\
&= \phi(x_n) - \xi \\
&= \mathrm{D}(\phi)(\xi)\cdot(x_n-\xi) + \frac{\mathrm{D}^{(2)}(\phi)(\xi)}{2}\cdot(x_n-\xi)^2 + \ldots \\
&\qquad + \frac{\mathrm{D}^{(p)}(\phi)(\eta_n)}{p!}\cdot(x_n-\xi)^p \\
&= \mathrm{D}(\phi)(\xi)\cdot e_n + \frac{\mathrm{D}^{(p)}(\phi)(\eta_n)}{p!}\cdot e_n^p \\
&= \left(\mathrm{D}(\phi)(\xi) + \frac{\mathrm{D}^{(p)}(\phi)(\eta_n)}{p!}\cdot e_n^{(p-1)}\right)\cdot e_n
\end{aligned}$$

Here $e_n = x_n - \xi$ and η_n lies in the interval determined by x_n and ξ. Setting $\lambda = \mathrm{D}(\phi)(\xi)$ and $\omega_n = \frac{\mathrm{D}^{(p)}(\phi)(\eta_n)}{p!}$ we have $\omega_n \to \frac{\mathrm{D}^{(p)}(\phi)(\xi)}{p!} \neq 0$ for $n \to \infty$. Then Corollary 1 follows from Theorem 3.

□

A direct proof of Corollary 1 using eq. (8) requires the application of the chain rule to higher derivatives, i.e. Faà di Bruno's formula [16, 17] has to be used, and therefore is more complex. Furthermore, it should be mentioned that the obtainment of the asymptotic error constant $|\mathrm{D}(\phi)(\xi)|^p$ is a theoretical result which quantifies the convergence improvement of the transformed sequence $\{S(x_n)\}$. In a truly numerical setting the fixed-point ξ is usually not known.

3. NUMERICAL EXAMPLES: AITKEN TRANSFORMED ITERATION SEQUENCES

As a first example we consider a fixed-point iteration method $x_{n+1} = \phi(x_n)$, $n \geq 0$, with starting value $x_0 = 5$ and iteration function

```
> phi:=x->exp(-x);
```

$$\phi := x \mapsto \mathrm{e}^{-x} \tag{17}$$

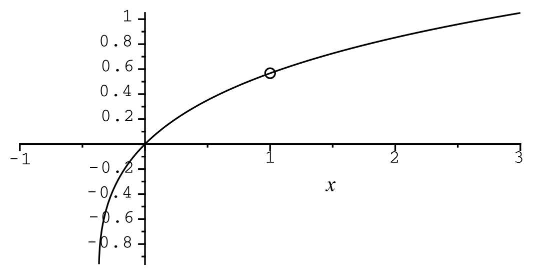
The equation for the fixed point $\phi(x) = x$ is calculated symbolically.

```
> xi:=solve(phi(x)=x,x);
```

$$\xi := W(1) \tag{18}$$

The limit of the iteration sequence $\{x_n\}_{n\geq 0}$ is the value of Lambert's W function at $x = 1$.

```
> plot([LambertW(x),[[1,LambertW(1)]]],x=-1..3,
       style=[line,point], symbol=circle,
       symbolsize=20,color=black);
```



The Lambert W function satisfies the equation $W(x) \cdot \mathrm{e}^{W(x)} = x$. In the literature the value $W(1)$ is also known as Ω constant [18] with the property $\Omega = \mathrm{e}^{-\Omega}$:

```
> alias(Omega=LambertW(1)):
> simplify(Omega-exp(-Omega));
```

$$0 \tag{19}$$

```
> Omega=evalf(Omega,10);
```

$$\Omega = 0.5671432904 \tag{20}$$

We compute the first and second derivative of $\phi(x)$ at the fixed point $x = \Omega$:

```
> seq(simplify((D@@k)(phi)(Omega)),k=1..2);
```

$$-\Omega, \Omega \tag{21}$$

Theorem 1 implies that $\{x_n\}$ converges linearly with the convergence rate $\rho = |\mathrm{D}(\phi)(\Omega)| = \Omega$. From Theorem 2 it follows that Aitken's Δ^2 process accelerates the convergence of this sequence. We study the convergence speed of $\{x_n\}$ and $\{S(x_n)\}$ numerically using functional programming in Maple:

```
> x:=n->(phi@@(n))(5.0);
```

$$x := n \mapsto \phi^{(n)}(5.0) \tag{22}$$

```
> e:=n->x(n)-xi:
> tau:=n->S(x)(n)-xi:
> Digits:=40:
> seq(printf("%5d %10.7f %13.10f %9.6 %9.6f\n",
      n,x(n),S(x)(n),abs(tau(n+1)/tau(n)),
      tau(n)/e(n)^2),n=0..20);
```

```
    0  5.0000000  0.8305245652  0.168238  0.013403
    1  0.0067379  0.6114541051  0.331139  0.141093
    2  0.9932847  0.5818163093  0.302829  0.080800
    3  0.3703582  0.5715866996  0.327725  0.114744
    4  0.6904870  0.5685995062  0.316821  0.095718
    5  0.5013319  0.5676046509  0.323933  0.106522
    6  0.6057234  0.5672927405  0.320213  0.100408
    7  0.5456796  0.5671911462  0.322421  0.103878
    8  0.5794479  0.5671587201  0.321200  0.101911
    9  0.5602076  0.5671482464  0.321903  0.103027
   10  0.5710905  0.5671448858  0.321508  0.102395
   11  0.5649091  0.5671438033  0.321733  0.102753
   12  0.5684118  0.5671434554  0.321605  0.102550
   13  0.5664243  0.5671433435  0.321678  0.102665
   14  0.5675512  0.5671433075  0.321637  0.102600
   15  0.5669120  0.5671432959  0.321660  0.102637
   16  0.5672745  0.5671432922  0.321647  0.102616
   17  0.5670689  0.5671432910  0.321654  0.102628
   18  0.5671855  0.5671432906  0.321650  0.102621
   19  0.5671194  0.5671432905  0.321652  0.102625
   20  0.5671569  0.5671432904  0.321651  0.102623
```

Sequence element $x(20)$ shows 4 correct decimal places of the limit $\xi = \Omega$

```
> 'xi'=evalf(Omega,10);
```

$$\xi = 0.5671432904 \tag{23}$$

whereas $S(x)(20)$ already has 10 valid decimal places. The rate of convergence $|\lambda| = \Omega$ of the fixed-point iteration $x_{n+1} = \phi(x_n)$ is improved by Aitken's Δ^2 process to $|\lambda|^2$:

```
> abs(lambda)^2=evalf(Omega^2,10);
```

$$|\lambda|^2 = 0.3216515118 \tag{24}$$

With $\omega = \dfrac{\mathrm{D}^{(2)}(\phi)(\Omega)}{2!} = \dfrac{\Omega}{2}$ and $\lambda = \mathrm{D}(\phi)(\Omega) = -\Omega$ we get for the limit (16):

```
> simplify(subs(omega=Omega/2,lambda=-Omega,(16)));
```

$$\lim_{n\to\infty} \frac{\tau_n}{{e_n}^2} = \frac{\Omega^2}{2\,(\Omega+1)} \tag{25}$$

```
> evalf((25),10);
```

$$\lim_{n\to\infty} \frac{\tau_n}{{e_n}^2} = 0.1026235169 \tag{26}$$

In addition, Corollary 1 is verified by calculating the rate of convergence of the transformed sequence $\{S(x_n)\}$ symbolically:

```
> 'tau[n+1]/tau[n]'=(S(x)(n+1)-Omega)/
                      (S(x)(n)-Omega);
```

$$\frac{\tau_{n+1}}{\tau_n} = \frac{x(n+1) - \dfrac{(x(n+2)-x(n+1))^2}{x(n+3)-2\,x(n+2)+x(n+1)} - \Omega}{x(n) - \dfrac{(x(n+1)-x(n))^2}{x(n+2)-2\,x(n+1)+x(n)} - \Omega} \tag{27}$$

Using the substitutions $x_{n+i} = \phi^{(i)}(x_n), i = 1\ldots3$, and $x_n \to \Omega$ for $n \to \infty$ we obtain

```
>  subs(seq(x(n+i)=(phi@@i)(x(n)),i=1..3),x(n)=xn,
        rhs((27))):
>  'limit(tau[n+1]/tau[n],n=infinity)'=
          simplify(limit(%,xn=Omega));
```

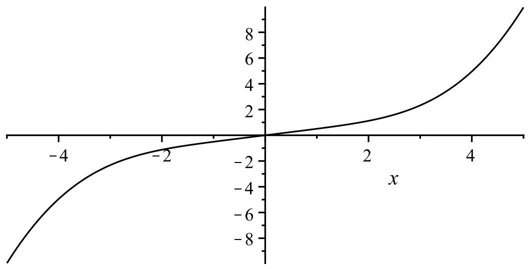
$$\lim_{n\to\infty} \frac{\tau_{n+1}}{\tau_n} = \Omega^2 \tag{28}$$

As second example we consider the fixed-point iteration method $x_{n+1} = \phi(x_n), n \geq 0$, with iteration function

```
>  phi:=x->sin(x)/2+x^3/12;
```

$$\phi := x \mapsto \frac{\sin(x)}{2} + \frac{x^3}{12} \tag{29}$$

```
>  plot(phi(x),x=-5..5);
```



ϕ has the fixed point

```
>  xi:=fsolve(phi(x)=x,x)
```

$$\xi := 0. \tag{30}$$

We compute the first eight derivatives at $\xi = 0$:

```
>  seq((D@@k)(phi)(0),k=1..8);
```

$$\frac{1}{2},\ 0,\ 0,\ 0,\ \frac{1}{2},\ 0,\ -\frac{1}{2},\ 0 \tag{31}$$

Thus, the iteration sequence $x_{n+1} = \phi(x_n)$ converges linearly with $\lambda = \frac{1}{2}$. Corollary 1 implies that the rate of linear convergence of the Aitken transformed sequence $\{S(x_n)\}$ is improved to $\lambda^5 = \frac{1}{32}\,(p = 5)$. Using $\omega = \frac{\mathrm{D}^{(5)}(\phi)(0)}{5!} = \frac{1}{2\cdot 5!} = \frac{1}{240}$ and $\lambda = \frac{1}{2}$ we obtain for the limit (15):

```
>  subs(p=5,omega=1/240,lambda=1/2,(15));
```

$$\lim_{n\to\infty} \frac{\tau_n}{{e_n}^5} = -\frac{1}{128} \tag{32}$$

In the following table the convergence behavior is reproduced numerically using $x_{n+1} = \phi(x_n)$ with starting value $x_0 = 2.0$:

```
>  x:=n->(phi@@n)(2.0);
```

$$x := n \mapsto \phi^{(n)}(2.0) \tag{33}$$

```
>  seq(printf("%5d %8.5f %4.2e %11.8f %11.8f\n",
            n,x(n),S(x)(n),abs(tau(n+1)/tau(n)),
            tau(n)/e(n)^5),n=0..15);

    0  2.00000  -3.74e-01  0.03760516  -0.01169437
    1  1.12132  -1.41e-02  0.03261119  -0.00793844
    2  0.56783  -4.59e-04  0.03148140  -0.00777431
    3  0.28416  -1.44e-05  0.03129998  -0.00779830
    4  0.14209  -4.52e-07  0.03126200  -0.00780866
    5  0.07104  -1.41e-08  0.03125297  -0.00781152
    6  0.03552  -4.42e-10  0.03125074  -0.00781225
    7  0.01776  -1.38e-11  0.03125018  -0.00781244
    8  0.00888  -4.31e-13  0.03125005  -0.00781248
    9  0.00444  -1.35e-14  0.03125001  -0.00781245
   10  0.00222  -4.21e-16  0.03125000  -0.00781250
   11  0.00111  -1.32e-17  0.03125000  -0.00781250
   12  0.00056  -4.11e-19  0.03125000  -0.00781250
   13  0.00028  -1.29e-20  0.03125000  -0.00781250
   14  0.00014  -4.02e-22  0.03125000  -0.00781250
   15  0.00007  -1.26e-23  0.03125000  -0.00781250
```

For comparison we compute the numerical values of the convergence rate of the transformed sequence $\{S(x_n)\}$

```
>  lambda^5=evalf(1/2^5,10);
```

$$\lambda^5 = 0.03125000000 \tag{34}$$

and of limit (32)

```
>  evalf((32),10);
```

$$\lim_{n\to\infty} \frac{\tau_n}{{e_n}^5} = -0.007812500000 \tag{35}$$

4. RATE OF LINEAR CONVERGENCE OF S$_2$ TRANSFORMED SEQUENCES

In 1955 Shanks introduced a generalization of Aitken's transformation [19]. The Shanks transforms $S_k(x_n)$, $k \geq 1$, can be represented as the ratio of Hankel determinants $S_k(x_n) = \frac{|H_{k+1}(x_n)|}{|H_k(\Delta^2 x_n)|}$. Here $H_k(i,j) = V_{i+j-1}$, $1 \leq i,j \leq k$ is a Hankel matrix of order k and the vector V is given by $V = [x_n, x_{n+1}, \ldots, x_{n+2\,k}]$ in the numerator and $V = [\Delta^2 x_n, \Delta^2 x_{n+1}, \ldots, \Delta^2 x_{n+2\,k-2}]$ in the denominator. The Shanks transforms $S_k(x_n)$ may be constructed symbolically by the following procedure:

```
> Shanks:=proc(k::posint,n,x)
     local Delta,H1,H;
     uses LinearAlgebra;
     Delta:=x->(m->x(m+1)-x(m));
     H1:=HankelMatrix(Vector(2*k+1,i->x(n+i-1)));
     H:=HankelMatrix(Vector(2*k-1,i->
        (Delta@@2)(x)(n+i-1)));
     Determinant(H1)/Determinant(H);
  end proc:
```

For $k = 1$ we get Aitken's transformation, i.e. $S_1(x_n) = S(x_n)$:

```
> S[1]:=x->(n->Shanks(1,n,x)):
> 'S[1](x)(n)'=S[1](x)(n);
```

$$S_1(x)(n) = \frac{x(n)\,x(n+2) - x(n+1)^2}{x(n+2) - 2\,x(n+1) + x(n)} \tag{36}$$

```
> S[2]:=x->(n->Shanks(2,n,x)):
> 'S[2](x)(n)'=S[2](x)(n);
```

$$
\begin{aligned}
S_2(x)(n) = \Big(&x(n)\,x(n+2)\,x(n+4) - x(n)\,x(n+3)^2\\
&+ 2\,x(n+1)\,x(n+3)\,x(n+2)\\
&- x(n+1)^2\,x(n+4) - x(n+2)^3\Big) \Big/\\
\Big(&x(n+2)\,x(n+4) + 2\,x(n+3)\,x(n+2)\\
&- 3\,x(n+2)^2 - 2\,x(n+1)\,x(n+4)\\
&+ 2\,x(n+1)\,x(n+3) + 2\,x(n+1)\,x(n+2)\\
&+ x(n)\,x(n+4) - 2\,x(n)\,x(n+3)\\
&+ x(n)\,x(n+2) - x(n+3)^2 - x(n+1)^2\Big)
\end{aligned} \tag{37}
$$

Aitken's transformation S_1 depends on three successive sequence elements, whereas Shanks' transformation S_2 needs five successive sequence elements. In another context formulae (36) and (37) were already used by James Clerk Maxwell in his treatise on electricity and magnetism [20] . The following result quantifies the convergence acceleration effect of the S_2 transformation in terms of the rate of convergence of the given sequence $\{x_n\}$. The assumption for the subclass of linearly convergent sequences $\varepsilon_n = \omega_n \cdot e_n$, $\lim\limits_{n\to\infty} \omega_n = \omega \neq 0$ is chosen more simple than in Theorem 3 because of the computational complexity.

Theorem 4. *Let $\{x_n\}_{n\geq 0}$ be a linearly convergent sequence in $\mathbb{R}$ with $\lim\limits_{n\to\infty} x_n = \xi$, i.e. there exists a constant $0 < |\lambda| < 1$ and a sequence $\{\varepsilon_n\}$ with $e_{n+1} = (\lambda + \varepsilon_n)\cdot e_n$ and $\varepsilon_n \to 0$ for $n \to \infty$. If $\varepsilon_n = \omega_n \cdot e_n$ with $\omega_n \to \omega \neq 0$ for $n \to \infty$, then the rate of linear convergence of the transformed sequence $\{S_2(x_n)\}$ is $|\lambda|^3$.*

Proof. We have to calculate the ratio of two successive error terms of the transformed sequence:

```
>   tau[n]:='S[2](x)(n)-xi';
```

$$\tau_n := S_2(x)(n) - \xi \tag{38}$$

```
>   tau[n+1]:='S[2](x)(n+1)-xi';
```

$$\tau_{n+1} := S_2(x)(n+1) - \xi \tag{39}$$

Substituting $x_{n+i} = e_{n+i} + \xi$, $i = 0\ldots 5$, we get for the quotient $\frac{\tau_{n+1}}{\tau_n}$

```
>   tau[n]:=subs(seq(x(n+i) = e[n+i]+xi,
                 i=0..4),tau[n]):
>   tau[n+1]:=subs(seq(x(n+i) = e[n+i]+xi,
                   i=0..5),tau[n+1]):
>   'tau[n+1]/tau[n]' = tau[n+1]/tau[n];
```

$$
\begin{aligned}
\frac{\tau_{n+1}}{\tau_n} = \Big(\big(&e_{n+1}\,e_{n+3}\,e_{n+5} + 2\,e_{n+2}\,e_{n+4}\,e_{n+3} - {e_{n+3}}^3\\
&- e_{n+1}\,{e_{n+4}}^2 - {e_{n+2}}^2\,e_{n+5}\big) \;\times\\
\big(&2\,e_{n+1}\,e_{n+3} + e_{n+2}\,e_{n+4} - 3\,{e_{n+2}}^2 - {e_{n+3}}^2\\
&+ 2\,e_{n+3}\,e_{n+2} - 2\,e_{n+1}\,e_{n+4} + e_n\,e_{n+2} - {e_{n+1}}^2\\
&+ 2\,e_{n+1}\,e_{n+2} + e_n\,e_{n+4} - 2\,e_n e_{n+3}\big)\Big) \Big/\\
\Big(\big(&e_{n+1}\,e_{n+3} - {e_{n+4}}^2 + 2\,e_{n+2}\,e_{n+4} - {e_{n+2}}^2\\
&- 3\,{e_{n+3}}^2 + e_{n+3}\,e_{n+5} + 2\,e_{n+4}\,e_{n+3} - 2\,e_{n+2}\,e_{n+5}\\
&+ 2\,e_{n+3}\,e_{n+2} + e_{n+1}\,e_{n+5} - 2\,e_{n+1}\,e_{n+4}\big) \;\times\\
\big(&2\,e_{n+1}\,e_{n+3}\,e_{n+2} - e_n\,{e_{n+3}}^2\\
&- {e_{n+1}}^2\,e_{n+4} - {e_{n+2}}^3 + e_n\,e_{n+2}\,e_{n+4}\big)\Big)
\end{aligned} \tag{40}
$$

Next, the error terms e_{n+i}, $i = 1\ldots 5$, are replaced by

```
>  for i from 1 to 5 do
       e[n+i]:=(lambda+epsilon[n+i-1])*e[n+i-1]:
   end do;
```

$$
\begin{aligned}
e_{n+1} &:= (\lambda+\varepsilon_n)\cdot e_n\\
e_{n+2} &:= (\lambda+\varepsilon_{n+1})(\lambda+\varepsilon_n)\cdot e_n\\
e_{n+3} &:= (\lambda+\varepsilon_{n+2})(\lambda+\varepsilon_{n+1})(\lambda+\varepsilon_n)\cdot e_n\\
e_{n+4} &:= (\lambda+\varepsilon_{n+3})(\lambda+\varepsilon_{n+2})(\lambda+\varepsilon_{n+1})(\lambda+\varepsilon_n)\cdot e_n\\
e_{n+5} &:= (\lambda+\varepsilon_{n+4})(\lambda+\varepsilon_{n+3})(\lambda+\varepsilon_{n+2})(\lambda+\varepsilon_{n+1})(\lambda+\varepsilon_n)\cdot e_n
\end{aligned} \tag{41}
$$

```
>  Q1:=factor(rhs((40))):
```

Now, the assumption $\varepsilon_n = \omega_n \cdot e_n$ is substituted recursively in expression `Q1`, starting with ε_{n+4}:

```
>  Q2:=subs(seq(epsilon[n+i]=omega[n+i]*e[n+i],
            i=4..0,-1),Q1):
```

For $n \to \infty$ we have $e_n \to 0$ and $\omega_{n+i} \to \omega$, $i = 0\ldots 4$, according to the assumptions. The numerator and denominator of expression `Q2` are polynomials in e_n with low degrees

```
>  ldegree(numer(Q2),e[n]);
```

$$3 \tag{42}$$

```
>  ldegree(denom(Q2),e[n]);
```

$$3 \tag{43}$$

and coefficients depending on λ and $\omega_{n+i}, i = 0\ldots 4$. As in the proof of Theorem 3 we calculate the limit separately for the numerator and denominator. Therefore, we determine the first non-vanishing terms of the polynomials by Taylor expansion:

```
>  LN:=factor(limit(convert(taylor(numer(Q2),e[n],5),
        polynom),seq(omega[n+i]=omega,i=0..4)));
```

$$LN \;:=\; 2\,\lambda^6\omega^4\,(\lambda+1)^3\,(\lambda-1)^8\cdot {e_n}^4 \tag{44}$$

```
>  LD:=factor(limit(convert(taylor(denom(Q2),e[n],5),
        polynom),seq(omega[n+i]=omega,i=0..4)));
```

$$LD \;:=\; 2\,\lambda^3\omega^4\,(\lambda+1)^3\,(\lambda-1)^8\cdot {e_n}^4 \tag{45}$$

Then, with the assumptions $0 < |\lambda| < 1$ and $\omega \neq 0$ it follows for the limit

```
> 'limit(tau[n+1]/tau[n],n=infinity)' =
                         limit(LN/LD,e[n]=0);
```

$$\lim_{n\to\infty} \frac{\tau_{n+1}}{\tau_n} = \lambda^3 \tag{46}$$

$\square$

Again, Theorem 4 is readily applied to linearly convergent fixed-point iteration methods.

COROLLARY 2. *Let $\{x_n\}_{n\geq 0}$ be a convergent iteration method, i.e. $x_{n+1} = \phi(x_n)$ and $\lim_{n\to\infty} x_n = \xi$. Assume that the real function $\phi(x)$ is two times continuously differentiable in a neighborhood of ξ with $0 < |\mathrm{D}(\phi)(\xi)| < 1$, $\mathrm{D}^{(2)}(\phi)(\xi) \neq 0$ and $\phi(\xi) = \xi$. Then the rate of linear convergence of the transformed sequence $\{S_2(x_n)\}$ is $|\mathrm{D}(\phi)(\xi)|^3$.*

PROOF. Using the Taylor series of $\phi(x)$ around ξ with expansion order 2 and the assumptions in Corollary 2 we get

$$\begin{aligned} e_{n+1} &= x_{n+1} - \xi \\ &= \phi(x_n) - \xi \\ &= \mathrm{D}(\phi)(\xi)\cdot(x_n-\xi) + \frac{\mathrm{D}^{(2)}(\phi)(\xi)}{2}\cdot(x_n-\xi)^2 \\ &= \mathrm{D}(\phi)(\xi)\cdot e_n + \frac{\mathrm{D}^{(2)}(\phi)(\eta_n)}{2!}\cdot e_n^2 \\ &= \left(\mathrm{D}(\phi)(\xi) + \frac{\mathrm{D}^{(2)}(\phi)(\eta_n)}{2!}\cdot e_n\right)\cdot e_n \end{aligned}$$

Here $e_n = x_n - \xi$ and η_n lies in the interval determined by x_n and ξ. Setting $\lambda = \mathrm{D}(\phi)(\xi)$ and $\omega_n = \frac{\mathrm{D}^{(2)}(\phi)(\eta_n)}{2!}$ we have $\omega_n \to \frac{\mathrm{D}^{(2)}(\phi)(\xi)}{2!} \neq 0$ for $n \to \infty$. Then Corollary 2 follows from Theorem 4.

$\square$

Next, we calculate the limit of the quotient $\frac{\tau_n}{e_n^3}$ for $n \to \infty$, where τ_n is the error term of a S_2 transformed sequence and e_n is the error term of the given convergent sequence. If the sequence is generated by a three times continuously differentiable iteration function $\phi(x)$, then e_{n+1} can be expressed in terms of a cubic polynomial in e_n: $e_{n+1} = \left(\mathrm{D}(\phi)(\xi) + \frac{\mathrm{D}^{(2)}(\phi)(\xi)}{2!}e_n + \frac{\mathrm{D}^{(3)}(\phi)(\eta_n)}{3!}{e_n}^2\right)\cdot e_n$
Setting $\lambda = \mathrm{D}(\phi)(\xi)$, $\omega = \frac{\mathrm{D}^{(2)}(\phi)(\xi)}{2!}$, $\theta_n = \frac{\mathrm{D}^{(3)}(\phi)(\eta_n)}{3!}$ and assuming $0 < |\lambda| < 1$ and $\omega \neq 0$ we obtain a linearly convergent sequence with $e_{n+1} = (\lambda + \varepsilon_n)\cdot e_n$ and $\varepsilon_n = \omega\cdot e_n + \theta_n\cdot {e_n}^2$. Now, polynomial ε_n is substituted into the error expression τ_n.

```
> tau1[n]:=subs(seq(epsilon[n+i]=omega*e[n+i]+
  theta[n+i]*e[n+i]^2,i=3..0,-1),tau[n]):
```

Considering $\theta_n \to \theta = \frac{\mathrm{D}^{(3)}(\phi)(\xi)}{3!}$ for $n \to \infty$ we calculate the first term of the Taylor series with respect to e_n:

```
> factor(limit(convert(taylor(tau1[n],e[n]=0,7),
  polynom),seq(theta[n+i]=theta,i=0..3)));
```

$$\frac{\lambda^4\left(\lambda^2\theta - \lambda\,\theta + 2\,\omega^2\right)}{(\lambda+1)\,(\lambda-1)^2}\cdot{e_n}^3 \tag{47}$$

Hence, the limit is evaluated to

```
> 'limit(tau[n]/e[n]^3,n=infinity)'=
                    limit((47)/e[n]^3,e[n]=0);
```

$$\lim_{n\to\infty}\frac{\tau_n}{{e_n}^3} = \frac{\lambda^4\left(\lambda^2\theta - \lambda\,\theta + 2\,\omega^2\right)}{(\lambda+1)\,(\lambda-1)^2} \tag{48}$$

Limit (48) corresponds to eq. (16), the result of Traub (ref. [15]) for Aitken transformed iteration sequences. It depends on the first three derivatives of the iteration function $\phi(x)$. Higher order derivatives are not involved. For example, if $\phi(x)$ is four times continuously differentiable and e_{n+1} is represented as the fourth-order polynomial $e_{n+1} = (\lambda + \varepsilon_n)\cdot e_n$ with $\varepsilon_n = \omega\cdot e_n + \theta\cdot{e_n}^2 + \kappa_n\cdot{e_n}^3$, $\kappa_n = \frac{\mathrm{D}^{(4)}(\phi)(\eta_n)}{4!} \to \kappa = \frac{\mathrm{D}^{(4)}(\phi)(\xi)}{4!}$ for $n \to \infty$, then Taylor expansion of τ_n yields the same first non-vanishing term as in (47):

```
> tau2[n]:=subs(seq(epsilon[n+i]=omega*e[n+i]+
  theta*e[n+i]]^2+kappa[n+i]*e[n+i]^3,i=3..0,-1),
  tau[n]):
> factor(limit(convert(taylor(tau2[n],e[n]=0,7),
  polynom),seq(kappa[n+i]=kappa,i=0..3)));
```

$$\frac{\lambda^4\left(\lambda^2\theta - \lambda\,\theta + 2\,\omega^2\right)}{(\lambda+1)\,(\lambda-1)^2}\cdot{e_n}^3 \tag{49}$$

5. NUMERICAL EXAMPLE: $\mathbf{S_2}$ TRANSFORMED ITERATION SEQUENCE

For the S_2 transformation we reuse the fixed-point iteration method analysed in the first example, i.e. we consider the iteration function

```
> phi:=x->exp(-x);
```

$$\phi := x \mapsto \mathrm{e}^{-x} \tag{50}$$

with fixed point

```
> xi:=fsolve(phi(x)=x,x);
```

$$\xi := \Omega \tag{51}$$

We compute the first six derivatives of $\phi(x)$ at the fixed point $x = \Omega$:

```
> seq(simplify((D@@k)(phi)(Omega)),k=1..6);
```

$$-\Omega,\ \Omega,\ -\Omega,\ \Omega,\ -\Omega,\ \Omega \tag{52}$$

The iteration sequence $x_{n+1} = \phi(x_n)$ converges linearly with $|\lambda| = \Omega$. Corollary 2 implies that the rate of linear convergence of the transformed sequence $\{S_2(x_n)\}$ is improved to $|\lambda|^3 = \Omega^3$. Using $\omega = \frac{\mathrm{D}^{(2)}(\phi)(\Omega)}{2!} = \frac{\Omega}{2}$, $\theta = \frac{\mathrm{D}^{(3)}(\phi)(\Omega)}{3!} = -\frac{\Omega}{6}$ and $\lambda = -\Omega$ we obtain for the limit (48):

```
> simplify(subs(omega=Omega/2,theta=-Omega/6,
                lambda=-Omega,'(48)'));
```

$$\lim_{n\to\infty} \frac{\tau_n}{{e_n}^3} = \frac{\Omega^6\,(\Omega-2)}{6\,(\Omega-1)\,(\Omega+1)^2} \tag{53}$$

In the following table the convergence behavior is reproduced numerically using $x_{n+1} = \phi(x_n)$ with starting value $x_0 = 5.0$:

```
> x:=n->(phi@@(n))(5.0):
> e:=n->x(n)-xi:
> tau:=n->S[2](x)(n)-xi:
> seq(printf("%5d %10.7f %11.8f %10.7 %10.7f\n",
    n,x(n),S[2](x)(n),abs(tau(n+1)/tau(n)),
    tau(n)/e(n)^3),n=0..20);
```

```
    0  5.0000000  0.58000254  0.1706525  0.0001476
    1  0.0067379  0.56494883  0.1680219  0.0124687
    2  0.9932847  0.56751201  0.1877789  0.0047647
    3  0.3703582  0.56707405  0.1783486  0.0090858
    4  0.6904870  0.56715564  0.1844362  0.0065805
    5  0.5013319  0.56714101  0.1811800  0.0079901
    6  0.6057234  0.56714370  0.1830959  0.0071858
    7  0.5456796  0.56714321  0.1820303  0.0076406
    8  0.5794479  0.56714330  0.1826417  0.0073822
    9  0.5602076  0.56714329  0.1822972  0.0075286
   10  0.5710905  0.56714329  0.1824932  0.0074456
   11  0.5649091  0.56714329  0.1823823  0.0074927
   12  0.5684118  0.56714329  0.1824453  0.0074659
   13  0.5664243  0.56714329  0.1824096  0.0074811
   14  0.5675512  0.56714329  0.1824298  0.0074725
   15  0.5669120  0.56714329  0.1824183  0.0074774
   16  0.5672745  0.56714329  0.1824249  0.0074746
   17  0.5670689  0.56714329  0.1824212  0.0074762
   18  0.5671855  0.56714329  0.1824233  0.0074753
   19  0.5671194  0.56714329  0.1824221  0.0074758
   20  0.5671569  0.56714329  0.1824227  0.0074755
```

For comparison we compute the numerical values of the convergence rate of the transformed sequence $\{S_2(x_n)\}$

```
> abs(lambda)^3=evalf(Omega^3,10);
```

$$|\lambda|^3 = 0.1824224968 \tag{54}$$

and of limit (53)

```
> evalf((53),10);
```

$$\lim_{n\to\infty} \frac{\tau_n}{{e_n}^3} = 0.007475611683 \tag{55}$$

6. RATE OF LINEAR CONVERGENCE OF $S^{(2)}$ TRANSFORMED SEQUENCES

The iterated Aitken transformation $S^{(2)}$ combines five successive sequence elements in a non-linear way (see, for instance, ref. [5], p. 225, or ref. [21]) just as the Shanks transformation S_2:

```
> '(S@@2)(x)(n)'=normal(S('S'(x))(n));
```

$$S^{(2)}(x)(n) = \frac{S(x)(n)\,S(x)(n+2) - S(x)(n+1)^2}{S(x)(n+2) - 2\,S(x)(n+1) + S(x)(n)} \tag{56}$$

where

```
> 'S(x)(n+i)'=normal(S(x)(n+i));
```

$$S(x)(n+i) = \frac{x(n+i)\,x(n+i+2) - x(n+i+1)^2}{x(n+i+2) - 2\,x(n+i+1) + x(n+i)} \tag{57}$$

for $i = 0..2$. The following result quantifies the convergence acceleration effect of the $S^{(2)}$ transformation in terms of the rate of convergence of the given sequence $\{x_n\}$.

THEOREM 5. *Let $\{x_n\}_{n\geq 0}$ be a linearly convergent sequence in $\mathbb{R}$ with $\lim_{n\to\infty} x_n = \xi$, i.e. there exists a constant $0 < |\lambda| < 1$ and a sequence $\{\varepsilon_n\}$ with $e_{n+1} = (\lambda + \varepsilon_n)\cdot e_n$ and $\varepsilon_n \to 0$ for $n \to \infty$. If $\varepsilon_n = \omega_n \cdot e_n$ with $\omega_n \to \omega \neq 0$ for $n \to \infty$, then the rate of linear convergence of the transformed sequence $\{S^{(2)}(x_n)\}$ is $|\lambda|^3$.*

PROOF. For the error terms of the transformed sequence

```
> tau[n]:='(S@@2)(x)(n)-xi';
```

$$\tau_n := S^{(2)}(x)(n) - \xi \tag{58}$$

```
> tau[n+1]:='(S@@2)(x)(n+1)-xi';
```

$$\tau_{n+1} := S^{(2)}(x)(n+1) - \xi \tag{59}$$

similar calculations as in the proof of Theorem 4 lead to

```
> tau[n]:=subs(seq(x(n+i) = e[n+i]+xi,
               i=0..4),tau[n]):
  tau[n+1]:=subs(seq(x(n+i) = e[n+i]+xi,
                 i=0..5),tau[n+1]):
  Q3:=normal(tau[n+1]/tau[n]):
  for i from 1 to 5 do
     e[n+i]:=(lambda+epsilon[n+i-1])*e[n+i-1]:
  end do:
  Q3:=factor(Q3):
  Q4:=subs(seq(epsilon[n+i]=omega[n+i]*e[n+i],
          i=4..0,-1),Q3):
> LN:=factor(limit(convert(taylor(numer(Q2),e[n],5),
      polynom),seq(omega[n+i]=omega,i=0..4)));
```

$$LN := -\lambda^6\omega^4\,(\lambda+1)^3\,(\lambda-1)^{17}\cdot {e_n}^4 \tag{60}$$

```
> LD:=factor(limit(convert(taylor(denom(Q4),e[n],5),
      polynom),seq(omega[n+i]=omega,i=0..4)));
```

$$LD := -\lambda^3\omega^4\,(\lambda+1)^3\,(\lambda-1)^{17}\cdot {e_n}^4 \tag{61}$$

```
> 'limit(tau[n+1]/tau[n],n=infinity)' =
                          limit(LN/LD,e[n]=0);
```

$$\lim_{n\to\infty} \frac{\tau_{n+1}}{\tau_n} = \lambda^3 \tag{62}$$

□

Theorem 4 and 5 show that the transformations S_2 and $S^{(2)}$ produce identical convergence improvements for the class of linearly convergent sequences $\{x_n\}$ with $\varepsilon_n = \omega_n \cdot e_n$ and $\omega_n \to \omega \neq 0$ for $n \to \infty$. For fixed-point iteration methods we have

Corollary 3. *Let $\{x_n\}_{n\geq 0}$ be a convergent iteration method, i.e. $x_{n+1} = \phi(x_n)$ and $\lim_{n\to\infty} x_n = \xi$. Assume that the real function $\phi(x)$ is two times continuously differentiable in a neighborhood of ξ with $0 < |\mathrm{D}(\phi)(\xi)| < 1$, $\mathrm{D}^{(2)}(\phi)(\xi) \neq 0$ and $\phi(\xi) = \xi$. Then the rate of linear convergence of the transformed sequence $\{S^{(2)}(x_n)\}$ is $|\mathrm{D}(\phi)(\xi)|^3$.*

Proof. Identical to the proof of Corollary 2.

□

Finally, we calculate the limit of the quotient $\frac{\tau_n}{e_n^3}$ for $n \to \infty$, where τ_n is the error term of a $S^{(2)}$ transformed sequence and e_n is the error term of the given convergent sequence. We consider a linearly convergent sequence $\{x_n\}$ with $e_{n+1} = (\lambda + \varepsilon_n)\cdot e_n$, $0 < |\lambda| < 1$, $\varepsilon_n = \omega\cdot e_n + \theta_n \cdot {e_n}^2$, $\omega \neq 0$ and $\theta_n \to \theta$ for $n \to \infty$. Substituting the polynomial ε_n into the error term τ_n and calculating the first term of the Taylor series with respect to e_n we obtain

```
> tau3[n]:=subs(seq(epsilon[n+i]=omega*e[n+i]+
  theta[n+i]*e[n+i]^2,i=3..0,-1),tau[n]):
> factor(limit(convert(taylor(tau3[n],e[n]=0,7),
  polynom),seq(theta[n+i]=theta,i=0..3)));
```

$$\frac{\lambda^4\left(\lambda\,\theta - \omega^2\right)}{(\lambda+1)\,(\lambda-1)}\cdot {e_n}^3 \tag{63}$$

Thus, the limit is evaluated to

```
> 'limit(tau[n]/e[n]^3,n=infinity)'=
                        limit((63)/e[n]^3,e[n]=0);
```

$$\lim_{n\to\infty}\frac{\tau_n}{{e_n}^3} = \frac{\lambda^4\left(\lambda\,\theta - \omega^2\right)}{(\lambda+1)\,(\lambda-1)} \tag{64}$$

7. NUMERICAL EXAMPLE: $S^{(2)}$ TRANSFORMED ITERATION SEQUENCE

Corollary 3 implies that the iterated Aitken transformation $S^{(2)}$ improves the linear convergence rate $|\lambda| = \Omega$ of $\{x_n\}$, where $x_{n+1} = \phi\,(x_n)$ and $\phi(x) = \mathrm{e}^{-x}$, to $|\lambda|^3 = \Omega^3$. Using $\omega = \frac{\mathrm{D}^{(2)}(\phi)(\Omega)}{2!} = \frac{\Omega}{2}$, $\theta = \frac{\mathrm{D}^{(3)}(\phi)(\Omega)}{3!} = -\frac{\Omega}{6}$ and $\lambda = -\Omega$ we obtain for the limit (64):

```
> simplify(subs(omega=Omega/2,theta=-Omega/6,
                lambda=-Omega,'(64)'));
```

$$\lim_{n\to\infty}\frac{\tau_n}{{e_n}^3} = \frac{\Omega^6}{12\,(1-\Omega)} \tag{65}$$

In the following table the convergence behavior is verified numerically:

```
> x:=n->(phi@@(n))(5.0):
> e:=n->x(n)-xi:
> tau:=n->(S@@2)(x)(n)-xi:
> seq(printf("%5d %10.7f %13.10f %10.7 %10.7f\n",
      n,x(n),(S@@2)(x)(n),abs(tau(n+1)/tau(n)),
      tau(n)/e(n)^3),n=0..20);
```

```
    0  5.0000000  0.57717931  0.0944980  0.0001152
    1  0.0067379  0.56619491  0.2363218  0.0053886
    2  0.9932847  0.56736741  0.1579987  0.0028962
    3  0.3703582  0.56710788  0.1975730  0.0046469
    4  0.6904870  0.56715029  0.1743446  0.0037284
    5  0.5013319  0.56714207  0.1871501  0.0042793
    6  0.6057234  0.56714352  0.1797916  0.0039754
    7  0.5456796  0.56714325  0.1839302  0.0041507
    8  0.5794479  0.56714330  0.1815726  0.0040522
    9  0.5602076  0.56714329  0.1829062  0.0041084
   10  0.5710905  0.56714329  0.1821487  0.0040766
   11  0.5649091  0.56714329  0.1825779  0.0040946
   12  0.5684118  0.56714329  0.1823344  0.0040844
   13  0.5664243  0.56714329  0.1824725  0.0040902
   14  0.5675512  0.56714329  0.1823942  0.0040869
   15  0.5669120  0.56714329  0.1824386  0.0040888
   16  0.5672745  0.56714329  0.1824134  0.0040877
   17  0.5670689  0.56714329  0.1824277  0.0040883
   18  0.5671855  0.56714329  0.1824196  0.0040880
   19  0.5671194  0.56714329  0.1824242  0.0040882
   20  0.5671569  0.56714329  0.1824216  0.0040881
```

The numerical values of the convergence rate of the transformed sequence $\{S^{(2)}(x_n)\}$ and of limit (65) are given by:

```
> abs(lambda)^3=evalf(Omega^3,10);
```

$$|\lambda|^3 = 0.1824224968 \tag{66}$$

```
> evalf((65),10);
```

$$\lim_{n\to\infty}\frac{\tau_n}{{e_n}^3} = 0.004088111042 \tag{67}$$

8. CONCLUSION AND OUTLOOK

It is well-known that Aitken's Δ^2 process and its generalization the Shanks transformation are powerful non-linear convergence acceleration methods. In this paper we have presented new results which quantify the convergence acceleration effects of the Aitken transformation S, the iterated Aitken transformation $S^{(2)}$ and the Shanks transformation S_2 for subclasses of linearly convergent sequences. The formal results are applied to fixed-point iteration methods, i.e. sequences $\{x_n\}$ generated by $x_{n+1} = \phi(x_n)$, where the iteration function $\phi(x)$ possesses a certain number of continuous derivatives. Furthermore, we have used the computer algebra system Maple for symbolic and numerical calculations. It is shown how this powerful mathematical system can be used for analysing numerical methods and generating results which are hard to find with paper and pencil only. In addition, we have investigated how the three transformations S, $S^{(2)}$ and S_2 affect quadratically convergent sequences. With minor variations of the Maple code we could rerun the symbolic calculations in the proofs of Theorems 3, 4 and 5 for second order sequences, i.e. sequences $\{x_n\}$ with the convergence behavior $e_{n+1} = \rho_n\cdot e_n^2$, where $\rho_n \to \rho \neq 0$ for $n \to \infty$. It turns out, that the transformed sequences are again quadratically convergent with the same asymptotic error constant ρ (see also ref. [15], p. 268). Future studies will include the extension of the subclasses of linearly convergent sequences, the higher order transformations S_k and $S^{(k)}$, $k \geq 3$, as well as special transformations such as $\{S_k(x_n)\}_{k\geq 0}$ and $\{S^{(k)}(x_n)\}_{k\geq 0}$, however the computational demand increases rapidly.

9. REFERENCES

[1] G. Dahlquist and Å. Björck, *Numerical Methods in Scientific Computing*, Volume I, SIAM Publication, 2008.

[2] A. C. Aitken, *On Bernoulli's Numerical Solution of Algebraic Equations*, Proc. Roy. Soc. Edinburgh 46, 289-305, 1926.

[3] P. Henrici, *Elements of Numerical Analysis*, John Wiley, New York, 1964.

[4] J. Wimp, *Sequence Transformations and Their Applications*, Vol. 154 in Mathematics in Science and Engineering, Academic Press, 1981.

[5] E. J. Weniger, *Nonlinear Sequence Transformations for the Acceleration of Convergence and the Summation of Divergent Series*, Computer Physics Reports 10, 193-371, 1989.

[6] C. Brezinski and M. R. Zaglia, *Extrapolation Methods - Theory and Practice*, Vol. 2 in Computational Mathematics, North-Holland, 1991.

[7] A. Sidi, *Practical Extrapolation Methods: Theory and Applications*, Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press, 2003.

[8] D. A. Smith and W. F. Ford, *Accelerators of Linear and Logarithmic Convergence*, SIAM J. Numer. Anal. 16, 223-240, 1979.

[9] D. A. Smith and W. F. Ford, *Numerical Comparisons of Nonlinear Convergence Accelerators*, Math. Comput. 38, 481-499, 1982.

[10] J. Grotendorst, *A Maple Package for Transforming Series, Sequences and Functions*, Comput. Phys. Commun. 67, 325-342, 1991.

[11] Maple 15, Maplesoft, a division of Waterloo Maple Inc., Waterloo, Ontario, 2011.

[12] Wolfram Research, Inc., Mathematica, Version 8.0, Champaign, IL, 2010.

[13] W. Gander and D. Gruntz, *Derivation of Numerical Methods Using Computer Algebra*, SIAM Review 41, No. 3, 577-593, 1999.

[14] W. Gander, *Generating Numerical Algorithms Using a Computer Algebra System*, BIT Numerical Mathematics, 46, 491-504, 2006.

[15] J. F. Traub, *Iterative Methods for the Solution of Equations*, Chelsea Publishing Company, New York, 1964.

[16] C. F. Faà di Bruno, *Note sur un nouvelle formule de calcul différentiel*, Quart. J. Math. 1, 359-360, 1857.

[17] W. P. Johnson, *The Curious History of Faà di Bruno's Formula*, Amer. Math. Monthly 109, 217-234, 2002.

[18] E. W. Weisstein, *Omega Constant*, From MathWorld - A Wolfram Web Resource. `http://mathworld.wolfram.com/OmegaConstant.html`

[19] D. Shanks, *Nonlinear Transformations of Divergent and Slowly Convergent Sequences*, J. Math. and Phys. (Cambridge, Mass.) 34, 1-42, 1955.

[20] J. C. Maxwell, *A Treatise on Electricity and Magnetism*, Oxford University Press, Oxford, 1892.

[21] H. H. H. Homeier, *A Hierarchically Consistent, Iterative Sequence Transformation*, Numer. Algo. 8, 47-81, 1994.

A Study of Hensel Series in General Case

Tateaki Sasaki[*]
University of Tsukuba
Tsukuba-shi, Ibaraki 305-8571, Japan
sasaki@math.tsukuba.ac.jp

Daiju Inaba
Mathematics Certifi. Inst.
Katsushika-ku, Tokyo 125-8602, Japan
inaba@math.tsukuba.ac.jp

ABSTRACT

The Hensel series is a series expansion of multivariate algebraic function at its singular point. The Hensel series is computed by the (extended) Hensel construction, and it is expressed in a well-structured form. In previous papers, we clarified theoretically various interesting properties of Hensel series in restricted cases. In this paper, we present a theory of Hensel series in general case. In particular, we investigate the Hensel series arising from non-squarefree initial factor, and derive a formula which shows "fine structure" of the Hensel series. If we trace a Hensel series along a path passing a divergence domain, the Hensel series often jumps from one branch of the algebraic function to another. We investigate the jumping phenomenon near the ramification point, which has not been clarified in our previous papers.

Categories and Subject Descriptors

G.1.2 [**Approximation**]: Nonlinear approximation, approximation of surfaces and contours; I.1.2 [**Symbolic and Algebraic manipulation**]: Algebraic algorithms

General Terms

Theory, Experimentation

Keywords

multivariate algebraic function, series expansion, singular point, convergence, many-valuedness

1. INTRODUCTION

Series expansion is a fundamental tool not only in mathematics but also in numerical as well as algebraic computations. So far, the Taylor series is used mostly but the Taylor expansion breaks down when the expansion point is a singular point. We can expand univariate algebraic functions at singular points (critical points) by Newton-Puiseux's method; see [21]. Newton-Puiseux's method can be generalized to multivariate algebraic functions recursively, giving "multivariate Puiseux series" which are sums of monomials of fractional powers w.r.t. each variable; see [10, 11, 1]. However, multivariate Puiseux series are not easy to use practically. in fact, even the convergence property is not known yet.

In a series of papers [17, 18, 13, 5, 14, 15, 16], we have developed a method of expanding multivariate algebraic function into series at singular points, where the algebraic function is defined to be the roots of a given multivariate polynomial. Since our method is based on the Hensel construction, we named the series obtained by our method *Hensel series.* For bivariate polynomials, our method computes Puiseux series roots simultaneously and efficiently. For multivariate polynomials which define multivariate algebraic functions, the series obtained by our method are very different from multivariate Puiseux series. In this paper, by Hensel series, we always mean multivariate series.

The Hensel series was used so far to the analytic continuation of algebraic functions via singular points [20], solving multivariate algebraic equations in series forms [17, 18], the multivariate polynomial factorization [4], the analytic factorization of polynomials of more than two variables [6, 7], and so on. For some other researches of utilizing series expansions at singular points, see [2, 9, 19, 12, 3].

The Hensel series has following remarkable properties.

1. Each term of Hensel series is well-structured. For example, given $F(x,u,v) = x^2 - 2(u+v)\,x + (u^2+v^2) - (u^3-v^3)$, our method computes the following Hensel series (to the 2nd order w.r.t. an auxiliary variable ξ): $\phi_{\pm}^{(2)}(u,v,\xi) = \alpha_{\pm} \pm \xi\,\frac{u^3-v^3}{2\sqrt{2uv}} \mp \xi^2\,\frac{(u^3-v^3)^2}{16uv\sqrt{2uv}}$, where $\alpha_{\pm} = u+v \pm \sqrt{2uv}$.

2. A simple formula has been derived to express the general term of Hensel series, which allows us to clarify the convergence properties of the Hensel series considerably; the convergence and the divergence domains co-exist in any small neighborhood of the expansion point, and the neighborhood is occupied mostly by the convergence domain.

3. The Hensel series often jumps from one branch of algebraic functions to another when it passes through the divergence domain.

The Hensel series is computed by the *extended Hensel construction* (EHC in short) which is the multivariate Hensel

[*]Professor emeritus

construction at the singular point [8, 17, 18]. The most important concept in the EHC is *Newton polynomial*. So far, we have investigate mostly the cases where the Newton polynomial is squarefree and the given polynomial is either monic [15] or nonmonic [16].

In Sect. 2, we survey the EHC briefly. In Sect. 3, we explain the method of computing Hensel series in the general case, in particular, the Hensel construction in mass and roots, which we devised recently [14]. Furthermore, we consider the Tschirnhaus transformation in nonmonic case and present a method for expressing the roots of the Newton polynomial in a series form. In Sect. 4, we investigate the convergence property generally, in particular, we investigate Hensel series arising from non-squarefree initial factor. In Sect. 5, we investigate the behavior of Hensel series around the ramification point, including the jumping phenomenon.

2. SURVEY OF EHC IN GENERAL CASE

Let $F(x,\boldsymbol{u}) \overset{\text{def}}{=} F(x,u_1,\dots,u_\ell) \in \mathbb{C}[x,u_1,\dots,u_\ell]$ be a given multivariate polynomial. By $\deg(F)$ and $\mathrm{lc}(F)$, we denote the degree and the leading coefficient, respectively, w.r.t. x, of $F(x,\boldsymbol{u})$. We put $\deg(F)=n$ and $\mathrm{lc}(F)=f_n(\boldsymbol{u})$. By $\mathrm{tdeg}(f)$, with $f(\boldsymbol{u}) \in \mathbb{C}[\boldsymbol{u}]$, we denote the *total-degree* of $f(\boldsymbol{u})$ w.r.t. sub-variables $u_1,\dots,u_\ell$; if $T = c\,u_1^{e_1}\cdots u_\ell^{e_\ell}$, $c \in \mathbb{C}$, then $\mathrm{tdeg}(T) = e_1+\cdots+e_\ell$, and $\mathrm{tdeg}(f)$ is the maximum of the total-degrees of terms of $f(\boldsymbol{u})$. By $\mathrm{ord}(f)$ we denote the order of $f(\boldsymbol{u})$, i.e., the minimum of the total-degrees of terms of $f(\boldsymbol{u})$. For rational function $N(\boldsymbol{u})/D(\boldsymbol{u})$ we define the order to be $\mathrm{ord}(N)-\mathrm{ord}(D)$. The order of algebraic function is defined in Subsect. 3.3. By $\mathrm{res}(F,G)$ and $\mathrm{rem}(F,G)$, we denote the resultant of F and G w.r.t. x and the remainder of F divided by G, respectively. By $\mathbb{C}\{(\boldsymbol{u})\}$ we mean a ring of infinite sum of rational functions such as $\sum_{k=0}^{\infty} N_k(\boldsymbol{u})/D_k(\boldsymbol{u})$, where N_k and D_k are homogeneous polynomials in $u_1,\dots,u_\ell$, satisfying $\mathrm{ord}(N_k/D_k)=k$. By F' we denote the partial derivative of F w.r.t. x. By $\|\boldsymbol{u}\|$ we denote the square norm $(|u_1|^2+\cdots+|u_\ell|^2)^{1/2}$ after substituting numbers for $u_1,\dots,u_\ell$. By $\|f(\boldsymbol{u})\| = O(\|\boldsymbol{u}\|^a)$ and $\|f(\boldsymbol{u})\| = o(\|\boldsymbol{u}\|^a)$ we mean $\|f(\boldsymbol{u})\|/\|\boldsymbol{u}\|^a \to$ nonzero-number and $\|f(\boldsymbol{u})\|/\|\boldsymbol{u}\|^a \to 0$, respectively, as $\|\boldsymbol{u}\| \to 0$.

Without loss of generality, we assume that $F(x,\boldsymbol{u})$ is irreducible hence squarefree. Let $\varphi(\boldsymbol{u})$ be an algebraic function defined by a root w.r.t. x, of $F(x,\boldsymbol{u})$: $F(\varphi(\boldsymbol{u}),\boldsymbol{u})=0$. Let $\boldsymbol{s} \overset{\text{def}}{=} (s_1,\dots,s_\ell) \in \mathbb{C}^\ell$ be an expansion point of $\varphi(\boldsymbol{u})$. If $f_n(\boldsymbol{s}) \neq 0$ and $F(x,\boldsymbol{s})$ is squarefree then the algebraic function defined by $F(x,\boldsymbol{u})$ can be expanded into Taylor series in $u_1-s_1,\dots,u_\ell-s_\ell$. Let the squarefree decomposition of $F(x,\boldsymbol{s})$ be $F(x,\boldsymbol{s}) = \tilde{F}_0(x)\prod_{i=1}^{l}[\hat{F}_{i0}(x)]^{m_i}$, where $\tilde{F}_0(x)$ and $\hat{F}_{i0}(x)$ are squarefree, so we have $\gcd(\hat{F}_{i0},\hat{F}_{j0}) = 1$ $(\forall i \neq j)$, and $m_i \geq 2$ for any i. Then, the Hensel construction shows that $F(x,\boldsymbol{u})$ is factored as $F(x,\boldsymbol{u}) = \tilde{F}(x,\boldsymbol{u})\prod_{i=1}^{l}\hat{F}_i(x,\boldsymbol{u})$, where $\tilde{F}(x,\boldsymbol{u}), \hat{F}_i(x,\boldsymbol{u}) \in \mathbb{C}\{\{\boldsymbol{u}\}\}[x]$, $\tilde{F}(x,\boldsymbol{s}) = \tilde{F}_0(x)$ and $\hat{F}_i(x,\boldsymbol{s}) = [\hat{F}_{i0}(x)]^{m_i}$ for each i (the factors of $\mathrm{lc}(F)$ may be distributed among $\tilde{F}$ and $\hat{F}_1,\dots,\hat{F}_l$). In the rest of this paper, we consider only one factor $\hat{F}_i(x,\boldsymbol{u})$, by renaming it to $F(x,\boldsymbol{u})$, hence we have

$$F(x,\boldsymbol{s}) = [F_0(x)]^m, \quad F_0(x) \text{ is squarefree and } m \geq 2. \tag{2.1}$$

Definition 1 (singularness). *We call the expansion point $\boldsymbol{s} \in \mathbb{C}^\ell$ a singular point of algebraic function, or a singular point in short, if $F(x,\boldsymbol{s})$ is not squarefree. If $f_n(\boldsymbol{s}) = 0$ then we say the leading coefficient is singular at $\boldsymbol{s}$.*

Without loss of generality, we assume that the origin $\boldsymbol{u} = \boldsymbol{0}$ is a singular point or $f_n(\boldsymbol{u})$ is singular at the origin, and we consider the series expansion at the origin. Note that at least one root $\varphi(\boldsymbol{u})$ of $F(x,\boldsymbol{u})$ becomes infinity at any zero-point of $f_n(\boldsymbol{u})$.

The conventional Hensel construction is unavailable to expand the roots of $F(x,\boldsymbol{u})$ given in (2.1), and we are necessary to adopt the EHC. The most important concept in the EHC is the Newton polynomial, defined as follows.

Definition 2 (Newton polynomial). *For each term $c\,x^i t^j u_1^{j_1}\cdots u_\ell^{j_\ell}$ of $F(x,t\boldsymbol{u})$, where $c \in \mathbb{C}$ and $j = j_1+\cdots+j_\ell$, plot a dot at the point $(i,\,j)$ in the (e_x,e_t)-plane. The Newton polygon $\mathcal{N}$ for $F(x,\boldsymbol{u})$ is a convex hull containing all the dots plotted. Let the bottom sides of $\mathcal{N}$, counted clockwise, be $\mathcal{L}_1,\dots,\mathcal{L}_q$. We call $\mathcal{L}_1,\dots,\mathcal{L}_q$* Newton lines. *For each $i \in \{1,\dots,q\}$,* Newton polynomial *$F_{\mathcal{L}_i}(x,\boldsymbol{u})$ is defined to be the sum of all the terms plotted on $\mathcal{L}_i$; see Fig. 1.*

Note that $F_{\mathcal{L}_i}(x,\boldsymbol{u})$ may or may not be squarefree.

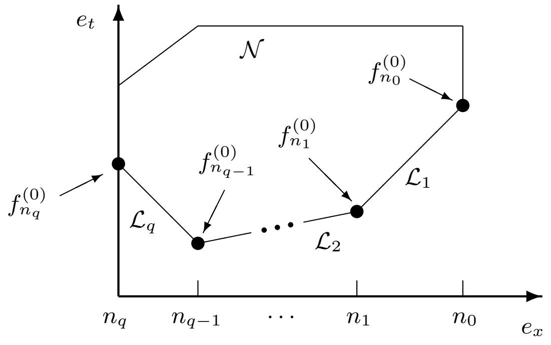


Fig. 1. Newton polygon $\mathcal{N}$ and Newton lines $\mathcal{L}_1,\dots,\mathcal{L}_q$,
$F_{\mathcal{L}_i}(x,\boldsymbol{u}) = [\text{sum of all the terms plotted on } \mathcal{L}_i]$,
$f_{n_i}^{(0)} = [\text{sum of the terms plotted on the left end of } \mathcal{L}_i]/x^{n_i}$.

We sketch the EHC in $\mathbb{C}\{(\boldsymbol{u})\}[x]$; see Sect. 3 for the EHC for computing the Hensel series.

We explain the EHC on Newton line $\mathcal{L}_1$. Let the slope of $\mathcal{L}_1$ be $-\lambda$ (note the "$-$" sign). We see that $F_{\mathcal{L}_1}(x,\boldsymbol{u})$ is homogeneous w.r.t. x^λ and $u_1,\dots,u_\ell$. We say this that $F_{\mathcal{L}_1}(x,\boldsymbol{u})$ is $(\lambda,1)$-homogeneous. Let $F_{\mathcal{L}_1}(x,\boldsymbol{u})$ be factored in $\mathbb{C}[x,\boldsymbol{u}]$ as follows.

$$F_{\mathcal{L}_1}(x,\boldsymbol{u}) = F_0^{(0)}(x)F_1^{(0)}(x,\boldsymbol{u})\cdots F_r^{(0)}(x,\boldsymbol{u}), \tag{2.2}$$

where $F_0^{(0)}(x) = x^{n_1}$ and $\gcd(F_i^{(0)},F_j^{(0)}) = 1$ $(\forall i \neq j)$. Since $F_{\mathcal{L}_1}(x,\boldsymbol{u})$ is $(\lambda,1)$-homogeneous, so are $F_1^{(0)},\dots,F_r^{(0)}$.

We first calculate *Moses-Yun's polynomials* $A_0^{(l)}(x,\boldsymbol{u}),\dots,A_r^{(l)}(x,\boldsymbol{u})$ $(l = 0,1,\dots,n-1)$, which are in $\mathbb{C}\{(\boldsymbol{u})\}[x]$, and satisfy (if $G_0^{(0)}(x) = 1$ then $A_0^{(l)} = 0$)

$$\begin{cases} A_0^{(l)}\dfrac{F_{\mathcal{L}_1}}{F_0^{(0)}} + A_1^{(l)}\dfrac{F_{\mathcal{L}_1}}{F_1^{(0)}} + \cdots + A_r^{(l)}\dfrac{F_{\mathcal{L}_1}}{F_r^{(0)}} = x^l, \\ \deg(A_i^{(l)}) < \deg(F_i^{(0)}) \quad (i = 0,1,\dots,r). \end{cases} \tag{2.3}$$

We next define ideals $\mathcal{I}_k$ $(k = 1,2,3,\dots)$. Let integers $\hat{n}$ and $\hat{\nu}$ satisfy $\hat{n} > 0$, $\hat{\nu}/\hat{n} = \lambda$ and $\gcd(|\hat{\nu}|,\hat{n}) = 1$ ($\hat{n} := 1$ if $\lambda = 0$), then we define $\mathcal{I}_k$ as follows.

$$\mathcal{I}_k = \langle x^n t^{(k+0\hat{\nu})/\hat{n}},\, x^{n-1}t^{(k+1\hat{\nu})/\hat{n}},\, \dots,\, x^0 t^{(k+n\hat{\nu})/\hat{n}} \rangle, \tag{2.4}$$

where $\langle g_1, \ldots, g_m\rangle$ denotes the ideal generated by $g_1, \ldots, g_m$ and t is the total-degree w.r.t. $u_1, \ldots, u_\ell$. Below, we omit the total-degree variable t in $F(x, t\boldsymbol{u})$ etc., for simplicity. Note that we have $F(x, \boldsymbol{u}) \equiv F_{\mathcal{L}_1}(x, \boldsymbol{u}) \pmod{\mathcal{I}_1}$.

Suppose we have calculated $F_1^{(k')}, \ldots, F_r^{(k')}$ up to $k' = k-1$. Then, we calculate

$$\begin{aligned}\delta F^{(k)} &\equiv F - F_0^{(k-1)} F_1^{(k-1)} \cdots F_r^{(k-1)} \pmod{\mathcal{I}_{k+1}} \\ &\overset{\text{def}}{=} \delta f_{n-1}^{(k)}(\boldsymbol{u}) x^{n-1} + \delta f_{n-2}^{(k)}(\boldsymbol{u}) x^{n-2} + \cdots + \delta f_0^{(k)}(\boldsymbol{u}),\end{aligned} \tag{2.5}$$

and construct $F_i^{(k)} \overset{\text{def}}{=} F_i^{(k-1)} + \delta F_i^{(k)}$ $(i = 0, 1, \ldots, r)$ as

$$\delta F_i^{(k)} = A_i^{(n-1)} \delta f_{n-1}^{(k)} + A_i^{(n-2)} \delta f_{n-2}^{(k)} + \cdots + A_i^{(0)} \delta f_0^{(k)}. \tag{2.6}$$

Precisely, we must distribute the factors of leading coefficient $f_n(\boldsymbol{u})$ among $F_1^{(k)}, \ldots, F_r^{(k)}$, but we omit the explanation; see [4] for the distribution.

In order to factor $F(x, \boldsymbol{u})$ in $\mathbb{C}\{(\boldsymbol{u})\}[x]$, we must perform the EHC q times from $\mathcal{L}_1$ to $\mathcal{L}_q$. Let the right ends of $\mathcal{L}_1$ be (n_0, ν_0) hence $n_0 = n$, and the terms plotted on (n_0, ν_0) be $f_n^{(0)}(\boldsymbol{u}) x^n$. Let the left ends of $\mathcal{L}_1, \ldots, \mathcal{L}_q$ be $(n_1, \nu_1), \ldots, (n_q, \nu_q)$, respectively, and the terms plotted on these ends be $f_1^{(0)}(\boldsymbol{u}) x^{n_1}, \ldots, f_q^{(0)}(\boldsymbol{u}) x^0$, respectively (see Fig. 1). For each i $(1 \le i \le q)$, let the Newton polynomial $F_{\mathcal{L}_i}(x, \boldsymbol{u})$ be factored in $\mathbb{C}[x, \boldsymbol{u}]$ as

$$\begin{cases} F_{\mathcal{L}_i}(x, \boldsymbol{u}) = x^{n_i} F_{i1}^{(0)}(x, \boldsymbol{u}) \cdots F_{ir_i}^{(0)}(x, \boldsymbol{u}), \\ \qquad \gcd(F_{ij}^{(0)}, F_{ij'}^{(0)}) = 1 \quad (\forall j \neq j'). \end{cases} \tag{2.7}$$

Then, we have the following theorem; see [13] for the proof. Note that, since $f_{n_i}^{(0)}(\boldsymbol{u})$ appears on both the left end of $\mathcal{L}_i$ and the right end of $\mathcal{L}_{i+1}$, $F_{\mathcal{L}_i}(x, \boldsymbol{u})$ is divided by $f_{n_i}^{(0)}(\boldsymbol{u})$.

Theorem 1 (Sasaki-Inaba 2000). *$F(x, \boldsymbol{u})$ can be factored in $\mathbb{C}\{(\boldsymbol{u})\}[x]$ as*

$$\begin{cases} F(x, \boldsymbol{u}) \;=\; f_0^{(0)}(\boldsymbol{u}) \displaystyle\prod_{i=1}^{q} \Big[F_{i1}^{(\infty)}(x, \boldsymbol{u}) \cdots F_{ir_i}^{(\infty)}(x, \boldsymbol{u}) / f_{n_i}^{(0)}(\boldsymbol{u}) \Big], \\ \qquad F_{ij}^{(\infty)}(x, \boldsymbol{u}) \in \mathbb{C}\{(\boldsymbol{u})\}[x] \quad (i = 1, \ldots, q;\ j = 1, \ldots, r_i), \\ \qquad F_{ij}^{(k)}(x, \boldsymbol{u}) \to F_{ij}^{(0)}(x, \boldsymbol{u}) \in \mathbb{C}[x, \boldsymbol{u}] \quad \text{as } k \to 0. \end{cases} \tag{2.8}$$

The decomposition is unique up to unit factors in $\mathbb{C}\{(\boldsymbol{u})\}$.

Corollary 1. *We can factor $F(x, \boldsymbol{u})$ in $\mathbb{C}\{(\boldsymbol{u})\}[x]$ as follows.*

$$\begin{cases} f_{n_1}^{(0)} \cdots f_{n_{q-1}}^{(0)} F(x, \boldsymbol{u}) = F_{\mathcal{L}_1}^{(\infty)}(x, \boldsymbol{u}) / x^{n_1} \cdots F_{\mathcal{L}_q}^{(\infty)}(x, \boldsymbol{u}) / x^{n_q}, \\ \qquad F_{\mathcal{L}_i}^{(0)}(x, \boldsymbol{u}) = F_{\mathcal{L}_i}(x, \boldsymbol{u}) \quad (i = 1, \ldots, q). \end{cases} \tag{2.9}$$

Here, for each $i \in \{1, \ldots, q\}$, only products of factors of $f_{n_1}^{(0)}(\boldsymbol{u}), \ldots, f_{n_i}^{(0)}(\boldsymbol{u})$ appear in the denominators of terms of $F_{\mathcal{L}_i}^{(\infty)}(x, \boldsymbol{u})$. $\diamondsuit$

3. COMPUTING HENSEL SERIES IN GENERAL CASE

In this section, we consider computing Hensel series, i.e., factoring each $F_{ij}^{(\infty)}(x, \boldsymbol{u})$ in (2.8) into linear factors w.r.t. x, by renaming $F_{ij}^{(\infty)}$ to G (actually, we treat $F_{ij}^{(k)}$ for a sufficiently large k). The first problem we must solve is how to treat the leading coefficient of $G(x, \boldsymbol{u})$, and the second problem is how to compute the Hensel series corresponding to multiple roots of the Newton polynomial. On the other hand, in order to compute and analyze the Hensel series simply, we have developed two techniques, *Hensel construction in mass* and *Hensel construction in roots*, see [14, 15].

As for the first problem, if we use the conventional technique for treating the leading coefficient, then the Hensel construction in mass and roots leads us to complicated expressions; see [14]. In [16], we found that, if we redefine the Newton polynomial as follows, then the leading coefficient can be treated simply and reasonably.

Definition 3 (Newton polynomial for $G(x, \boldsymbol{u})$). *Following Def. 2, construct the* homogeneous Newton polynomial *$G_{\text{hNw}}(x, \boldsymbol{u})$ for $G(x, \boldsymbol{u})$. The* Newton polynomial *$G_{\text{New}}(x, \boldsymbol{u})$ for $G(x, \boldsymbol{u})$ is defined by replacing the leading coefficient of $G_{\text{hNw}}(x, \boldsymbol{u})$ by $\mathrm{lc}(G)$.*

Let $G_{\text{hNw}}(x, \boldsymbol{u}) = g_n^{(0)}(\boldsymbol{u}) x^n + g_{n-1}^{(0)}(\boldsymbol{u}) + \cdots + g_0^{(0)}(\boldsymbol{u})$. The slope of the Newton line for $G(x, \boldsymbol{u})$ is $-\lambda$, hence each root $\alpha_i(\boldsymbol{u})$ of $G_{\text{hNw}}(x, \boldsymbol{u})$ is such that $\|\alpha_i(\boldsymbol{u})\| = O(\|\boldsymbol{u}\|^\lambda)$ no matter how the coefficients of $g_n^{(0)}$ and $g_0^{(0)}$ are small compared with other coefficients. From the viewpoint of applications, however, if $\|\alpha_n(\boldsymbol{u})\| > 1000\,\|\alpha_i(\boldsymbol{u})\|$ for $\forall i < n$ (this case occurs when $\|g_n^{(0)}\|$ is very small) or $\|\alpha_1(\boldsymbol{u})\| < \|\alpha_i(\boldsymbol{u})\|/1000$ for $\forall i > 1$ (this case occurs when $\|g_0^{(0)}\|$ is very small), for example, then treating such roots equally is unpractical. Therefore, we define normality and closeness.

Definition 4 (normality, closeness). *Let $N = \max\{\|g_i^{(0)}(\boldsymbol{u})\| / \|\boldsymbol{u}\|^{\nu+(n-i)\lambda} \mid i = 0, 1, \ldots, n\}$ and S be a given small positive number, $0 < S \ll 1$, specifying the practical measure of smallness. If $\|g_n^{(0)}(\boldsymbol{u})\| / \|\boldsymbol{u}\|^\nu > SN$ and $\|g_0^{(0)}(\boldsymbol{u})\| / \|\boldsymbol{u}\|^{\nu+n\lambda}) > SN$ for generic $\boldsymbol{u}$ then we say $G_{\text{hNw}}(x, \boldsymbol{u})$ is* normal. *Let $\alpha_i(\boldsymbol{u})$ and $\alpha_j(\boldsymbol{u})$ $(i \neq j)$ be roots of $G_{\text{hNw}}(x, \boldsymbol{u})$ or $G_{\text{New}}(x, \boldsymbol{u})$. We say $\alpha_i(\boldsymbol{u})$ and $\alpha_j(\boldsymbol{u})$ are mutually* close *if, for generic $\boldsymbol{u}$ such that $\|\boldsymbol{u}\| < S$, we have $\|\alpha_i(\boldsymbol{u}) - \alpha_j(\boldsymbol{u})\| < S\,\|\alpha_i(\boldsymbol{u})\|$. Furthermore, we say $\boldsymbol{u}$ is* close *to $\boldsymbol{u}_0$ if $\|\boldsymbol{u} - \boldsymbol{u}_0\| < S\,\|\boldsymbol{u}\|$.*

The second problem can be solve by the Tschirnhaus transformation, see Subsect. 3.2 for details. However, if the Newton polynomial has close roots then the situation becomes complicated. In Def. 3, we have defined the Newton polynomial $G_{\text{New}}(x, \boldsymbol{u})$ so that we have $\mathrm{lc}(G_{\text{New}}) = \mathrm{lc}(G)$. If $\mathrm{lc}(G)$ has higher order terms then $G_{\text{New}}(x, \boldsymbol{u})$ is squarefree in most cases. The reason is as follows. Let $R = \mathrm{res}(G_{\text{New}}, G'_{\text{New}})$ be considered as a polynomial in g_n. Among the coefficients of $G_{\text{New}}(x, \boldsymbol{u})$, only g_n has higher order terms, hence R becomes 0 identically iff every coefficient w.r.t. g_n, of R is 0, which is very rare. On the other hand, we are interested in the behaviors of Hensel series near the expansion point, where the Hensel series are determined mostly by $G_{\text{hNw}}(x, \boldsymbol{u})$ defined in Def. 3. If $G_{\text{hNw}}(x, \boldsymbol{u})$ has multiple roots then $G_{\text{New}}(x, \boldsymbol{u})$ will have close roots. Therefore, we confine ourselves to investigating the following two cases in this paper.

Case 1) $G_{\text{hNw}}(x, \boldsymbol{u})$ has neither multiple nor close root.
Case 2) $G_{\text{hNw}}(x, \boldsymbol{u})$ has multiple roots but no close root.

We do not investigate the case that $G_{\text{hNw}}(x, \boldsymbol{u})$ has close roots, which will be a theme in approximate algebra.

3.1 Case 1) Hensel construction in mass and roots

The case 1) has been studied in [16], by assuming that $G_{\rm New}(x,\boldsymbol{u})$ is squarefree. In order to make this paper self-contained, we describe previous theories briefly.

Let the roots of $G_{\rm New}(x,\boldsymbol{u})$ be $\alpha_1(\boldsymbol{u}),\dots,\alpha_n(\boldsymbol{u})$, which we often write as $\alpha_1,\dots,\alpha_n$.

$$\begin{aligned} G_{\rm New}(x,\boldsymbol{u}) = g_n(\boldsymbol{u})(x-\alpha_1(\boldsymbol{u}))\cdots(x-\alpha_n(\boldsymbol{u})), \\ \alpha_i(\boldsymbol{u}) \neq \alpha_j(\boldsymbol{u}) \ \ (\forall i\neq j). \end{aligned} \tag{3.1}$$

The $\alpha_i(\boldsymbol{u})$ is usually an algebraic function. We define $\overline{G}(x,\boldsymbol{u},\xi)$ by introducing an auxiliary variable ξ, as follows (we put $\xi=1$ after finishing the Hensel construction).

$$\begin{cases} G(x,\boldsymbol{u}) & \overset{\rm def}{=} & G_{\rm New}(x,\boldsymbol{u}) + G_{\rm h}(x,\boldsymbol{u}), \\ \overline{G}(x,\boldsymbol{u},\xi) & \overset{\rm def}{=} & G_{\rm New}(x,\boldsymbol{u}) + \xi\, G_{\rm h}(x,\boldsymbol{u}). \end{cases} \tag{3.2}$$

We factor $G_{\rm New}(x,\boldsymbol{u})$ as $G_{\rm New}=G_1^{(0)}\tilde{G}^{(0)}$, where $G_1^{(0)}(x,\boldsymbol{u}) = x-\alpha_1(\boldsymbol{u})$ and $\tilde{G}^{(0)}(x,\boldsymbol{u}) = G_{\rm New}(x,\boldsymbol{u})/(x-\alpha_1(\boldsymbol{u}))$. Then, choosing $G_1^{(0)}(x,\boldsymbol{u})$ and $\tilde{G}^{(0)}(x,\boldsymbol{u})$ as initial factors, we perform the Hensel construction of $\overline{G}(x,\boldsymbol{u},\xi)$ w.r.t. modulus ξ (*Hensel construction in mass*), obtaining

$$\begin{cases} \overline{G}(x,\boldsymbol{u},\xi) \equiv G_1^{(k)}(x,\boldsymbol{u},\xi)\cdot\tilde{G}^{(k)}(x,\boldsymbol{u},\xi) \quad (\bmod\ \xi^{k+1}), \\ G_1^{(k)}(x,\boldsymbol{u},\xi) = x-\phi_1^{(k)}(\boldsymbol{u},\xi), \qquad \phi_1^{(0)}(\boldsymbol{u},\xi)=\alpha_1(\boldsymbol{u}). \end{cases} \tag{3.3}$$

The $\phi_1^{(\infty)}(\boldsymbol{u},1)$ is the Hensel series corresponding to $\alpha_1(\boldsymbol{u})$.

Actual Hensel construction is performed as follows. Suppose we computed $G_1^{(k')}$ and $\tilde{G}^{(k')}$ up to $k'=k-1$. Then, we compute the k-th order residual $\xi^k\delta G^{(k)}$ as follows.

$$\begin{aligned} &\overline{G}(x,\boldsymbol{u},\xi) - G_1^{(k-1)}(x,\boldsymbol{u},\xi)\tilde{G}^{(k-1)}(x,\boldsymbol{u},\xi) \quad (\bmod\ \xi^{k+1}) \\ \equiv\ & \xi^k\,[\delta g_{n-1}^{(k)}(\boldsymbol{u})x^{n-1} + \delta g_{n-2}^{(k)}(\boldsymbol{u})x^{n-2} + \cdots + \delta g_0^{(k)}(\boldsymbol{u})]. \end{aligned} \tag{3.4}$$

We compute the k-th order Hensel factors $G_1^{(k)}$ and $\tilde{G}^{(k)}$ by the well-known formula

$$\begin{aligned} G_1^{(k)}(x,\boldsymbol{u},\xi) &= G_1^{(k-1)}(x,\boldsymbol{u},\xi) + \xi^k \textstyle\sum_{l=0}^{n-1} A_l\,\delta g_l^{(k)}(\boldsymbol{u}), \\ \tilde{G}^{(k)}(x,\boldsymbol{u},\xi) &= \tilde{G}^{(k-1)}(x,\boldsymbol{u},\xi) + \xi^k \textstyle\sum_{l=0}^{n-1} B_l\,\delta g_l^{(k)}(\boldsymbol{u}). \end{aligned} \tag{3.5}$$

Here, $A_l = A_l(x,\boldsymbol{u})$ and $B_l = B_l(x,\boldsymbol{u})$, $0\le l\le n-1$, are determined to satisfy (A_n and B_n are unnecessary because $\deg(\delta G^{(k)})\le n-1$)

$$\begin{cases} A_l(x,\boldsymbol{u})\,\dfrac{G_{\rm New}(x,\boldsymbol{u})}{x-\alpha_1} + B_l(x,\boldsymbol{u})(x-\alpha_1) = x^l, \\ A_l(x,\boldsymbol{u}), B_l(x,\boldsymbol{u}) \in \mathbb{C}(\boldsymbol{u})[x,\alpha_1], \\ \deg_x(A_l) < 1, \qquad \deg_x(B_l) < n-1. \end{cases} \tag{3.6}$$

We express $A_l(x,\boldsymbol{u})$ and $B_l(x,\boldsymbol{u})$ in $\alpha_1,\dots,\alpha_n$. Substituting α_1 and α_j ($j\ge 2$) for x in (3.6), we obtain $A_l(\alpha_1,\boldsymbol{u}) = \alpha_1^l/\prod_{j=2}^n(\alpha_1-\alpha_j)$ and $B_l(\alpha_j,\boldsymbol{u}) = \alpha_j^l/(\alpha_j-\alpha_1)$. By these, $A_l(x,\boldsymbol{u})$ and $B_l(x,\boldsymbol{u})$ are determined uniquely as follows.

$$\begin{cases} A_l(x,\boldsymbol{u}) = \dfrac{\alpha_1^l}{G'_{\rm New}(\alpha_1,\boldsymbol{u})}, \\ B_l(x,\boldsymbol{u}) = \displaystyle\sum_{j=2}^n \alpha_j^l\,\frac{G_{\rm New}(x,\boldsymbol{u})/(x-\alpha_1)(x-\alpha_j)}{G'_{\rm New}(\alpha_j,\boldsymbol{u})}. \end{cases} \tag{3.7}$$

Since $\delta G^{(1)}(x,\boldsymbol{u}) = G_{\rm h}(x,\boldsymbol{u})$, substituting the above expressions of $A_l(x,\boldsymbol{u})$ and $B_l(x,\boldsymbol{u})$ for those in (3.5), we obtain the following theorem (*Hensel construction in roots*).

Theorem 2 (Sasaki-Inaba 2009). *If $G_{\rm New}(x,\boldsymbol{u})$ is squarefree then the Hensel factors $G_1^{(\infty)}$ and $\tilde{G}^{(\infty)}$ in (3.3) are expressed as follows.*

$$G_1^{(\infty)}(x,\boldsymbol{u},\xi) = x-\alpha_1 + \sum_{k=1}^{\infty}\xi^k\,\frac{\delta G^{(k)}(\alpha_1,\boldsymbol{u})}{G'_{\rm New}(\alpha_1,\boldsymbol{u})}, \tag{3.8}$$

$$\begin{aligned} \tilde{G}^{(\infty)}(x,\boldsymbol{u},\xi) &= \frac{G_{\rm New}(x,\boldsymbol{u})}{x-\alpha_1} \\ &\quad + \sum_{j=2}^n \frac{G_{\rm New}(x,\boldsymbol{u})}{(x-\alpha_1)(x-\alpha_j)}\Big(\sum_{k=1}^\infty \xi^k\,\frac{\delta G^{(k)}(\alpha_j,\boldsymbol{u})}{G'_{\rm New}(\alpha_j,\boldsymbol{u})}\Big), \end{aligned} \tag{3.9}$$

where $\delta G^{(1)} = G_{\rm h}(x,\boldsymbol{u})$ and the k-th order residual $\delta G^{(k)}$ ($k\ge 2$) is given as follows.

$$\begin{aligned} \delta G^{(k)}(x,\boldsymbol{u}) &= -\sum_{j=2}^n \frac{G_{\rm New}(x,\boldsymbol{u})}{(x-\alpha_1)(x-\alpha_j)} \\ &\quad \times\Big(\sum_{k'=1}^{k-1}\frac{\delta G^{(k')}(\alpha_1,\boldsymbol{u})}{G'_{\rm New}(\alpha_1,\boldsymbol{u})}\,\frac{\delta G^{(k-k')}(\alpha_j,\boldsymbol{u})}{G'_{\rm New}(\alpha_j,\boldsymbol{u})}\Big). \end{aligned} \tag{3.10}$$

The above formula (3.8) gives (we omit 1 in $\phi_1^{(k)}(\boldsymbol{u},1)$).

$$\phi_1^{(\infty)}(\boldsymbol{u}) = \alpha_1 - \sum_{k=1}^\infty \frac{\delta G^{(k)}(\alpha_1,\boldsymbol{u})}{G'_{\rm New}(\alpha_1,\boldsymbol{u})}. \tag{3.11}$$

Definition 5 (rational, algebraic series). *If $\alpha_i(\boldsymbol{u})\in\mathbb{C}(\boldsymbol{u})$ then the truncated Hensel series $\phi_i^{(k)}(\boldsymbol{u})$, $k\ge 1$, is called* rational, *otherwise it is called* algebraic.

3.2 Case 2) Tschirnhaus transformation

We can assume that the factorization in (2.7) is the squarefree decomposition. The squarefree factors of $F_{\mathcal{L}_1}(x,\boldsymbol{u})$ have been treated in Case 1). Therefore, in Case 2), we have only to consider such an $H(x,\boldsymbol{u}) \overset{\rm def}{=} F_{ij}^{(\infty)}(x,\boldsymbol{u})$ that its homogeneous Newton polynomial $H_{\rm hNw}(x,\boldsymbol{u})$ is of the form $H_{\rm hNw}(x,\boldsymbol{u}) = [H_0(x,\boldsymbol{u})]^m$, where $m\ge 2$ and $H_0(x,\boldsymbol{u})$ is irreducible in $\mathbb{C}[x,\boldsymbol{u}]$. Let $\deg(H_0)=r$ and the roots of $H_0(x,\boldsymbol{u})$ be $\bar{\alpha}_1,\dots,\bar{\alpha}_r$:

$$H_0(x,\boldsymbol{u}) = h_r^{(0)}(\boldsymbol{u})\,(x-\bar{\alpha}_1(\boldsymbol{u}))\cdots(x-\bar{\alpha}_r(\boldsymbol{u})). \tag{3.12}$$

We first define the order of algebraic function $\alpha(\boldsymbol{u})$.

Definition 6. *Let $\alpha(\boldsymbol{u})$ be a root of $H_{\rm New}(x,\boldsymbol{u})$ or $H_{\rm hNw}(x,\boldsymbol{u})$. We define the* order *of $\alpha(\boldsymbol{u})$ to be* $\mathrm{ord}(\alpha)=\lambda$, *where $-\lambda$ is the slope of the Newton line. We plot $\alpha(\boldsymbol{u})^j x^i$ at point $(i,\lambda j)$ in the (e_x,e_t)-plane defined in Def. 2.*

Let $\deg(H)=n=mr$. Since $H_{\rm hNw}(x,\boldsymbol{u})$ has m multiple roots, $H_{\rm New}(x,\boldsymbol{u})$ will have r clusters of m close roots. Hence, we express the roots of $H_{\rm New}(x,\boldsymbol{u})$ as follows.

$$\begin{cases} H_{\rm New}(x,\boldsymbol{u}) = h_n(\boldsymbol{u})\prod_{i=1}^r(x-\alpha_{i1}(\boldsymbol{u}))\cdots(x-\alpha_{im}(\boldsymbol{u})), \\ \|\alpha_{ij}(\boldsymbol{u})-\bar{\alpha}_i(\boldsymbol{u})\| = o(\|\boldsymbol{u}\|^\lambda) \ \ (i=1,\dots,r;\ j=1,\dots,m). \end{cases} \tag{3.13}$$

In the monic case, a procedure to handle $H(x,\boldsymbol{u})$ was established in [18]. We generalize the procedure to nonmonic case.

If $r=1$ then go to the following Tschirnhaus transformation. If $r\ge 2$ then, putting $H_i^{(0)}(x,\boldsymbol{u}) = \prod_{j=1}^m(x-\alpha_{ij}(\boldsymbol{u}))$

$(i=1,\ldots,r)$, we perform the EHC of $H(x,\boldsymbol{u})$, with initial factors $H_1^{(0)},\ldots,H_r^{(0)}$, obtaining (actually, we perform the EHC only up to a finite order)

$$\left\{\begin{array}{l} H(x,\boldsymbol{u}) = h_n(\boldsymbol{u})\, H_1^{(\infty)}(x,\boldsymbol{u})\cdots H_r^{(\infty)}(x,\boldsymbol{u}), \\ H_i^{(\infty)}(x,\boldsymbol{u}) \in \mathbb{C}\{(\boldsymbol{u})\}[x,\alpha_{i1},\ldots,\alpha_{im}] \quad (i=1,\ldots,r). \end{array}\right. \tag{3.14}$$

Let $H_i^{(\infty)}(x,\boldsymbol{u}) = x^m + h_{i,m-1}x^{m-1} + \cdots + h_{i,0}$. We apply the following Tschirnhaus transformation to each factor $H_i^{(\infty)}(x,\boldsymbol{u})$.

$$H_i^{(\infty)}(x,\boldsymbol{u}) \;\mapsto\; \check{H}_i(x,\boldsymbol{u}) = H_i^{(\infty)}(x - \frac{h_{i,m-1}}{m},\boldsymbol{u}). \tag{3.15}$$

Since $\alpha_{i1},\ldots,\alpha_{im}$ are the roots of Newton polynomial for $H_i^{(\infty)}$, the above transformation maps the Newton polynomial to x^m. Therefore, we can apply the EHC to $\check{H}_i(x,\boldsymbol{u})$; let the slope of the Newton line for $\check{H}_i$ be $-\lambda'$, then we have $\lambda' > \lambda$. If the homogeneous Newton polynomial for $\check{H}_i$ is still not squarefree, we apply the Tschirnhaus transformation to $\check{H}_i$ again.

Remark 1. *The above formulation is, although consistent with Def. 3, quite complicated from the viewpoint of computation. For easier computation, we may use the homogeneous Newton polynomial $H_{\rm hNw} = h_n^{(0)}(\boldsymbol{u})\prod_{i=1}^r (x-\bar{\alpha}_i(\boldsymbol{u}))^m$ for defining the initial factors of Hensel construction, and perform the Hensel construction to satisfy*

$$\left\{\begin{array}{l} H(x,\boldsymbol{u}) = h_n(\boldsymbol{u}) H_1^{(\infty)}(x,\boldsymbol{u})\cdots H_r^{(\infty)}(x,\boldsymbol{u}), \\ H_i^{(0)} = (x-\bar{\alpha}_i)^m, \quad H_i^{(\infty)} \in \mathbb{C}\{(\boldsymbol{u})\}[x,\bar{\alpha}_i] \quad (i=1,\ldots,r). \end{array}\right. \tag{3.16}$$

In the next sections, we consider only the homogeneous Newton polynomials.

3.3 Computing $\alpha_1,\ldots,\alpha_n$ in a series form

In the nonmonic case, we defined the Newton polynomial $G_{\rm New}(x,\boldsymbol{u})$ so that $\mathrm{lc}(G_{\rm New}) = \mathrm{lc}(G) = g_n(\boldsymbol{u})$. If $g_n(\boldsymbol{u})$ contains higher order terms then the homogeneity of $G_{\rm New}(x,\boldsymbol{u})$ is destroyed by $g_n(\boldsymbol{u})$. The behavior of the Hensel series near the expansion point is determined mostly by $G_{\rm hNw}(x,\boldsymbol{u})$ and we want to know the behaviors of $\alpha_1(\boldsymbol{u}),\ldots,\alpha_n(\boldsymbol{u})$ near the expansion point. So, we consider a method to express the roots in a series form.

Let $g_n(\boldsymbol{u})$ be split into lower and higher terms:

$$\begin{array}{l} g_n(\boldsymbol{u}) = g_{\rm l}(\boldsymbol{u}) + g_{\rm h}(\boldsymbol{u}), \\ \mathrm{ord}(g_{\rm l}) = \mathrm{tdeg}(g_{\rm l}) < \mathrm{ord}(g_{\rm h}). \end{array} \tag{3.17}$$

Then, we can express $\tilde{G}_{\rm New} \overset{\rm def}{=} g_{\rm l}/(g_{\rm l}+g_{\rm h})\, G_{\rm New}$ as follows.

$$\left\{\begin{array}{rcl} \tilde{G}_{\rm New} &=& \tilde{G}_{\rm New,l} + \tilde{G}_{\rm New,h}, \\ \tilde{G}_{\rm New,l} &=& g_{\rm l}(\boldsymbol{u})x^n + g_{n-1}(\boldsymbol{u})x^{n-1} + \cdots + g_0(\boldsymbol{u}), \\ \tilde{G}_{\rm New,h} &=& -(g_{\rm h}/g_n)(g_{n-1}(\boldsymbol{u})x^{n-1} + \cdots + g_0(\boldsymbol{u})). \end{array}\right. \tag{3.18}$$

Note that $\tilde{G}_{\rm New,l}$ is $(\lambda,1)$-homogeneous and $\tilde{G}_{\rm New,h}$ can be regarded to be of higher order than $\tilde{G}_{\rm New,l}$, because $g_{\rm h}/g_n = g_{\rm h}/(g_{\rm l}+g_{\rm h}) = (g_{\rm h}/g_{\rm l})[1-(g_{\rm h}/g_{\rm l})+(g_{\rm h}/g_{\rm l})^2 - \cdots]$.

Let the roots of $\tilde{G}_{\rm New,l}$ be $\alpha_1^{(0)}(\boldsymbol{u}),\ldots,\alpha_n^{(0)}(\boldsymbol{u})$:

$$\tilde{G}_{\rm New,l} = g_{\rm l}(\boldsymbol{u})\,(x-\alpha_1^{(0)}(\boldsymbol{u}))\cdots(x-\alpha_n^{(0)}(\boldsymbol{u})). \tag{3.19}$$

Applying the EHC to $\tilde{G}_{\rm New}$, with initial factors $(x–\alpha_1^{(0)})$ and $\tilde{G}_{\rm New,l}/(x–\alpha_1^{(0)})$, we can compute $\alpha_1(\boldsymbol{u})$ in a series form, just similarly as we have computed the power-series root $\phi_1^{(k)}(\boldsymbol{u})$ in the previous subsection. We show the resulting formula.

Theorem 3. *If $\tilde{G}_{\rm New,l}(x,\boldsymbol{u})$ is squarefree then $\alpha_1(\boldsymbol{u})$ can be expressed as follows.*

$$\alpha_1(\boldsymbol{u}) \;=\; \alpha_1^{(0)}(\boldsymbol{u}) - \sum_{k=1}^{\infty} \frac{\delta\tilde{G}^{(k)}(\alpha_1^{(0)},\boldsymbol{u})}{\tilde{G}'_{\rm New,l}(\alpha_1^{(0)},\boldsymbol{u})}, \tag{3.20}$$

where $\delta\tilde{G}^{(1)} = \tilde{G}_{\rm New,h}(x,\boldsymbol{u})$ and the k-th order residual $\delta\tilde{G}^{(k)}$ $(k\ge 2)$ is given as follows.

$$\begin{array}{rl} \delta\tilde{G}^{(k)}(x,\boldsymbol{u}) = & -\displaystyle\sum_{j=2}^{n} \frac{\tilde{G}_{\rm New,l}(x,\boldsymbol{u})}{(x-\alpha_1^{(0)})(x-\alpha_j^{(0)}} \\ & \times\Big(\displaystyle\sum_{k'=1}^{k-1} \frac{\delta\tilde{G}^{(k')}(\alpha_1^{(0)},\boldsymbol{u})}{\tilde{G}'_{\rm New,l}(\alpha_1^{(0)},\boldsymbol{u})}\,\frac{\delta\tilde{G}^{(k-k')}(\alpha_j^{(0)},\boldsymbol{u})}{\tilde{G}'_{\rm New,l}(\alpha_j^{(0)},\boldsymbol{u})}\Big). \end{array} \tag{3.21}$$

4. CONVERGENCE PROPERTY NEAR THE EXPANSION POINT

In this section, we study the convergence property of Hensel series near the expansion point (= the origin). After surveying some previous results in **4.1**, we study the Hensel series computed on different Newton lines in **4.2**, and study the Hensel series corresponding to multiple roots of the Newton polynomial in **4.3**. We will show that, in both Case 1) and Case 2) mentioned in Sect. 3, the divergence domain of any Hensel series occupies only a small part of the neighborhood of the expansion point.

4.1 When the Newton polynomial is squarefree

We consider $G(x,\boldsymbol{u})$ defined at the beginning of Sect. 3, the Newton polynomial which is factored as in (3.1). Theorem 2 tells us that the Hensel series $\phi_1^{(\infty)}(\boldsymbol{u})$ corresponding to $\alpha_1(\boldsymbol{u})$ is expressed as in (3.11). Therefore, we can see the convergence property of $\phi_1^{(\infty)}(\boldsymbol{u})$ by investigating the behaviors of $\delta G^{(k)}(\alpha_1,\boldsymbol{u})$ and $G'_{\rm New}(\alpha_1,\boldsymbol{u})$. Such investigations were done in [15, 16], and we show several important results (Theorem 6 is new). We put

$$G(x,\boldsymbol{u}) = g_n(\boldsymbol{u})x^n + g_{n-1}(\boldsymbol{u})x^{n-1} + \cdots + g_0(\boldsymbol{u}). \tag{4.1}$$

Note that at least one algebraic function diverges at the zero-points of $g_n(\boldsymbol{u})$ and becomes 0 at the zero-points of $g_0(\boldsymbol{u})$. We first show an important lemma; below, by $o(\|\boldsymbol{u}\|^a)$ we mean either $O(\|\boldsymbol{u}\|^{a+1})$ or less.

Lemma 1. *We have the following order estimations near the expansion point, so long as $\boldsymbol{u}$ is generic and not close to the zero-points of $g_n(\boldsymbol{u})$ and $\alpha_i(\boldsymbol{u})-\alpha_j(\boldsymbol{u})$ $(\forall i\neq j)$.*

$$\begin{array}{rcll} \|\alpha_1\|,\, \|\alpha_1-\alpha_j\| &=& O(\|\boldsymbol{u}\|^{\lambda}) & (j=2,\ldots,n), \quad (4.2) \\ \|G'_{\rm New}(\alpha_i,\boldsymbol{u})\| &=& O(\|\boldsymbol{u}\|^{\nu+(n-1)\lambda}) & (i=1,\ldots,n), \quad (4.3) \\ \|G_{\rm h}(\alpha_i,\boldsymbol{u})\| &=& o(\|\boldsymbol{u}\|^{\nu+n\lambda}) & (i=1,\ldots,n). \quad (4.4) \end{array}$$

Let $\delta\phi_1^{(k)}(\boldsymbol{u})$ be the coefficient of ξ^k-term of Hensel series $\phi_1^{(\infty)}(\boldsymbol{u},\xi)$. We have the following order estimation near the expansion point, so long as $\boldsymbol{u}$ is not close to the zero-points of $g_n(\boldsymbol{u})$, $g_0(\boldsymbol{u})$ and $\alpha_i(\boldsymbol{u})-\alpha_j(\boldsymbol{u})$ $(\forall i\neq j)$.

$$\|\delta\phi_1^{(k)}(\boldsymbol{u})\| = o(\|\boldsymbol{u}\|^{\lambda+k-1}). \tag{4.5}$$

THEOREM 4. *Assume that $G_{\mathrm{hNw}}(x,\boldsymbol{u})$ is normal and has no close roots. Except near the zero-points of $g_n(\boldsymbol{u})$, any divergence domain of Hensel series $\phi_1^{(\infty)}(\boldsymbol{u})$ starts from the expansion point and runs outside radially along the zero-points of $\alpha_1(\boldsymbol{u})-\alpha_j(\boldsymbol{u})$, $2 \le j \le n$.*

THEOREM 5. *Let S_r be the surface of the hypersphere $\|\boldsymbol{u}\|^2 = r^2$, where r is a small real positive number. Suppose S_r contains zero-points of $g_n(\boldsymbol{u})$ on which $\alpha_1(\boldsymbol{u})$ diverges, and let δS_r be small neighborhood of the zero-points, on S_r. Let $\check{S}_r$ be $S_r - \delta S_r$. Then, we have*

$$\frac{[\text{divergence area of } \phi_1^{(\infty)}(\boldsymbol{u}) \text{ on } \check{S}_r]}{[\text{convergence area of } \phi_1^{(\infty)}(\boldsymbol{u}) \text{ on } \check{S}_r]} \to 0 \quad \text{as} \quad r \to 0. \tag{4.6}$$

THEOREM 6. *Assume that the Newton polynomial $G_{\mathrm{New}}(x,\boldsymbol{u})$ is normal and has no close roots. Let $\alpha_i(\boldsymbol{u})$ be any root of $G_{\mathrm{New}}(x,\boldsymbol{u})$. Let $\mathcal{D}_i$ be the divergence domain of $\phi_i^{(\infty)}(\boldsymbol{u})$ running along the zero-points of $\alpha_i(\boldsymbol{u})-\alpha_j(\boldsymbol{u})$ $(i \ne j)$. Then, the spread of $\mathcal{D}_i$ on the surface $\|\boldsymbol{u}\| = r$, $0 < r \ll 1$, is of order $\|G_{\mathrm{h}}(\alpha_i,\boldsymbol{u})\|/\|G_{\mathrm{New}}(\alpha_i,\boldsymbol{u})\|$ for generic $\boldsymbol{u}$. Therefore, if $\|\alpha_i(\boldsymbol{u})-\alpha_j(\boldsymbol{u})\|$ becomes small then $\mathcal{D}_i$ on the surface $\|\boldsymbol{u}\| = r$ spreads in proportion to $1/\|\alpha_i(\boldsymbol{u})-\alpha_j(\boldsymbol{u})\|$ approximately.*

Proof Consider formulas in Theorem 2. The initial term of $\phi_i^{(\infty)}(\boldsymbol{u})$ is $\alpha_i(\boldsymbol{u})$ which does not diverge, because G_{New} is normal. The order-estimations in (4.5) tells us that $\mathcal{D}_i$ is determined mostly by the second term $G_{\mathrm{h}}(\alpha_i,\boldsymbol{u})/G'_{\mathrm{New}}(\alpha_i,\boldsymbol{u})$ because $\boldsymbol{u}$ is near the origin. Hence, the theorem follows. $\diamondsuit$

It is remarkable that the Hensel series converges not only in any small neighborhood of the expansion point but also at almost all points of the small neighborhood.

4.2 Hensel series on different Newton lines

We consider the Hensel factors in (2.9) in details, by separating the Hensel factor on $\mathcal{L}_1$, for example, from $F(x,\boldsymbol{u})$, as follows.

$$\begin{cases} F(x,\boldsymbol{u}) = \hat{F}^{(\infty)}(x,\boldsymbol{u})\bar{F}^{(\infty)}(x,\boldsymbol{u}), \\ \hat{F}^{(0)}(x,\boldsymbol{u}) = x^{n_1}, \quad \bar{F}^{(0)}(x,\boldsymbol{u}) = F_{\mathcal{L}_1}(x,\boldsymbol{u})/x^{n_1}. \end{cases} \tag{4.7}$$

We use the Hensel construction in mass for this separation; below, we omit the auxiliary variable ξ for simplicity. Moses-Yun's polynomials are expressed concisely, as follows.

LEMMA 2 (SASAKI-INABA 2000). *Moses-Yun's polynomials $A^{(l)}$ and $B^{(l)}$ satisfying $A^{(l)}\bar{F}^{(0)}(x,\boldsymbol{u}) + B^{(l)}x^{n_1} = x^l$ $(l = 0,1,\dots,n-1)$, with $\deg(A^{(l)}) < n_1$ and $\deg(B^{(l)}) < n-n_1$, are given uniquely as follows.*

$$\begin{cases} \text{for } l \ge n_1: & A^{(l)} = 0, \quad B^{(l)} = x^{l-n_1}, \\ \text{for } l < n_1: & A^{(l)} = \bar{F}_{\mathrm{Inv}\langle x^{n_1-l}\rangle}x^l, \\ & B^{(l)} = [1-\bar{F}_{\mathrm{Inv}\langle x^{n_1-l}\rangle}\bar{F}^{(0)}]/x^{n_1-l}. \end{cases} \tag{4.8}$$

Here, $\bar{F}_{\mathrm{Inv}\langle x^m\rangle}$ is a polynomial in the main variable x, satisfying $\deg(\bar{F}_{\mathrm{Inv}\langle x^m\rangle}) < m$ and $\bar{F}_{\mathrm{Inv}\langle x^m\rangle}\bar{F}^{(0)} \equiv 1 \pmod{x^m}$, i.e., it is the inverse of $\bar{F}^{(0)}(x,\boldsymbol{u})$ modulo x^m.

We can compute $\bar{F}_{\mathrm{Inv}\langle x^m\rangle}$ by the power-series division of 1 by $\bar{F}^{(0)}$; note than $\bar{F}^{(0)}$ has a nonzero x^0-term. Therefore, only $f_{n_1}^{(0)}(\boldsymbol{u})$ (= the x^0-term of $\bar{F}^{(0)}$) appears in the denominators of $A^{(l)}$ and $B^{(l)}$ $(l < n_1)$.

We investigate the Hensel series arising from $\hat{F}^{(\infty)}(x,\boldsymbol{u})$ and $\bar{F}^{(\infty)}(x,\boldsymbol{u})$ near the expansion point. We put $[1-\bar{F}_{\mathrm{Inv}\langle x^{n_1-l}\rangle}\bar{F}^{(0)}] = Q_l(x,\boldsymbol{u})x^{n_1-l}$. Note that $\bar{F}_{\mathrm{Inv}\langle x^m\rangle}$ has $1/f_{n_1}^{(0)}(\boldsymbol{u})$ as the constant term w.r.t. x. Since $\bar{F}^{(0)}(x,\boldsymbol{u})$ is $(\lambda,1)$-homogeneous, $\bar{F}_{\mathrm{Inv}\langle x^m\rangle}(x,\boldsymbol{u})$ in (4.8) is $(\lambda,1)$-homogeneous, and $Q_l(x,\boldsymbol{u})$ is also $(\lambda,1)$-homogeneous. Let

$$\begin{aligned} \delta F^{(k)} &\equiv F(x,\boldsymbol{u}) - \hat{F}^{(k-1)}(x,\boldsymbol{u})\bar{F}^{(k-1)}(x,\boldsymbol{u}) \pmod{\xi^{k+1}} \\ &\overset{\text{def}}{=} \delta f_{n-1}^{(k)}(\boldsymbol{u})x^{n-1} + \delta f_{n-2}(\boldsymbol{u})x^{n-2} + \cdots + \delta f_0^{(k-1)}(\boldsymbol{u}). \end{aligned} \tag{4.9}$$

Put $\hat{F}^{(k)} = \hat{F}^{(k-1)} + \delta\hat{F}^{(k)}$ and $\bar{F}^{(k)} = \bar{F}^{(k-1)} + \delta\bar{F}^{(k)}$. We can compute $\delta\hat{F}^{(k)}$ and $\delta\bar{F}^{(k)}$ as $\delta\hat{F}^{(k)} = \sum_{l=0}^{n-1} A^{(l)}\delta f_l^{(k)}$ and $\delta\bar{F}^{(k)} = \sum_{l=1}^{n-1} B^{(l)}\delta f_l^{(k)}$. Substituting $A^{(l)}$ and $B^{(l)}$ into these expressions and noting that $\bar{F}_{\mathrm{Inv}\langle x^m\rangle} \equiv \bar{F}_{\mathrm{Inv}\langle x^{n_1}\rangle} \pmod{x^m}$, which is valid for any $m < n_1$, we obtain

$$\begin{aligned} \delta\hat{F}^{(k)} &= \textstyle\sum_{l=0}^{n_1-1} \bar{F}_{\mathrm{Inv}\langle x^{n_1-l}\rangle}\,\delta f_l^{(k)}(\boldsymbol{u})x^l \qquad (4.10)\\ &\equiv \bar{F}_{\mathrm{Inv}\langle x^{n_1}\rangle}\left[\textstyle\sum_{l=0}^{n_1-1}\delta f_l^{(k)}(\boldsymbol{u})x^l\right] \pmod{x^{n_1}}, \end{aligned}$$

$$\begin{aligned} \delta\bar{F}^{(k)} &= \delta f_{n-1}^{(k)}(\boldsymbol{u})x^{n-n_1-1} + \cdots + \delta f_{n_1}^{(k)}(\boldsymbol{u}) \qquad (4.11)\\ &\quad + \textstyle\sum_{l=0}^{n_1-1}\delta f_l^{(k)}(\boldsymbol{u})\,Q_l(x,\boldsymbol{u}). \end{aligned}$$

Let the slope of Newton line $\mathcal{L}_i$ be $-\lambda_i$ and put $\deg(F_{\mathcal{L}_i}(x,\boldsymbol{u})/x^{n_i}) = d_i$ $(i = 1,\dots,q)$. Let $\alpha_{i,1},\dots,\alpha_{i,d_i}$ be the roots of $F_{\mathcal{L}_i}(x,\boldsymbol{u})$. The following lemma is essentially the same as Corollary 1.

LEMMA 3. *For each $i \in \{1,\dots,q\}$, let $\phi_{i,j}^{(\infty)}$ $(j = 1,\dots,d_i)$ be d_i Hensel series obtained by the EHC on the Newton line $\mathcal{L}_i$, then they have $\alpha_{i,1}(\boldsymbol{u}),\dots,\alpha_{i,d_i}(\boldsymbol{u})$ as initial terms.*

Proof Let $F(x,\boldsymbol{u}) = F_{\mathcal{L}_i}(x,\boldsymbol{u}) + F_{\mathrm{h}}(x,\boldsymbol{u})$, and consider the above $\delta\hat{F}^{(k)}$ and $\delta\bar{F}^{(k)}$. Since $\delta F^{(1)} = F_{\mathrm{h}}(x,\boldsymbol{u})$, all the terms of $\delta\hat{F}^{(1)}$ and $\delta\bar{F}^{(1)}$ contain coefficients of $F_{\mathrm{h}}(x,\boldsymbol{u})$, and so are $\delta\hat{F}^{(k)}$ and $\delta\bar{F}^{(k)}$ $(\forall k \ge 2)$. Hence, the homogeneous Newton line for $\delta\bar{F}^{(\infty)}$ is $F_{\mathcal{L}_1}(x,\boldsymbol{u})$. Since $\bar{F}_{\mathrm{Inv}\langle x^{n_1}\rangle}$ is $(\lambda,1)$-homogeneous and $\bar{F}_{\mathrm{Inv}\langle x^{n_1}\rangle}$ has the constant term $1/f_{n_1}^{(0)}(\boldsymbol{u})$, the Newton polynomial for $\delta\hat{F}^{(\infty)}$ is determined by the lowest order terms of $\sum_{l=0}^{n_1-1}\delta f_l^{(k)}(\boldsymbol{u})x^l$. Hence, the Newton polynomial in the range $n_2 \le e_x \le n_1$ is $F_{\mathcal{L}_2}(x,\boldsymbol{u})$. This discussion is available for $F_{\mathcal{L}_i}$ $(i \ge 3)$, too. $\diamondsuit$

By Lemma 3, we classify the n Hensel series of $F(x,\boldsymbol{u})$ into q sets, as follows.

$$\{\phi_{i,1}^{(\infty)}(\boldsymbol{u}),\dots,\phi_{i,d_i}^{(\infty)}(\boldsymbol{u})\} \quad (i = 1,\dots,q), \tag{4.12}$$

where $\phi_{i,j}^{(\infty)}(\boldsymbol{u}) = \alpha_{i,j}(\boldsymbol{u})+$ higher-order-terms $(j=1,\dots,d_i)$.

THEOREM 7. *Assume that Newton polynomials $F_{\mathcal{L}_1}(x,\boldsymbol{u})$, $\dots$, $F_{\mathcal{L}_q}(x,\boldsymbol{u})$ have neither multiple nor close roots. Furthermore, let $\boldsymbol{u}$ be not close to the zero-points of $\alpha_{i,1}(\boldsymbol{u})-\alpha_{i,j}(\boldsymbol{u})$ $(i = 1,\dots,q;\ j = 1,\dots,d_i)$ and the zero-points of $f_{n_0}^{(0)}(\boldsymbol{u}), f_{n_1}^{(0)}(\boldsymbol{u}),\dots,f_{n_q}^{(0)}(\boldsymbol{u})$. Then, near the expansion point, the divergence domain of Hensel series $\phi_{i,j}^{(\infty)}(\boldsymbol{u})$ is well separated from that of $\phi_{i',j'}^{(\infty)}(\boldsymbol{u})$ for any $i' \ne i$, j and j'.*

Proof We note that if $\boldsymbol{u}$ is close to a zero-point of $\alpha_{i,1}-\alpha_{i.j}$ $(j > 1)$ then $\|F'_{\mathcal{L}_i}(\alpha_{i,1},\boldsymbol{u})\|$ becomes small, and the Hensel series $\phi_{i,1}^{(\infty)}(\boldsymbol{u})$ may diverge, see (3.8). Furthermore, if $\boldsymbol{u}$ is close to a zero-point of $f_{n_{i-1}}^{(0)}(\boldsymbol{u})$ (or $f_{n_i}^{(0)}(\boldsymbol{u})$) then some root $\alpha_{i,j}$ $(1 \le j \le d_i)$ will be very large (resp. very small), but such cases are excluded in the theorem. Then, the Hensel series $\phi_{i,j}^{(\infty)}(\boldsymbol{u})$ is dominated by its initial term $\alpha_{i,j}(\boldsymbol{u})$ for $\|\boldsymbol{u}\| \ll 1$, and we have $\|\alpha_{i,j}(\boldsymbol{u})\| = O(\|\boldsymbol{u}\|^{\lambda_i})$ near the expansion point. The Hensel series $\phi_{i',j'}^{(\infty)}(\boldsymbol{u})$ is dominated by its initial term $\alpha_{i',j'}(\boldsymbol{u})$ and $\|\alpha_{i',j'}(\boldsymbol{u})\| = O(\|\boldsymbol{u}\|^{\lambda_{i'}})$ near the expansion point. Since λ_i and $\lambda_{i'}$ are rational numbers satisfying $\lambda_i \neq \lambda_{i'}$, Theorem 6 with order-estimations in Lemma 1 assures that, near the expansion point, divergence domains are well separated one another. ♢

4.3 When the Newton polynomial is not squarefree

We consider Hensel series arising from the Hensel factor $H_i^{(\infty)}(x,\boldsymbol{u})$ in (3.16). We will see that the Hensel series is of the form $\bar{\alpha}_i +$ [higher-order-terms showing "fine structure"].

First, we consider denominators appearing in $H_1^{(\infty)}(x,\boldsymbol{u})$, which is constructed by the EHC with initial factors $H_1^{(0)} = (x-\bar{\alpha}_1)^m$ and $\tilde{H}^{(0)} = [H_0(x,\boldsymbol{u})]^m/(x-\bar{\alpha}_1)^m$, where $H_0(x,\boldsymbol{u}) = (x-\bar{\alpha}_1)\cdots(x-\bar{\alpha}_r)$.

Lemma 4. *Except for denominators introduced by the separation of $H \overset{\text{def}}{=} F_{ij}^{(\infty)}$ in (2.8), the denominators appearing in the terms of $H_1^{(\infty)}(x,\boldsymbol{u})$ are only powers of* $\mathrm{res}(H_0, H_0')$.

Proof Let $A^{(l)}$ and $B^{(l)}$ $(0 \le l \le mr-1)$ be Moses-Yun's polynomials satisfying $A^{(l)}\tilde{H}^{(0)} + B^{(l)}H_1^{(0)} = x^l$, $\deg(A^{(l)}) < m$ and $\deg(B^{(l)}) < m(r-1)$. We can compute $A^{(0)}$ and $B^{(0)}$ by the extended Euclidean algorithm and $A^{(l)}$ and $B^{(l)}$ $(l \ge 1)$ as $A^{(l)} = \mathrm{rem}(x^l A^{(0)}, H_1^{(0)})$ and $B^{(l)} = \mathrm{rem}(x^l B^{(0)}, \tilde{H}^{(0)})$. The extended Euclidean algorithm allows us to express $A^{(0)}$ and $B^{(0)}$ in $\mathbb{C}(\boldsymbol{u},\bar{\alpha}_1)[x]$, where the denominators are $\mathrm{res}(H_1^{(0)}, \tilde{H}^{(0)}) = \left[\mathrm{res}((x-\bar{\alpha}_1), (x-\bar{\alpha}_2)\cdots(x-\bar{\alpha}_r))\right]^{m^2} = [(\bar{\alpha}_1-\bar{\alpha}_2)\cdots(\bar{\alpha}_1-\bar{\alpha}_r)]^{m^2} = \left[H_0'(\bar{\alpha}_1,\boldsymbol{u})\right]^{m^2}$. We can compute the inverse of $H_0'(\bar{\alpha}_1,\boldsymbol{u})$ by the extended Euclidean algorithm: let $C(x,\boldsymbol{u})$ and $D(x,\boldsymbol{u})$ be in $\mathbb{C}(\boldsymbol{u})[x]$, satisfying $C(x,\boldsymbol{u})H_0(x,\boldsymbol{u}) + D(x,\boldsymbol{u})H_0'(x,\boldsymbol{u}) = 1$, $\deg(C) < \deg(H_0')$ and $\deg(D) < \deg(H_0)$, then we have $D(\bar{\alpha}_1,\boldsymbol{u}) = [H_0'(\bar{\alpha}_1,\boldsymbol{u})]^{-1}$. The extended Euclidean algorithm tells us that the denominator of $D(x,\boldsymbol{u})$ is $\mathrm{res}(H_0, H_0')$. ♢

Next, we consider the Hensel-series roots of $H_1^{(\infty)}(x,\boldsymbol{u})$, by renaming $H_1^{(\infty)}$, λ_1 and $\bar{\alpha}_1$ to H, λ and $\bar{\alpha}$, respectively. The Tschirnhaus transformation for $H(x,\boldsymbol{u})$ now is

$$H(x,\boldsymbol{u}) \mapsto \check{H}(x,\boldsymbol{u}) = H(x+\bar{\alpha},\boldsymbol{u}). \qquad (4.13)$$

For simplicity, we assume that the Newton polynomial for $\check{H}(x,\boldsymbol{u})$ is squarefree; if not so then we separate the non-squarefree factor again.

Let the slope of Newton line for $\check{H}(x,\boldsymbol{u})$ be $-\check{\lambda}$, and the roots of the Newton polynomial for $\check{H}(x,\boldsymbol{u})$ be $\check{\alpha}_1(\boldsymbol{u}),\dots,\check{\alpha}_m(\boldsymbol{u})$. It is obvious that $\check{\lambda} > \lambda$.

Lemma 5. *Each root $\check{\alpha}_i(\boldsymbol{u})$ $(1 \le i \le m)$ is such that $\|\check{\alpha}_i(\boldsymbol{u})\| = O(\|\boldsymbol{u}\|^{\check{\lambda}})$ for generic $\boldsymbol{u}$. Furthermore, $\check{\alpha}_i(\boldsymbol{u})$ may diverge on the zero-points of* $\mathrm{res}(H_0, H_0')$ *and $\bar{\alpha}_1 - \bar{\alpha}_j$ $(j \ge 2)$.*

Proof The first claim is obvious because the Newton polynomial for $\check{H}(x,\boldsymbol{u})$ is $(\check{\lambda},1)$-homogeneous. Since $\mathrm{res}(H_0, H_0')$ appears in the denominators of the terms of $\check{H}(x,\boldsymbol{u})$ except for term x^m, it may appear in some denominators of the Newton polynomial for $\check{H}(x,\boldsymbol{u})$. The same is true for the EHC in (3.16). ♢

Theorem 8. *If the Newton polynomial for $\check{H}(x,\boldsymbol{u})$ is squarefree and has no close root, then m Hensel-series $\phi_i^{(\infty)}$ $(i=1,\dots,m)$ of $H(x,\boldsymbol{u})$ are expressed as*

$$\phi_i^{(\infty)}(\boldsymbol{u}) \;=\; \bar{\alpha}(\boldsymbol{u}) + \check{\alpha}_i(\boldsymbol{u}) + \sum_{k=1}^{\infty} \delta\phi^{(k)}(\boldsymbol{u}). \qquad (4.14)$$

where $\delta\phi^{(k)}$ is the k-th order term which is computed just similarly as in **3.1**. *The divergence domains of $\phi_i^{(\infty)}(\boldsymbol{u})$ near the expansion point start from the expansion point and run outside radially along the zero-points of $\bar{\alpha}_i(\boldsymbol{u}) - \bar{\alpha}_j(\boldsymbol{u})$ $(j \neq i)$. The ratio defined by (4.6), showing the spread of the divergence domain, is specified by $\|\boldsymbol{u}\|^{\lambda}$ (not by $\|\boldsymbol{u}\|^{\check{\lambda}}$).*

Proof Eq. (4.14) is obvious from the transformation in (4.13). Lemma 4 tells us that $H(x,\boldsymbol{u})$ (= the Hensel factor $H_1^{(\infty)}(x,\boldsymbol{u})$) is in $\mathbb{C}\{(\boldsymbol{u})\}[x]$ and the coefficients of higher terms contain $\mathrm{res}(H_0, H_0')$ and its powers in the denominators. Thus, the spread of divergence domain of $\phi_i^{(\infty)}(\boldsymbol{u})$ is specified by both $\|\boldsymbol{u}\|^{\lambda}$ and $\|\boldsymbol{u}\|^{\check{\lambda}}$. Since $\lambda < \check{\lambda}$, the spread is specified mostly by $\|\boldsymbol{u}\|^{\lambda}$ near the expansion point. ♢

Example 1 (fine structure of Hensel series)

$$F(x,u,v) = (x^2-2ux+u^2)\,(x-v) + u^4+v^4.$$

The Newton polynomial for $F(x,u,v)$ is $F_{\mathrm{New}} = F_1^{(0)}F_2^{(0)}$, where $F_1^{(0)} = (x-u)^2$ and $F_2^{(0)} = x-v$, hence $\bar{\alpha} = u$. We put $D = u-v$ and $F_{\mathrm{h}} = u^4+v^4$. Performing the EHC of $F(x,u,v)$ with initial factors $F_1^{(0)}$ and $F_2^{(0)}$, up to order 5, we obtain

$$\begin{aligned} F_1^{(5)} = \;& (x-u)^2 - (x-2u+v)F_{\mathrm{h}}/D^2 + (2x-3u+v)F_{\mathrm{h}}^2/D^5 \\ & + \cdots - (143x-198u+55v)F_{\mathrm{h}}^5/D^{14}, \\ F_2^{(5)} = \;& (x-v) + F_{\mathrm{h}}/D^2 - 2F_{\mathrm{h}}^2/D^5 + 7F_{\mathrm{h}}^3/D^8 \\ & - 30F_{\mathrm{h}}^4/D^{11} + 143F_{\mathrm{h}}^5/D^{14}. \end{aligned}$$

Applying the transformation $x \mapsto x+u$ to $H(x,u,v) \overset{\text{def}}{=} F_1^{(5)}(x,u,v)$, we obtain

$$\begin{aligned} \check{H}(x,u,v) = \;& x^2 - (x-u+v)F_{\mathrm{h}}/D^2 + (2x-u+v)F_{\mathrm{h}}^2/D^5 \\ & + \cdots - (143x-55u+55v)F_{\mathrm{h}}^5/D^{14}. \end{aligned}$$

Note that D appears as denominator factor in $\check{H}(x,u,v)$. The Newton polynomial for $\check{H}(x,u,v)$ and its roots are

$$x^2 + (u-v)F_{\mathrm{h}}/D^2 = (x-\check{\alpha})(x+\check{\alpha}), \quad \check{\alpha} = \sqrt{(u^4+v^4)/(v-u)}.$$

Applying the EHC, we can compute the Hensel-series roots $\check{\phi}_{\pm}^{(5)}(u,v)$ of $\check{H}(x,u,v)$, as follows.

$$\begin{aligned} \check{\phi}_{\pm}^{(5)}(u,v) \;=\; & \frac{u^4+v^4}{2(u-v)^2} - \frac{(u^4+v^4)^2}{(u-v)^5} + \frac{7(u^4+v^4)^3}{2(u-v)^8} \\ & \pm \check{\alpha}\left(1 - \frac{5(u^4+v^4)}{8(u-v)^3} + \frac{231(u^4+v^4)^2}{128(u-v)^6}\right). \end{aligned}$$

We put $\check{\alpha}_{\pm} = \pm\check{\alpha}$. We have $\mathrm{res}(F_1^{(0)}, F_2^{(0)}) = (u-v)^2$, hence Theorem 8 tell us that the divergence domain of $\check{\phi}_{\pm}^{(k)}(u,v)$

runs along the line $u = v$, where three branches of the algebraic function become equal as $u=v \to 0$.

We check the convergence/divergence domains of $\check{\phi}_{\pm}^{(5)}(u,v)$. Since we have now no formula of convergence domain, we compute the following domains $\mathcal{C}$ and $\mathcal{D}$ for simulating the convergence and divergence domains, respectively.

$$\begin{aligned}
\mathcal{C}_T^{\lambda} &\overset{\text{def}}{=} \{\, (u,v) \in \mathbb{R}^2 \mid d(u,v)/(|u|^2+|v|^2)^{\lambda/2} < T \,\}, \\
&\qquad d(u,v) \overset{\text{def}}{=} \min_{i=1,2,3}\{|\bar{\alpha} + \check{\phi}_{+}^{(5)}(u,v) - \varphi_i(u,v)|\}, \\
\mathcal{D}_T^{\check{\lambda}} &\overset{\text{def}}{=} \{\, (u,v) \in \mathbb{R}^2 \mid |\check{\phi}_{+}^{(5)}(u,v)|/(|u|^2+|v|^2)^{\check{\lambda}/2} > T \,\}.
\end{aligned} \tag{4.15}$$

Here, T is a positive number denoting the threshold, $\varphi_i(u,v)$ is a branch of the algebraic function defined by $F(x,u,v)$, and $\lambda = 1$ and $\check{\lambda} = 3/2$ in this example. Since $\bar{\alpha} = O(\|\boldsymbol{u}\|^1)$ and $\check{\alpha}_{\pm} = O(\|\boldsymbol{u}\|^{3/2})$, we divide $d(u,v)$ and $|\check{\phi}_{+}^{(5)}(u,v)|$ by N and $N^{3/2}$, respectively, where $N = (|u|^2+|v|^2)^{1/2}$. We have computed $\varphi_1(\boldsymbol{u}), \dots, \varphi_3(\boldsymbol{u})$ exactly by Mathematica.

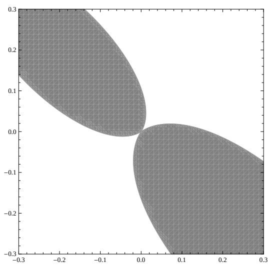


Fig. 2a Domain $\mathcal{C}_{0.01}^{1}$

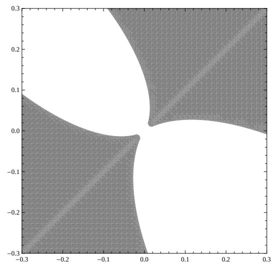


Fig. 2b Domain $\mathcal{D}_{0.1}^{3/2}$

Fig. 2a shows $\mathcal{C}_{0.01}^{1}$ (gray area), $-0.3 < u,v < 0.3$. We see that the divergence domain runs along $u-v = 0$ and it is very thin in a small neighborhood of the origin, as the theory predicts. Fig. 2b shows $\mathcal{D}_{0.1}^{3/2}$ (gray area), $-0.3 < u,v < 0.3$. (The thin line $u = v$ appears because $\check{\phi}_{+}^{(5)}$ diverges on the line). Noting that the gray area shows "divergence domain", we may say that both figures are consistent with each other.

$F(x,u,v)$ tells us that $F(x,u,v) = F_{\text{New}}(x,u,v)$ on the lines determined by $u^4+v^4 = 0$. So, we compare numerically $u+\check{\phi}_{+}^{(5)}(u,v)$ and $\varphi_i(u,v)$ by transforming (u,v) to (u',v') as $(u,v) = (\exp(i\pi/8)u', \exp(-i\pi/8)v')$. Table I shows the comparison, where we set $v' = 0.05$ and $u' \in \{-0.20, -0.15, -0.10, -0.05, 0.00, 0.05, 0.10, 0.15, 0.20\}$. We see that the Hensel series with Tschirnhaus transformation approximates the algebraic function pretty well in the region where we have investigated.

Table I: Numerical comparison of $u+\check{\phi}_{+}^{(5)}$ and φ_i where $(u,v) = (\exp(i\pi/8)u', 0.05\exp(-i\pi/8))$

u'	$u+\check{\phi}_{+}^{(5)}(u,v))$	$\varphi_1$or$\varphi_2$or$\varphi_3(u,v)$
-0.20	$-0.11719 - 0.00256\,i$	$-0.11862 - 0.00406\,i$
-0.15	$-0.09730 - 0.01490\,i$	$-0.09746 - 0.01511\,i$
-0.10	$-0.07259 - 0.01833\,i$	$-0.07259 - 0.01834\,i$
-0.05	$-0.16864 + 0.46786\,i$	$- - - -$
0.00	$+0.01043 - 0.00757\,i$	$+0.01043 - 0.00759\,i$
$+0.05$	$+0.04619 + 0.01913\,i$	$+0.04619 + 0.01913\,i$
$+0.10$	$+0.10944 + 0.00937\,i$	$+0.11250 + 0.00822\,i$
$+0.15$	$+0.17599 + 0.00132\,i$	$+0.17686 + 0.00974\,i$
$+0.20$	$+0.25453 - 0.00440\,i$	$+0.24339 + 0.00897\,i$

The results in this section may be summarized as follows.

1. The Hensel series are classified by Newton polynomials $F_{\mathcal{L}_1}, \dots, F_{\mathcal{L}_q}$; see Fig.1. The Hensel series $\phi_{ij}^{(\infty)}$ $(j = 1, \dots, \deg(F_{\mathcal{L}_i}))$ arising from $F_{\mathcal{L}_i}(x,\boldsymbol{u})$ are characterized as $\|\phi_{ij}^{(\infty)}\| = O(\|\boldsymbol{u}\|^{\lambda_i})$ for generic $\|\boldsymbol{u}\| \ll 1$, where $-\lambda_i$ is the slope of the Newton line $\mathcal{L}_i$.

2. When the Newton polynomial is not squarefree but contains m multiple root $\bar{\alpha}$, the corresponding m Hensel series are characterized by the second dominant terms $\check{\alpha}_1, \dots, \check{\alpha}_m$ which are the roots of reconstructed Newton polynomial. Divergence domains of m Hensel series are nearly the same; they start the expansion point and run radially along the zero-points of $\check{\alpha}_1 - \check{\alpha}_j$ $(j \geq 2)$, and their spreads are specified by the slope of Newton line before the Tschirnhaus transformation.

3. The most part of small neighborhood of the expansion point is occupied by the convergence domain.

5. HENSEL SERIES AROUND THE RAMIFICATION POINT

If $F(x,\boldsymbol{u})$ is irreducible over $\mathbb{C}$ then the algebraic function defined by $F(x,\boldsymbol{u})$ has mutually conjugate n branches, hence it is n-valued. On the other hand, if the Newton polynomial is factored as $F_{\text{New}} = F_{\text{New},1}F_{\text{New},2}$, for example, with $F_{\text{New},1}$ and $F_{\text{New},2}$ irreducible in $\mathbb{C}[x,\boldsymbol{u}]$ and $\deg(F_{\text{New},1}) = m < n$, then the Hensel series are divided into two classes, m mutually conjugate series of m-valuedness and n–m mutually conjugate ones of $(n-m)$-valuedness; see [17]. Then, one may think that there occurs discrepancy between algebraic functions and Hensel series on many-valuedness. Actually, no discrepancy seems to occur: the Hensel series often jumps from one branch of algebraic function to another when it passes a divergence domain, removing the discrepancy.

The jumping has been studied considerably in [15, 16], however the jumping is still a mysterious phenomenon, in particular, around ramification points. In this section, we investigate the jumping near the ramification point. For simplicity, we assume that the leading coefficient has no zero-point near the expansion point.

We first show an example of jumping; the next example is the same as that given in [15], where we could not explain how the jumping occurs actually.

Example 2 (jumping near the ramification point)

$$F(x,u,v) = (x-u)(x^2-2ux+v^2) + u^4+v^4.$$

The Newton polynomial is $F_{\text{New}} = (x-u)(x^2-2ux+v^2)$, and we have one rational Hensel series $\phi_1^{(k)}(u,v)$ with initial term $\alpha_1 = u$ and two algebraic Hensel series $\phi_{\pm}^{(k)}(u,v)$ with initial terms $\alpha_{\pm} = u \pm \sqrt{u^2-v^2}$:

$$\begin{aligned}
\phi_1^{(\infty)} &= u + \frac{u^4+v^4}{u^2-v^2} + \frac{(u^4+v^4)^3}{(u^2-v^2)^4} + \cdots, \\
\phi_{\pm}^{(\infty)} &= u - \frac{u^4+v^4}{2(u^2-v^2)} - \cdots \pm \xi \Big[1 - \frac{3(u^4+v^4)^2}{8(u^2-v^2)^3} - \cdots\Big], \\
&\qquad \text{where} \quad \xi = \sqrt{u^2-v^2}.
\end{aligned}$$

We have $\operatorname{res}((x-u),(x^2-2ux+v^2)) = v^2 - u^2$, hence the Hensel series diverges along the lines $u \pm v = 0$.

Let $\mathrm{C}_{0.1} : (u, v) = 0.1 \times (\cos t, \sin t)$ be a path on which we trace the real parts of two Hensel series and three branches of the algebraic function. In each of the following figures, black curves show a Hensel series and three gray curves show three algebraic functions. We change the parameter t in the range $0 \le t \le 2\pi$.

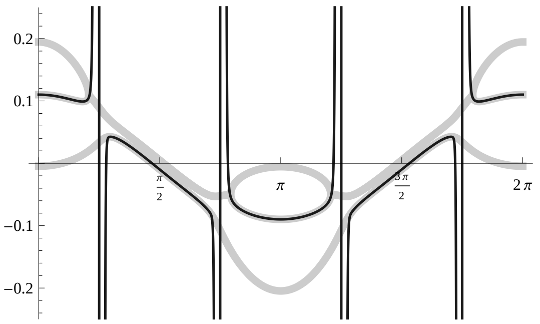


Fig. 3a $\mathcal{R}e(\phi_1^{(10)})$ traced along $\mathrm{C}_{0.1}$

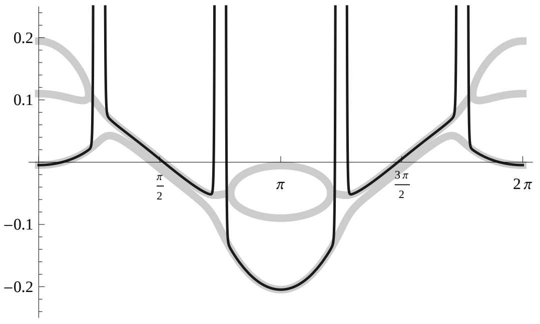


Fig. 3b $\mathcal{R}e(\phi_-^{(10)})$ traced along $\mathrm{C}_{0.1}$

Observe that the lower gray curve is separated from upper two. Hence, it is clear that the jumping occurs actually. The gray curves show that the algebraic function on $\mathrm{C}_{0.1}$ has a ramification point near the ramification points of algebraic Hensel series, and it seems to be quite difficult to know from which branch to another the Hensel series jumps. The figures suggest that two upper gray curves and lower gray one seem to correspond to $\phi_\pm^{(10)}$ and $\phi_1^{(10)}$, respectively, but it is wrong. Then, how the jumping occurs? ♢

A key idea to answer the above question is to deform the path along which we trace the Hensel series. **Observation A**: *the Hensel series is a continuous function in its convergence domain, and the algebraic function is also continuous except at the zero-points of the leading coefficient.* Therefore, by deforming the path so that it does not pass the divergence domain, we can know the correspondence between the Hensel series and the branches of algebraic function. Another useful idea is to choose the deformed path near the expansion point, where the theoretical analysis becomes simple because the Hensel series (and the algebraic function, too) can be approximated well by lower order terms of the Hensel series. Furthermore, we can choose the deformed path easily there so that it does not pass the divergence domain, because most area in the small neighborhood of the expansion point is the convergence domain.

In the neighborhood of the expansion point, we can find one-to-one correspondence among $\alpha_1, \dots, \alpha_n$ and $\varphi_1, \dots, \varphi_n$. So, let φ_i corresponds to α_i $(i = 1, \dots, n)$. The item 2 at the end of Sect. 4 says that any divergence domain near the expansion point runs along a line on which two roots of the Newton polynomial are equal. So, let $\mathcal{D}$ be a divergence domain running along the zero-points of $\alpha_i(\boldsymbol{u}) - \alpha_j(\boldsymbol{u})$ $(i \ne j)$ near the expansion point. Let C be a path passing $\mathcal{D}$. We assume that path C passes only the domain $\mathcal{D}$. Let D be a path which is obtained by deforming C so that it does not pass $\mathcal{D}$. Since we assumed that the leading coefficient has no zero-point around the expansion point, we have three cases:
case 1) $\alpha_i(\boldsymbol{u})$ and $\alpha_j(\boldsymbol{u})$ cross each other,
case 2) $\alpha_i(\boldsymbol{u})$ and $\alpha_j(\boldsymbol{u})$ are tangent to each other, and
case 3) $\alpha_i(\boldsymbol{u})$ and $\alpha_j(\boldsymbol{u})$ are conjugate to each other.

What happens when we trace a Hensel series along path C ? The above Observation A leads us to the followings. Case 1): Jumping occurs between φ_i and φ_j. The reason is as follows. When we are on path D, there is one-to-one correspondence between the Hensel series and the n branches: $\phi_i^{(k)} \Leftrightarrow \varphi_i$ $(i-1, \dots, n)$. Consider that we trace the branch φ_i along path C. Since $\varphi_i(\boldsymbol{u})$ is continuous even in $\mathcal{D}$, we are always on φ_i during this tracing. On the other hand, φ_i and φ_j cross in $\mathcal{D}$. Hence, if we look φ_i and φ_j on the path D, they look as if they represent a single Hensel series. In other words, if we think that we are tracing the same Hensel series, we must regard that the branch φ_i is renamed to be φ_j after passing $\mathcal{D}$. By the same reason, we have the next. Case 2): Jumping does not occur.

In case 3), the situation is complicated because, at the ramification point $\boldsymbol{u}_0$, we have $\alpha_i(\boldsymbol{u}_0) = \alpha_{j_1}(\boldsymbol{u}_0) = \cdots = \alpha_{j_m}(\boldsymbol{u}_0)$, where $\alpha_{j_1}, \dots, \alpha_{j_m}$ are mutually conjugate roots of the Newton polynomial. For example, in Example 2, we have $\alpha_+ - \alpha_1 = \sqrt{u^2 - v^2} = -(\alpha_- - \alpha_1)$. Observation A tells us that, in this case too, jumping will occur to a branch which is closest to $\phi_1^{(k)}$ after passing $\mathcal{D}$. We can determine which branch is the closest by the numerical comparison of $\phi_1^{(k)}(\boldsymbol{u})$ and $\varphi_i(\boldsymbol{u})$ $(i = 1, \dots, m)$. In Example 2, $\phi_1^{(k)}$ is real on path $\mathrm{C}_{0.1}$ while only one branch φ_3 is real in the range $\pi/4 \lesssim t \lesssim 3\pi/4$ of the path. Therefore, $\phi_1^{(k)}$ jumps to φ_3 after passing $\mathcal{D}$.

So far, we have displayed the convergence domain and jumping for curves which are rather smooth. The algebraic function changes rapidly near the ramification point: two real curves become complex conjugate just after passing the point. So, one may wonder: does the Hensel series approximate algebraic function well around the ramification point? Example 3 below will answer to this question.

Example 3 (Hensel series around ramification point)
We consider $F(x, \boldsymbol{u})$ given in Example 2, and show behaviors of the Hensel series and three branches of the algebraic function, by tracing them on the following path C_s:

$$\mathrm{C}_s : (u, v) = 0.1 \times (\cos t, si + \sin t), \quad 0 \le t \le 2\pi.$$

Figs. 4a, 4b, 4c and 4d (in the next page) show $\mathcal{R}e(\phi_i^{(10)})$, $i \in \{1, +, -\}$, traced along four circles $\mathrm{C}_{0.05}$, $\mathrm{C}_{0.10}$, $\mathrm{C}_{0.15}$ and $\mathrm{C}_{0.20}$, respectively. In each figure, black curve shows $\phi_1^{(10)}$ and gray curves show $\phi_\pm^{(10)}$. Figs. 5a, 5b, 5c and 5d show $\mathcal{R}e(\varphi_i)$, $i \in \{1, 2, 3\}$, traced along four circles $\mathrm{C}_{0.05}$, $\mathrm{C}_{0.10}$, $\mathrm{C}_{0.15}$ and $\mathrm{C}_{0.20}$, respectively. In each figure, black curve shows $\phi_1^{(10)}$ and gray curves show φ_i $(i = 1, 2, 3)$. We see that three Hensel series approximate three branches of the algebraic function pretty well. ♢

$\phi_i^{(10)}(\boldsymbol{u})$ and $\varphi_i(\boldsymbol{u})$ near ramification points

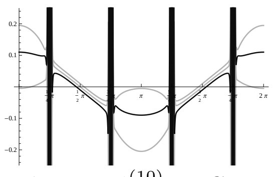


Fig. 4a $\phi_i^{(10)}$ on $C_{0.05}$

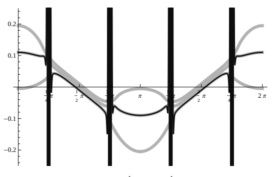


Fig. 5a $\varphi_i(u,v)$ on $C_{0.05}$

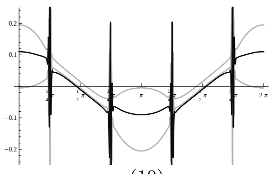


Fig. 4b $\phi_i^{(10)}$ on $C_{0.10}$

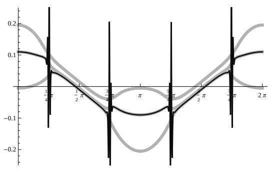


Fig. 5b $\varphi_i(u,v)$ on $C_{0.10}$

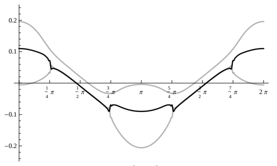


Fig. 4c $\phi_i^{(10)}$ on $C_{0.15}$

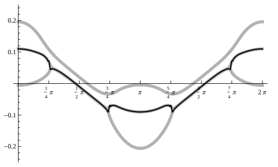


Fig. 5c $\varphi_i(u,v)$ on $C_{0.15}$

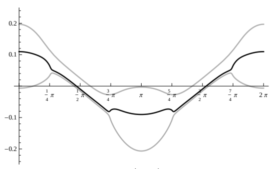


Fig. 4d $\phi_i^{(10)}$ on $C_{0.20}$

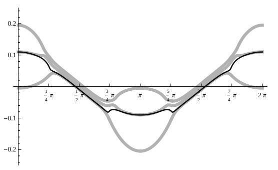


Fig. 5d $\varphi_i(u,v)$ on $C_{0.20}$

The results in this section may be summarized as follows.

1. The correspondence among Hensel series $\phi_1^{(k)}, \dots, \phi_n^{(k)}$ and branches $\varphi_1, \dots, \varphi_n$ of the algebraic function will change if we trace them by passing the divergence domain of the Hensel series. This looks as if the Hensel series jumps from one branch to another.

2. Although the algebraic function behaves complicatedly, the Hensel series approximate all the branches pretty well, in not only the region where algebraic function behaves smoothly but also the region around the ramification points.

6. REFERENCES

[1] F. Beringer and F. Jung, Multi-variate polynomials and Newton-Puiseux expansions. *Proc. SNSC 2001*, W. Winkler and U. Langer (Eds.): *Lec. Notes Comp. Sci.*, **2630**, 240–254, 2003.

[2] J. Della Dora and F. Richard-Jung, About the Newton algorithm for non-linear ordinary differential equation. *Proc. ISSAC'97*, W.W. Küchlin (Ed.), 298–304, ACM Press, 1997.

[3] A. Galligo and A. Poteaux, Continuations and monodromy on random Riemann surfaces. *Proc. SNC 2009*, H. Kai and H. Sekigawa (Eds.), 115–123, ACM Press, 2009.

[4] D. Inaba, Factorization of multivariate polynomials by extended Hensel construction. *ACM SIGSAM Bulletin*, **39**(1), 2–14, 2005.

[5] D. Inaba and T. Sasaki, A numerical study of extended Hensel series. *Proc. SNC 2007*, J. Verschede and S.T. Watt (Eds.), 103–109, ACM Press, 2007.

[6] M. Iwami, Analytic factorization of the multivariate polynomial. *Proc. CASC 2003*, V.G. Ganzha, E.W. Mayr and E.V. Vorozhtsov (Eds.), 213–225, Technishe Universität München Press, 2003.

[7] M. Iwami, Extension of expansion base algorithm to multivariate analytic factorization. *Proc. CASC 2004*, V.G. Ganzha, E.W. Mayr and E.V. Vorozhtsov (Eds.), 269–282, Technishe Universität München Press, 2004.

[8] T.-C. Kuo, Generalized Newton-Puiseux theory and Hensel's lemma in $\mathbf{C}[[x,y]]$. *Canad. J. Math.*, **XLI**, 1101-1116, 1989.

[9] S. McCallum, On testing a bivariate polynomial for analytic reducibility. *J. Symb. Comput.*, **24**, 509–535, 1997.

[10] J. McDonald, Fiber polytopes and fractional power series. *J. Pure Appl. Algebra*, **104**, 213–233, 1995.

[11] P.D. González Pérez, Singularités quasi-ordinaires toriques et polyèdre de Newton du discriminant. *Canad. J. Math.*, **52**(2), 348–368, 2000.

[12] A. Poteaux, Computing monodromy groups defined by plane algebraic curves. *Proc. SNC 2007*, J. Verschede and S.T. Watt (Eds.), 36–45, ACM Press, 2007.

[13] T. Sasaki and D. Inaba, Hensel construction of $F(x, u_1, \dots, u_\ell)$, $\ell \geq 2$, at a singular point and its applications. *ACM SIGSAM Bulletin*, **34**(1), 9–17, 2000.

[14] T. Sasaki and D. Inaba, Multivariate Hensel construction in roots. Preprint of Univ. Tsukuba, 2008. (submitted)

[15] T. Sasaki and D. Inaba. Convergence and many-valuedness of Hensel series near the expansion point. *Proc. SNC 2009*, H. Kai and H. Sekigawa (Eds.), 159–167, ACM Press, Aug. 2009.

[16] T. Sasaki and D. Inaba. Series expansion of multivariate algebraic functions at singular points – nonmonic case –. *Proc. ASCM 2009 (Asian Symposium on Computer Mathematics)*, 177–186, Kyushuu University, 2009.

[17] T. Sasaki and F. Kako, Solving multivariate algebraic equation by Hensel construction. Preprint of Univ. Tsukuba, March, 1993.

[18] T. Sasaki and F. Kako, Solving multivariate algebraic equation by Hensel construction. *Japan J. Indust. Appl. Math.*, **16**(2), 257–285, 1999.

[19] T. Sasaki and S. Yamaguchi, An analysis of cancellation error in multivariate Hensel construction with floating-point number arithmetic. *Proc. ISSAC'98*, O. Gloor (Ed.), 1–8, ACM Press, 1998.

[20] K. Shiihara and T. Sasaki, Analytic continuation and Riemann surface determination of algebraic functions by computer. *Japan J. Indust. Appl. Math.*, **13**(1), 107–116, 1996.

[21] R.J. Walker. *Algebraic Curves.* Springer-Verlag, New York-Heidelberg-Berlin, 1950.

A Regularization Method for Computing Approximate Invariants of Plane Curves Singularities

Mădălina Hodorog
Johann Radon Institute for Computational and Applied Mathematics
Austrian Academy of Sciences
Linz, Austria
madalina.hodorog@oeaw.ac.at

Josef Schicho
Johann Radon Institute for Computational and Applied Mathematics
Austrian Academy of Sciences
Linz, Austria
josef.schicho@oeaw.ac.at

ABSTRACT

We approach the algebraic problem of computing topological invariants for the singularities of a plane complex algebraic curve defined by a squarefree polynomial with inexactly-known coefficients. Consequently, we deal with an ill-posed problem in the sense that, tiny changes in the input data lead to dramatic modifications in the output solution.

We present a regularization method for handling the ill-posedness of the problem. For this purpose, we first design symbolic-numeric algorithms to extract structural information on the plane complex algebraic curve: (i) we compute the link of each singularity by numerical equation solving; (ii) we compute the Alexander polynomial of each link by using algorithms from computational geometry and combinatorial objects from knot theory; (iii) we derive a formula for the delta-invariant and the genus. We then prove the convergence for inexact data of the symbolic-numeric algorithms by using concepts from algebraic geometry and topology.

Moreover we perform several numerical experiments, which support the validity for the convergence statement.

Categories and Subject Descriptors

I.1.2 [**Symbolic and Algebraic Manipulation**]: algorithms—*algebraic algorithms*; G.4 [**Mathematics of Computing**]: Mathematical Software; G.1.2 [**Numerical Analysis**]: Approximation—*approximation of surfaces, piecewise polynomial approximation*; G.1.0 [**Numerical Analysis**]: General—*numerical algorithms, stability (and instability)*

General Terms

Algorithms, Design, Experimentation, Theory

Keywords

Plane curve singularity, ill-posed problem, regularization, symbolic-numeric algorithms, link of a singularity, Alexander polynomial, delta-invariant, genus

1. INTRODUCTION

In this paper, we treat the algebraic problem of computing topological invariants for each singularity of a plane complex algebraic curve defined by a squarefree polynomial with coefficients of limited accuracy, i.e. the coefficients are both exact and inexact data. The problem is ill-posed in the sense that tiny changes in the input data cause huge changes in the output solution. We employ an adapted regularization method based on [7, 18] to handle the ill-posedness of the problem. This regularization method allows us to construct approximate solutions to the ill-posed problem, which are stable under small changes in the initial data.

We first design symbolic-numeric algorithms for computing invariants for each singularity of a plane complex algebraic curve defined by a squarefree polynomial. We compute the link of each singularity by intersecting the curve with a sphere centered in the singularity and of a small radius, based on [4, 15]. The computation of the link of the singularity allows us to analyze the local topology of each singularity. We then compute the Alexander polynomial attached to the link of the singularity using algorithms from computational geometry [6] and combinatorial objects from knot theory, based on [5, 13]. The Alexander polynomial is a complete invariant for links of singularities, i.e. different links of singularities have different Alexander polynomials [22]. As applications, from the Alexander polynomial we derive formulas for the delta-invariant of each singularity and for the genus of the curve. In [2] a numerical method based on homotopy continuation for computing the genus of any one-dimensional irreducible component of an algebraic set is presented, while in [17] the authors provide a formula for the genus of an algebraic curve with all singularities affine and ordinary.

We implement the designed symbolic-numeric algorithms for invariants of plane curves singularities in the free library called GENOM3CK-GENus cOMputation of plane Complex algebraiC Curves using Knot theory-written in the free algebraic geometric modeler Axel [21] and in the free computer algebra system Mathemagix [11].

We sketch the proof for the convergence for inexact data property of the designed symbolic-numeric algorithms using concepts from algebraic geometry and topology. We perform several numerical experiments with the library GENOM3CK, which confirm the convergence for inexact data property.

We organize this paper as follows. In Section 2 we define the plane complex algebraic curves and their singularities. We also introduce invariants for each singularity of a plane complex algebraic curve: the link of each singularity, the Alexander polynomial attached to the link, and the delta-invariant of each singularity. In Section 3 we present the symbolic-numeric algorithms developed for the computation of the defined invariants. Section 4 contains regularization principles that we employ to handle the ill-posedness of the problem. We also sketch the proof for the convergence for inexact data property of the designed symbolic-numeric algorithms. In Section 5 we discuss implementation issues and we perform several test experiments. We give the conclusions in Section 6.

2. PLANE COMPLEX ALGEBRAIC CURVES

2.1 Singularities of Plane Complex Algebraic Curves

For our study, we define the (affine) plane complex algebraic curves following [19]:

Definition 1. Let $\mathbb{C}$ be the algebraically closed field of complex numbers, and $\mathbb{A}^2(\mathbb{C}) = \{(z, w) \in \mathbb{C}^2\}$ the affine complex plane. Let $p(z, w) \in \mathbb{C}[z, w]$ be an irreducible polynomial in z and w with coefficients in $\mathbb{C}$ of degree m. An affine plane algebraic curve over $\mathbb{C}$ of degree m defined by $p(z, w)$ is the set of zeroes of the polynomial $p(z, w)$, i.e.

$$\mathcal{C} = \{(z, w) \in \mathbb{A}^2(\mathbb{C}) | p(z, w) = 0\}.$$

We consider $\mathbb{P}^2(\mathbb{C}) = \{(z : w : u) | (z, w, u) \in \mathbb{C}^3 \backslash \{0, ..., 0\}\}$, with $(z : w : u) = \{(\alpha z, \alpha w, \alpha u) | \alpha \in \mathbb{C} \setminus \{0\}\}$, the projective plane over $\mathbb{C}$. We consider $p^*(z, w, u)$ the homogenized polynomial of $p(z, w)$ in u with $p^*(z, w, u) = p_m(z, w) + p_{m-1}(z, w)u + ... + p_0(z, w)u^m$.

We define the singular points of a plane complex algebraic curve in the following way:

Definition 2. Let $\mathcal{C}$ be a plane complex algebraic curve defined by the irreducible polynomial $p(z, w) \in \mathbb{C}[z, w]$. We denote by $\partial_z p := \partial p(z, w)/\partial z$ and by $\partial_w p := \partial p(z, w)/\partial w$ the partial derivatives of $p(z, w)$ with respect to z and w. The set of singular points (or singularities) of $\mathcal{C}$ is defined as

$$\begin{aligned} Sing(\mathcal{C}) = \{(z_0, w_0) \in \mathbb{A}^2(\mathbb{C}) | p(z_0, w_0) = \\ \partial_z p(z_0, w_0) = \partial_w p(z_0, w_0) = 0\}. \end{aligned}$$

The points of a plane complex algebraic curve that are not singular are called nonsingular or regular points. An irreducible plane complex algebraic curve has at most finitely many singular points, and if it has none it is called nonsingular (or smooth). For simplicity reasons we denote the affine complex plane by $\mathbb{C}^2$. Since $\mathbb{C}^2$ is isomorphic with $\mathbb{R}^4$, we consider a plane complex algebraic curve $\mathcal{C} \subset \mathbb{C}^2$ as a real two-dimensional object in $\mathbb{R}^4$. For visualization purposes, we cannot draw this object in $\mathbb{R}^4$, but we sketch the equivalent curve in $\mathbb{R}^2$.

An important observation is that computing the singularities of a plane complex algebraic curve is an ill-posed problem, in the sense that small changes in the defining polynomial of the curve lead to dramatic changes in the topology (shape) of the curve itself.

Example 1. In Figure 1 the red inner curve represents the topology of $\mathcal{C} = \{(z, w) \in \mathbb{R}^2 : z^3 + z^2 - w^3 = 0\}$, and the blue outer curve the topology of $\mathcal{D} = \{(z, w) \in \mathbb{R}^2 : z^3 + z^2 - w^3 - 0.1w^2 = 0\}$. The curves $\mathcal{C}$ and $\mathcal{D}$ have a singularity in the origin, i.e. $\mathcal{C}$ has a cusp in the origin and $\mathcal{D}$ has a double point in the origin. We notice that the singularity of $\mathcal{C}$ changes its type under small changes of the defining polynomial of $\mathcal{C}$ obtaining the singularity of $\mathcal{D}$.

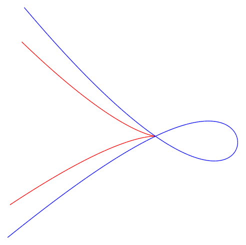


Figure 1: Example of ill-posedness of the singularity $(0, 0)$ of $z^3 + z^2 - w^3$. Picture produced with Axel, see Section 5 for more information.

2.2 Invariants of Plane Complex Algebraic Curves

First, we define an homeomorphism in the following way:

Definition 3. Two subsets $U \subset \mathbb{R}^k, V \subset \mathbb{R}^n$ are topologically equivalent or homeomorphic if there exists a bijective function $\varphi : U \rightarrow V$ such that both φ and its inverse are continuous. In this case, φ is called an homeomorphism.

A pair (X, A) of spaces is a topological space together with a subspace $A \subseteq X$. A mapping $\varphi : (X, A) \rightarrow (Y, B)$ of pairs is a continuous mapping $\varphi : X \rightarrow Y$ with $\varphi(A) \subseteq B$. A homeomorphism $\varphi : (X, A) \rightarrow (Y, B)$ of pairs is a mapping of pairs which is a homeomorphism $\varphi : X \rightarrow Y$ and induces a homeomorphism $\varphi/_A : A \rightarrow B$.

In this paper, the (topological) invariants of a plane complex algebraic curve $\mathcal{C}$ are those properties of $\mathcal{C}$ and its singularities that are unchanged under homeomorphism of small disks around 0 mapping the first curve onto the second curve.

We consider the stereographic projection from $\mathbb{R}^3$ to $\mathbb{R}^2$ as a mapping that projects a sphere onto a plane. It is constructed as follows: we take a sphere; we draw a line from the north pole N of the sphere to a point $\hat{P}$ in the equator plane to intersect the sphere at a point P. The stereographic projection of $\hat{P}$ is P. The stereographic projection gives an explicit homeomorphism from the unit sphere minus the north pole to the Euclidean plane.

For our study, we use the stereographic projection from $\mathbb{R}^4$ to $\mathbb{R}^3$ to project objects from $\mathbb{R}^4$ to $\mathbb{R}^3$ by preserving their topological properties.

Link of a Plane Curve Singularity

We introduce notions from knot theory, which are useful for the purpose of this paper. First, we define a knot and a link:

Definition 4. A knot is a piecewise linear or a differentiable simple closed curve in $\mathbb{R}^3$ and a link is a finite union of disjoint knots, see Figure 2. The knots that make up a link are called the components of the link, and thus a knot is a link with one component.

We define the equivalence of two links as follows:

Definition 5. We say that two links are equivalent if there exists an orientation-preserving homeomorphism on $\mathbb{R}^3$ that

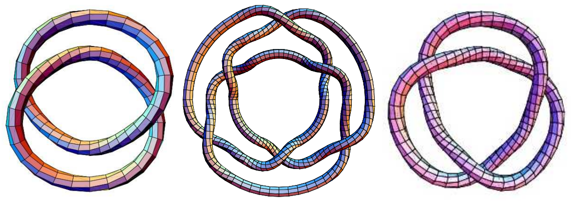


Figure 2: Examples of links/knot. From left to right: the Hopf link, the Borromean rings, the trefoil knot

maps one link onto the other. This equivalence is called (ambient) isotopy.

We introduce some preliminary notions: a polygonal curve P is a curve specified by a sequence of points $(p_1, p_2, ..., p_n)$ called its vertices such that the curve consists of the segments connecting the consecutive vertices. A polygonal curve is simple if each segment intersects exactly two other segments only at their endpoints. A polygonal curve is closed if the first vertex coincides with the last vertex. In this paper, we approximate knots by simple closed polygonal curves in $\mathbb{R}^3$, but we usually draw them as smooth curves that do not intersect themselves in $\mathbb{R}^3$. We use the following terminology: if we approximate a knot by the simple closed polygonal curve P represented by $(p_1, p_2, ..., p_n)$, then the points $(p_1, p_2, ..., p_n)$ of P are called the vertices of the knot and the segments of P are called the edges of the knot.

When we work with knots we actually work with their regular projections in $\mathbb{R}^2$. We consider that a regular projection of a knot is a linear projection for which no three points on the knot project to the same point, and no vertex projects to the same point as any other point on the knot. A crossing point is the image of two knot points of such a regular projection to $\mathbb{R}^2$.

For our study, we work with (knot) diagrams: (i) a diagram is the image under regular projection, together with the information on each crossing point telling which branch goes over and which goes under, see Figure 3. (ii) a diagram together with an arbitrary orientation of each knot in the link is called an oriented diagram.

We introduce the elements of an oriented diagram as follows: (i) a crossing is called lefthanded (denoted with -1) if the underpass traffic goes from left to right or it is called righthanded (denoted with $+1$) if the underpass traffic goes from right to left as indicated by the dotted round arrow in Figure 4; (ii) an arc is the part of a diagram between two undercrossings. Whether lefthanded or righthanded, each crossing is determined by three arcs and we denote the overgoing arc with i, and the undergoing arcs with j and k, as indicated in Figure 4. We notice that the number of arcs equals the number of crossings in a diagram, see Figure 5.

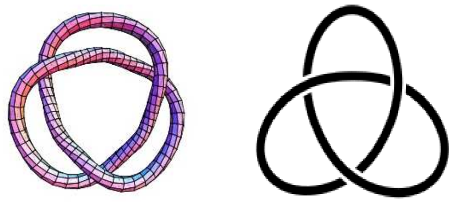


Figure 3: Example of a trefoil knot and its diagram

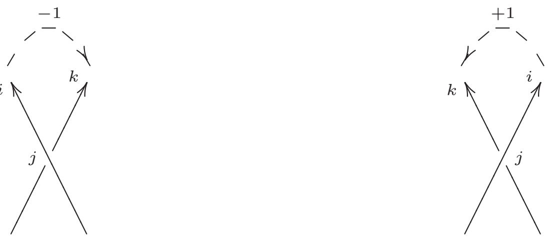


Figure 4: Types of crossings: lefthanded (-1) and righthanded (+1), together with the labels for the 3 arcs of a crossing

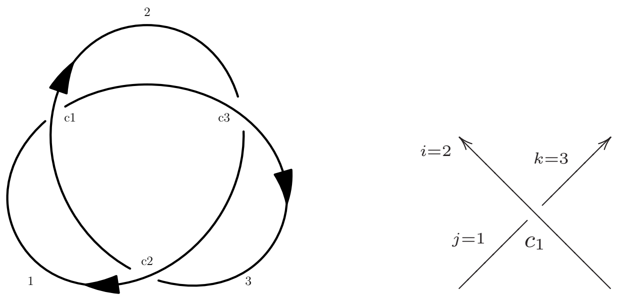


Figure 5: Oriented diagram of the trefoil knot with 3 arcs denoted with $\{1, 2, 3\}$ and 3 crossings denoted $\{c_1, c_2, c_3\}$. Example of arcs labeling for the crossing denoted c_1, which is lefthanded.

We employ the following theorem, which asserts that the equivalence class of a special type of link determines the homeomorphism type of the singularity:

Theorem 1. *(Milnor[15]) Let $V \subset \mathbb{C}^{n+1}$ be a hypersurface in $\mathbb{C}^{n+1}$, i.e. an algebraic variety defined by a single polynomial f. Assume $\vec{0} \in V$ and $\vec{0}$ is an isolated singularity, i.e. there is no other singularity on a sufficiently small neighborhood of $(0,0)$; S_ϵ is the sphere centered in $\vec{0}$ and of radius ϵ; and D_ϵ is the disk centered in $\vec{0}$ of radius ϵ. Then, for sufficiently small ϵ, $X_\epsilon = S_\epsilon \cap V$ is a $(2n-1)$-dimensional nonsingular set and the pair $(D_\epsilon, D_\epsilon \cap V)$ is homeomorphic to the pair consisting of the cone over S_ϵ and the cone over $X_\epsilon = S_\epsilon \cap V$.*

For the case $n = 1$, Milnor's theorem says that there exists $\epsilon_0 \in \mathbb{R}_{>0}$ such that for any $\epsilon_1, \epsilon_2 \in \mathbb{R}_{>0}$ with $\epsilon_1 < \epsilon_0$ and $\epsilon_2 < \epsilon_0$, the images of $X_{\epsilon_1} \subset S_{\epsilon_1}$ and $X_{\epsilon_2} \subset S_{\epsilon_2}$ through stereographic projection are links and they are equivalent, i.e. $D_{\epsilon_1} \cap \mathcal{C}$ and $D_{\epsilon_2} \cap \mathcal{C}$ are homeomorphic. In addition, for any $0 < \epsilon < \epsilon_0$ the image of X_ϵ through stereographic projection is called *the link L of the singularity* of f (or of $\mathcal{C}$) at $(0,0)$ and it is well-defined up to homeomorphism of pairs. In this case, the link defined as the image of $X_\epsilon \subset S_\epsilon$ through stereographic projection determines the topological type of the singularity $(0,0)$ of $\mathcal{C}$. In theory, a link is called algebraic if it is equivalent to the link of a plane curve singularity.

Under the same hypotheses from Theorem 1 and considering S^1 the unit circle, Milnor fibration theorem states that the mapping $\phi : S_\epsilon \setminus L \to S^1, \phi(z,w) = f(z,w)/|f(z,w)|$ is a fibration, i.e. the complement $S_\epsilon \setminus L$ is a union of smooth surfaces, each being the preimage of one point.

Alexander Polynomial of a Plane Curve Singularity

An important result of Yamamoto [22] says that the Alexander polynomial is a complete invariant for the algebraic links, i.e. the Alexander polynomial uniquely defines all the algebraic links up to an (ambient) isotopy. In this way, we can use the Alexander polynomial of the link of a singularity to distinguish the topological type of the singularity itself. In [9] we present a straightforward algorithm to compute the Alexander polynomial attached to the link of a singularity by using combinatorial objects from knot theory such as the diagram of the link, i.e. its arcs and its crossings. For introducing the Alexander polynomial, we need some preliminary definitions based on [13]:

Definition 6. Let $D(L)$ be an oriented link diagram with r components and n crossings $x_q : q \in \{1, ..., n\}$. We denote the arcs of $D(L)$ with the labels $\{1, ..., n\}$ and separately the crossings of $D(L)$ with $\{1, ..., n\}$. We denote the labeling matrix of $D(L)$ with $LM(L) \in \mathcal{M}(n, 4, \mathbb{Z})$. We define $LM(L) = (b_{ql})_{q,l}$ with $q \in \{1, ..., n\}, l \in \{1, ..., 4\}$ row by row for each crossing x_q as follows: (i) at b_{q1} store the type of the crossing x_q ($+1$ or -1); (ii) at b_{q2} store the label of the arc i of x_q in $D(L)$; (iii) at b_{q3} store the label of the arc j of x_q in $D(L)$; (iv) at b_{q4} store the label of the arc k of x_q in $D(L)$, see Figure 5 for an example.

Definition 7. Let $D(L)$ be an oriented link diagram with r components and n crossings $x_q : q \in \{1, ..., n\}$. We denote the arcs and the crossings of $D(L)$ as in Definition 6. We consider $LM(L)$ the labeling matrix of $D(L)$ as in Definition 6. We denote the prealexander matrix of L with $PM(L) \in \mathcal{M}(n, n, \mathbb{Z}[t_1, t_1, ..., t_r])$. If $D(L)$ has no crossings, then $PM(L) \in \mathcal{M}(0, 0, \emptyset)$, otherwise we define $PM(L)$ row by row for each crossing x_q depending on $LM(L)$. For x_q we consider the variable t_s, where $s \in \{1, ..., r\}$ is the s-th knot component of $D(L)$, which contains the overgoing arc that determines the crossing x_q. Then: (i) if x_q is righthanded, i.e. $b_{q1} = +1$ in $LM(L)$, then at position b_{q2} of $PM(L)$ store the label $1 - t_s$, at position b_{q3} store -1 and at position b_{q4} store t_s; (ii) if x_q is lefthanded, i.e. $b_{q1} = -1$ in $LM(L)$, then at position b_{q2} of $PM(L)$ store the label $1 - t_s$, at position b_{q3} store t_s and at position b_{q4} store -1; (iii) if two or all of the positions b_{q2}, b_{q3}, b_{q4} have the same value, then store the sum of the corresponding labels at the corresponding position. All other entries of the matrix are 0.

We define the Alexander polynomial of $D(L)$ depending on the number of knot components in L:

Definition 8. Let $D(L)$ be an oriented link diagram with r components and n crossings, $LM(L)$ be its labeling matrix as in Definition 6 and $PM(L)$ be its prealexander matrix as in Definition 7. Then: (i) the univariate Alexander polynomial [13] $\Delta_L(t_1) \in \mathbb{Z}[t_1^{\pm 1}]$ is the normalized polynomial computed as the determinant of any $(n-1) \times (n-1)$ minor of the prealexander matrix of $D(L)$. A normalized polynomial is a polynomial in which the term of the lowest degree is a positive constant; (ii) the multivariate Alexander polynomial [5] $\Delta_L(t_1, ..., t_r) \in \mathbb{Z}[t_1^{\pm 1}, ..., t_r^{\pm 1}]$ is the normalized polynomial computed as the greatest common divisor of all the $(n-1) \times (n-1)$ minor determinants of the prealexander matrix of $D(L)$. If $PM(L) \in \mathcal{M}(0, 0, \emptyset)$, then we define $\Delta_L(t_1) = 1$ and $\Delta_L(t_1, ..., t_r) = 0$.

In Definition 8, the univariate polynomial computed as the determinant of any $(n-1) \times (n-1)$ minor of the prealexander matrix of $D(L)$ depends on the choice of the original diagram $D(L)$ of a knot and its labelings. Alexander's result [1] is that although the choice of the original diagram of a knot and its labelings may produce different polynomials, any of them will differ by a multiple of $\pm t_1^k$, for some integer k. Thus, if we normalize the polynomial to have a positive constant term, the resulting Alexander polynomial will be a knot invariant. A similar argument follows from [5] for the multivariate polynomial.

Delta-Invariant of a Plane Curve Singularity

From the Alexander polynomial we derive a formula for the delta-invariant of the singularity of a plane complex algebraic curve in the following way:

Definition 9. (based on Milnor[15]) Let $\Delta_L(t_1, \ldots, t_r)$ be the Alexander polynomial of the link of the isolated singularity $P = (0, 0)$ of a plane complex algebraic curve. Let r be the number of variables in Δ_L and let μ be the degree of Δ_L. If $r = 1$, then the delta-invariant of P is computed as $\delta_P = \mu/2$, otherwise $\delta_P = (\mu + r)/2$.

We can derive a formula for the genus of a plane complex algebraic curve as described in [15]:

Definition 10. Let $\mathcal{C}$ be a plane complex algebraic curve in the projective plane as introduced in [20]. We denote by $Sing(\mathcal{C})$ the singularities of $\mathcal{C}$, and by $\delta_P \in \mathbb{N}$ the delta-invariant of the singularity P. The genus of $\mathcal{C}$, $genus(C) \in \mathbb{Z}$, is defined as: $genus(\mathcal{C}) = ((m-1)(m-2))/2 - \sum_{P \in Sing(\mathcal{C})} \delta_P$.

Approximate Invariants of a Plane Curve Singularity

We have previously introduced several invariants for a plane complex algebraic curve $\mathcal{C}$ with an isolated singularity, i.e. the Alexander polynomial attached to the link of the singularity, the delta-invariant of the singularity and the genus of the curve. We notice that the computation of these invariants is conditioned by the computation of the image of X_ϵ through stereographic projection, which is the *link L of the singularity* and which depends on the parameter $\epsilon \in \mathbb{R}_+$.

Hence we are motivated to define the *ϵ-invariants* of a plane complex algebraic curve with an isolated singularity, which depend on a parameter $\epsilon \in \mathbb{R}_{>0}$:

Definition 11. Let $\mathcal{C}$ be a plane complex algebraic curve defined by the squarefree polynomial $p(z, w) \in \mathbb{C}[z, w]$. Let $P = (z_0, w_0) \in \mathbb{C}^2$ be an isolated singularity of $\mathcal{C}$ and let $S_\epsilon(P) = \{(z, w) \in \mathbb{C}^2 : |z - z_0|^2 + |w - w_0|^2 = \epsilon^2\}$ be the sphere centered in P of radius $\epsilon \in \mathbb{R}_{>0}$. We take $Y = \mathcal{C} \cap S_\epsilon(P)$. We consider $\pi_{(\epsilon,N)}$ the stereographic projection of the sphere $S_\epsilon(P)$ from its north pole N, which does not belong to $\mathcal{C}$ and which is defined as:

$$\begin{array}{c} \pi_{(\epsilon,N)} : S_\epsilon \setminus \{N\} \subset \mathbb{R}^4 \to \mathbb{R}^3 \\ (a, b, c, d) \to (x, y, z) = (\frac{a}{\epsilon - d}, \frac{b}{\epsilon - d}, \frac{c}{\epsilon - d}) \end{array} . \qquad (1)$$

If $\pi_{(\epsilon,N)}(Y)$ has no singularities, then:

- we call $L_\epsilon := \pi_{(\epsilon,N)}(Y)$ the *ϵ-link of the singularity* of $p(z, w)$ (or of $\mathcal{C}$) at P. We call L_ϵ an ϵ-algebraic link.
- we define the *ϵ-Alexander polynomial* of $\mathcal{C}$ at P as the Alexander polynomial of L_ϵ.
- we define the *ϵ-delta-invariant* of P as the delta-invariant of the ϵ-Alexander polynomial of $\mathcal{C}$ at P.

3. SYMBOLIC-NUMERIC ALGORITHMS FOR INVARIANTS OF PLANE CURVE SINGULARITIES

We shortly describe the symbolic-numeric algorithms we design for computing the ϵ-invariants of a plane complex algebraic curve as introduced in Subsection 2.2. For more information on these algorithms see [9, 10].

Problem 1. Given the following: (i) a squarefree polynomial $p(z,w) \in \mathbb{C}[z,w]$ that defines a plane complex algebraic curve $\mathcal{C} \subset \mathbb{C}^2$; (ii) a parameter $\epsilon \in \mathbb{R}_{>0}$ that determines the sphere S_ϵ centered in the origin $(0,0)$ of radius ϵ.
our goal is: (1) to compute the singularities of $\mathcal{C}$ in $\mathbb{C}^2$; (2) to compute a set of ϵ-invariants of $\mathcal{C}$, i.e. the ϵ-algebraic link, the ϵ-Alexander polynomial, the ϵ-delta-invariant as introduced in Definition 11;

We describe an algorithm for computing the set of singularities $Sing(\mathcal{C})$ of the plane complex algebraic curve $\mathcal{C}$ of degree m defined by a squarefree polynomial $p(z,w) \in \mathbb{C}[z,w]$. From Definition 2, it follows that to compute $Sing(\mathcal{C})$ we need to solve the following overdeterminate system:

$$p(z_0,w_0) = \partial_z p(z_0,w_0) = \partial_w p(z_0,w_0) = 0. \qquad (2)$$

We compute the roots of the system (2) in $\mathbb{R}^2$ with subdivision methods [16]. These methods take as input the polynomials defining the system (2), a box $B = [-a,a]\times[-b,b] \subset \mathbb{R}^2$, and a positive real number δ. The box B has to be big enough to contain all the roots of (2). The output of the subdivision methods is a set of boxes S in $\mathbb{R}^2$ smaller than δ, which contains all the roots of (2), and a set M containing the middle points of all the boxes from S.

We compute the real singularities of the plane algebraic curve defined by the squarefree polynomial $p(z,w) \in \mathbb{C}[z,w]$ in the projective space by homogenizing and dehomogenizing the polynomial $p(z,w)$ w.r.t different variables and making sure not to return solutions in the overlaps twice. We consider the projective plane over the real numbers $\mathbb{P}^2(\mathbb{R}) = U_z \cup U_w \cup U_u$, where U_z, U_w, U_u are homeomorphic to $\mathbb{R}^2$ and $U_z = \{(1 : \frac{w}{z} : \frac{u}{z})\}, U_w = \{(\frac{z}{w} : 1 : \frac{u}{w})\}, U_u = \{(\frac{z}{u} : \frac{w}{u} : 1)\}$. Without loss of generality we assume $|u| \geq |z|, |w|$. We notice that any point from $\mathbb{P}^2(\mathbb{R})$ is in B_z, B_w, B_u, where $B_z \subseteq U_z, B_w \subseteq U_w, B_u \subseteq U_u$ and $B_z = B_w = B_u = [-1,1]\times[-1,1]$. Thus using subdivision methods, we compute a list of δ-boxes S in $B = [-1,1]\times[-1,1] \subset \mathbb{R}^2$ smaller than a given tolerance δ, and a list M of middle points for all the boxes in S with two properties: (1) each real singularity of the plane algebraic curve is contained in one of the δ-boxes from S; (2) the value of p and its first derivatives in each point from M are small.

In the same way, we can use subdivision methods to find the complex singularities of $\mathcal{C}$ in the projective plane: we consider $z = z_1 + iz_2, w = w_1 + iw_2, u = u_1 + iu_2$ and the projective plane over the complex numbers $\mathbb{P}^2(\mathbb{C}) = U_z \cup U_w \cup U_u$, where U_z, U_w, U_u are homeomorphic to $\mathbb{C}^2$ and $U_z = \{(1 : \frac{w}{z} : \frac{u}{z})\}, U_w = \{(\frac{z}{w} : 1 : \frac{u}{w})\}, U_u = \{(\frac{z}{u} : \frac{w}{u} : 1)\}$. We assume $|z_1| \geq |z_2|, |w_1|, |w_2|, |u_1|, |u_2|$ and we show that any point from $\mathbb{P}^2(\mathbb{C})$ is in B_z, B_w, B_u, where $B_z \subseteq U_z$ and $B_z = \{(w,u) \in \mathbb{C}^2 | w_1, w_2, u_1, u_2 \in [-1,1]\}$ (we obtain equivalent formula for B_w, B_u). For instance, we consider $U_z = \{(1 : \frac{w}{z} : \frac{u}{z})\}$ and $Re(\frac{w}{z})$ the real part of the complex number $\frac{w}{z}$. We rewrite $Re(\frac{w}{z}) = Re\left(\frac{w_1 + iw_2}{z_1 + iz_2}\right) = Re\left(\frac{(z_1 - iz_2)(w_1 + iw_2)}{z_1^2 + z_2^2}\right) = \frac{z_1 w_1 + z_2 w_2}{z_1^2 + z_2^2}$. We get:

$$|Re(\frac{w}{z})| = \left|\frac{z_1 w_1 + z_2 w_2}{z_1^2 + z_2^2}\right| \leq \frac{|z_1||z_1| + |z_1||z_1|}{2|z_1|^2} = \frac{2|z_1|^2}{2|z_1|^2} = 1.$$

In our implementation, we apply the subdivision methods for the real case. As discussed before, we can apply the subdivision methods to the complex case, but this is not available yet in our current implementation.

Algorithm 1 Singularities of an algebraic curve: SING$(f, p, \mathcal{C})$

Input: $p(z,w) \in \mathbb{C}[z,w]$ a squarefree polynomial,
m the degree of $p(z,w)$, $\delta \in \mathbb{R}_{>0}$ positive real number,
$\mathcal{C} = \{(z,w) \in \mathbb{C}^2 | p(z,w) = 0)\}$ a plane algebraic curve
Output: a list of points $M \subset \mathbb{R}^2$ such that for every $s \in Sing(\mathcal{C})$ there exists a unique $m \in M$ such that $d(m,s) \leq \delta$, where $Sing(\mathcal{C})$ is the set of real singularities of $\mathcal{C}$ in the projective plane

```
1. Homogenize $p(z,w)$ w.r.t. $u$ obtaining $p^*(z,w,u)$;
    (a) Dehomogenize $p_1(z,w) := p^*(z,w,1)$
    (b) Get $S_1$ by solving $p_1 = \partial_z p_1 = \partial_w p_1 = 0$ with subdivision methods.
    (c) Homogenize $S_1 = \{(z_0,w_0) \in \mathbb{R}^2\}$ to get $S_1' = \{(z_0 : w_0 : 1) \in \mathbb{P}^2(\mathbb{R})\}$.
    (d) Dehomogenize $p_2(w,u) := p^*(1,w,u)$
    (e) Get $S_2$ by solving $p_2 = \partial_w p_2 = \partial_u p_2 = u = 0$ with subdivision methods.
    (f) Homogenize $S_2 = \{(w_0,u_0) \in \mathbb{R}^2\}$ to get $S_2' = \{(1 : w_0 : u_0) \in \mathbb{P}^2(\mathbb{R})\}$.
    (g) Dehomogenize $p_3(z,u) := p^*(z,1,u)$
    (h) Get $S_3$ by solving $p_3 = \partial_z p_3 = \partial_u p_3 = z = u = 0$ with subdivision methods.
    (i) Homogenize $S_3 = \{(z_0,u_0) \in \mathbb{R}^2\}$ to get $S_3' = \{(z_0 : 1 : u_0) \in \mathbb{P}^2(\mathbb{R})\}$.
2. Return $Sing(\mathcal{C}) = S_1' \cup S_2' \cup S_3'$
```

Example 2. We consider $\mathcal{C}$ the plane complex algebraic curve defined by the squarefree polynomial $p(z,w) = z^2w + w^4 \in \mathbb{C}[x,y]$. We compute $Sing(\mathcal{C})$. We homogenize $p(z,w)$ w.r.t. the variable u obtaining $p^*(z,w,u) = z^2wu + w^4$. We replace $u = 1$ in p^* and obtain $p_1(z,w) = z^2w + w^4$. We solve the overdeterminate system $z_0^2 w_0 + w_0^4 = 2z_0 w_0 = z_0^2 + 4w_0^3 = 0$ and get $S_1 = \{(0,0)\}$. We homogenize S_1 and get $S_1' = \{(0:0:1)\}$.

We replace $z = 1$ in p^* and obtain $p_2(w,u) = wu + w^4$. Similarly as in (2) we solve the overdeterminate system $w_0 u_0 + w_0^4 = u_0 + 4w_0^3 = w_0 = u_0 = 0$ and obtain $S_2 = \{(0,0)\}$. We homogenize S_2 and we get $S_2' = \{(1:0:0)\}$. We replace $w = 1$ in p^* and obtain $p_3(z,u) = z^2u + 1$. We notice that $S_3 = \emptyset$. We return $Sing(\mathcal{C}) = \{(0:0:1), (1:0:0)\}$.

We describe the algorithm `APPROXLINK`$(p, \mathcal{C}, P, \epsilon)$ for computing the ϵ-algebraic link L_ϵ of the singularity P of the plane complex algebraic curve $\mathcal{C}$ defined by the squarefree polynomial $p(z,w) \in \mathbb{C}[z,w]$. The parameter ϵ denotes the radius of the sphere $S_\epsilon \subset \mathbb{C}^2$ which we intersect with the zero set of $p(z,w)$, as described in Definition 11.

Algorithm 2 ϵ-link of the singularity P of the plane curve $\mathcal{C}$ defined by $p(z,w)$: `APPROXLINK`$(p, \mathcal{C}, P, \epsilon)$

```
Input:  $p(z,w) \in \mathbb{C}[z,w]$ a squarefree complex polynomial,
$\mathcal{C} = \{(z,w) \in \mathbb{C}^2 | p(z,w) = 0\}$ a plane algebraic curve,
$P = (z_0, w_0)$ a numerical singularity of $\mathcal{C}$,
$\epsilon \in \mathbb{R}_{>0}$ a positive real number
Output:  $G,\ H \in \mathbb{R}[x,y,z]$
where the common zero set of $G, H$ equals $L_\epsilon$.

1. Translate the singularity $(z_0, w_0)$ in the origin by substituting $z \leftarrow z + z_0, w \leftarrow w + w_0$ in $p(z,w)$. In the new polynomial $p(z,w)$ set the terms of degree 0, 1 to zero.
2. Substitute $z \leftarrow a + ib, w \leftarrow c + id$ in $p(z,w)$ and obtain
       $p(a,b,c,d) = R(a,b,c,d) + iI(a,b,c,d),$
   with $R, I \in \mathbb{R}[a,b,c,d]$.
3. Extract $R(a,b,c,d) = I(a,b,c,d) = 0$ which define
       $\mathcal{C} = \{(a,b,c,d) \in \mathbb{R}^4 : R(a,b,c,d) = I(a,b,c,d) = 0\}.$
4. Compute the inverse of $\pi_{(\epsilon,N)}$ from Definition 1:
       $\pi^{-1}_{(\epsilon,N)} : \mathbb{R}^3 \to S_\epsilon \setminus \{N\} \subset \mathbb{R}^4$
       $(x,y,z) \mapsto (a,b,c,d) = (\frac{2x\epsilon}{n}, \frac{2y\epsilon}{n}, \frac{2z\epsilon}{n}, \frac{-\epsilon + x^2\epsilon + y^2\epsilon + z^2\epsilon}{n}),$
   where  $n = 1 + x^2 + y^2 + z^2$ .
5. Define $\alpha =: (\frac{2x\epsilon}{n}, \frac{2y\epsilon}{n}, \frac{2z\epsilon}{n}, \frac{-\epsilon + x^2\epsilon + y^2\epsilon + z^2\epsilon}{n})$,
6. Substitute $(a,b,c,d) \leftarrow \alpha$ in $\mathcal{C}$ to get $R(\alpha) = I(\alpha) = 0$.
7. Eliminate the denominators in $R(\alpha) = I(\alpha) = 0$ to get $g_\epsilon(x,y,z) = h_\epsilon(x,y,z) = 0$, with $g_\epsilon, h_\epsilon \in \mathbb{R}[x,y,z]$. For $Y = \mathcal{C} \cap S_\epsilon(P)$ these 2 equations define
       $\pi_{(\epsilon,N)}(Y) = \{(x,y,z) \in \mathbb{R}^3 :$
                 $g_\epsilon(x,y,z) = h_\epsilon(x,y,z) = 0\}.$
8. If $\pi_{(\epsilon,N)}(Y)$ has no singularities, then
     • return $G =: g_\epsilon(x,y,z)$ and $H =: h_\epsilon(x,y,z)$.
     • else return "failure".
```

We implement the algorithm `APPROXLINK` in the Axel [21] system as Axel offers a wide range of algebraic and geometric functions for manipulating algebraic curves and surfaces.

We notice that the ϵ-link of the singularity L_ϵ computed by the algorithm `APPROXLINK` is an implicit smooth space algebraic curve given as the intersection of two implicit surfaces S_1, S_2 with defining equations $g_\epsilon, h_\epsilon \in \mathbb{R}[x,y,z]$. For visualization reasons, we also compute the surfaces defined by the sum $S_1 + S_2$ and the difference $S_1 - S_2$. Thus L_ϵ is at the intersection of any two of the surfaces $\{S_1, S_2, S_1 + S_2, S_1 - S_2\}$, that are all part of the Milnor fibration. We employ subdivision methods [12] from Axel to compute the certified piecewise linear approximation (topology) of the implicit smooth space algebraic curve L_ϵ. This approximation of L_ϵ is computed as a graph $Graph(L_\epsilon)$. The data structure $Graph(L_\epsilon)$ is given as a set of vertices V together with their Euclidean coordinates in $\mathbb{R}^3$, and a set of edges E connecting them. In addition $Graph(L_\epsilon) = \langle V, E \rangle$ is isotopic to L_ϵ. In Figure 6 we visualize the link (trefoil knot) of the singularity $(0,0)$ of the plane complex algebraic curve $\mathcal{C}$ defined by the polynomial $p(z,w) = z^3 - w^2$. By using subdivision methods, Axel computes the piecewise linear approximation of the trefoil knot as a graph data structure.

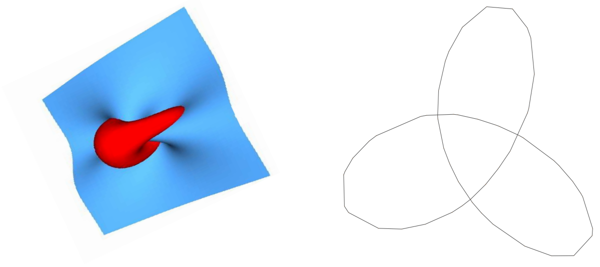


Figure 6: Piecewise linear approximation of the trefoil knot, computed as the intersection of two implicit surfaces with algorithm APPROXLINK in Axel

We next manipulate the approximation $Graph(L_\epsilon)$ symbolically to compute the ϵ-Alexander polynomial of L_ϵ. We first design an algorithm to compute the diagram $D(L_\epsilon)$ of the approximation $Graph(L_\epsilon)$, as defined in Subsection 2.2. We based this algorithm on computational geometry algorithms [6]. The algorithm requires as input the approximation $Graph(L_\epsilon) = \langle V, E \rangle$, and it returns as output the diagram $D(L_\epsilon)$, and that is: (1) the list of n crossings of $D(L_\epsilon)$ computed as all the intersections of the edges from E; (2) the list of n pairs of edges containing each intersection point. Each pair of edges (e_i, e_j) is ordered, i.e. e_i is under e_j in $\mathbb{R}^3$; (3) the r lists of edges from E for all the r knot components of $D(L_\epsilon)$; (4) the list of arcs of $D(L_\epsilon)$ and the type of each crossing. For more details on this algorithm see [10]. In Figure 7 we visualize the diagram $D(L_\epsilon)$ of the approximation $Graph(L_\epsilon)$ of the trefoil knot.

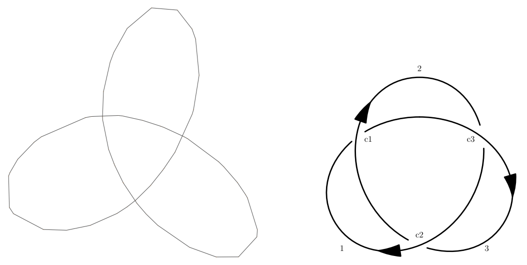


Figure 7: Diagram with 3 crossings and 3 arcs of the piecewise linear approximation of the trefoil knot

We now give the algorithm `APPROXALEXPOLY`$(D(L_\epsilon), r, n)$ for computing the ϵ-Alexander polynomial of the diagram $D(L_\epsilon)$ with r components and n crossings. We base this algorithm on Definition 8 from Subsection 2.2. For more details on this algorithm and an example see [9].

Algorithm 3 ϵ-Alexander polynomial of the diagram $D(L_\epsilon)$ of the $\texttt{APPROXALEXPOLY}(D(L_\epsilon), r, n)$

Input: $D(L_\epsilon)$ oriented algebraic link diagram of L_ϵ with r components, n crossings
Output: $\Delta_\epsilon(t_1, ..., t_r) \in \mathbb{Z}[t_1^{\pm 1}, ..., t_r^{\pm 1}]$
where $\Delta_\epsilon(t_1, ..., t_r)$ is the ϵ-Alexander polynomial of L_ϵ with diagram $D(L_\epsilon)$.

```
1. Denote the arcs and separately the crossings of $D(L_\epsilon)$ with $\{1, ..., n\}$;
2. Compute $LM(L_\epsilon)$ the labeling matrix of $D(L_\epsilon)$ ;
3. Compute $PM(L_\epsilon)$ the prealexander matrix of $D(L_\epsilon)$;
4. If $r = 1$ then:
    (a) Compute $M$ any $(n-1)\times(n-1)$ minor of $PM(L_\epsilon)$;
    (b) Compute $D$ the determinant of the minor $M$;
    (c) Return $\Delta_\epsilon(t_1) = Normalize(D_\epsilon)$;
5. If $r \geq 2$ then:
    (a) Compute all the $(n-1) \times (n-1)$ minors of $PM(L_\epsilon)$;
    (b) Compute $G$ the greatest common divisor of all the computed minors in 5.(a);
    (c) Return $\Delta_\epsilon(t_1, ...t_r) = Normalize(G)$.
```

We now present the algorithm $\texttt{APPROXDELTA}(\Delta_\epsilon, \mu, r)$ for computing the ϵ-delta-invariant from the ϵ-Alexander polynomial of degree μ and with r variables.

Algorithm 4 ϵ-delta-invariant of the singularity P of the plane curve $\mathcal{C}$ defined by $p(z, w)$: $\texttt{APPROXDELTA}(\Delta_\epsilon, \mu, r)$

Input: $\Delta_\epsilon(t_1, ..., t_m)$ the ϵ-Alexander polynomial of L_ϵ, L_ϵ the ϵ-algebraic link of the singularity $P = (z_0, w_0)$, μ the degree of Δ_ϵ, r the number of variables in Δ_{L_ϵ}
Output: $\delta_\epsilon \in \mathbb{Z}_{>0}$
where δ_ϵ is the ϵ-delta-invariant of $P = (z_0, w_0)$.

```
1. If $r = 1$ then return $\delta_\epsilon = \mu/2$.
2. If $r \geq 2$ then return $\delta_\epsilon = (\mu + r)/2$.
```

4. REGULARIZATION PRINCIPLES

4.1 Basic Notations

We denote by I the set of coefficient vectors of all the squarefree polynomials from $\mathbb{C}[z, w]$ of degree bounded by some natural number $m \in \mathbb{N} \setminus \{0\}$. The set $\mathcal{P} := \{\mathbb{Z}[t_1] \cup \mathbb{Z}[t_1, t_2] \cup ... \cup \mathbb{Z}[t_1, ..., t_i] \cup ...\}$ represents the set of all normalized Alexander polynomials either in the t_1 variable, or in the t_1, t_2 variables, or in the $t_1, t_2, ...t_i$ sequence of variables with $i \in \mathbb{N} \setminus \{0\}$, etc. We denote by O the discrete set of integer coefficient vectors of all the polynomials from $\mathcal{P}$. For a polynomial $p(x, y)$ of fixed degree we denote with p its corresponding coefficient vector. The sets I, O are metric spaces by the Euclidean distance of coefficient vectors, denoted with $|| \cdot ||$. The notation $| \cdot |$ represents the absolute value function.

For $p(z, w) \in \mathbb{C}[z, w]$ we denote by:

$$M_p(z, w) := \begin{pmatrix} \partial_z p(z, w) & \partial_w p(z, w) \\ \overline{z} & \overline{w} \end{pmatrix}$$

the two-by-two matrix formed by the partial derivatives of $p(z, w)$ with respect to z and w, and by the complex conjugates $\overline{z}, \overline{w}$. We denote by $Zeroes(p)$ the set of zeroes of the polynomial $p(z, w)$.

4.2 Definitions

First we establish a general framework for handling ill-posed algebraic problems using adapted regularization principles from [7, 18]. We then apply these principles to Problem 1 from Section 3, which we treat in this paper.

We define a well-posed problem as it was first formulated by J. Hadamard: a problem is said well-posed if: (i) there exists a solution to the problem (**existence**); (ii) the solution is unique (**uniqueness**); (iii) the solution depends continuously on the data in some given topological space (**stability**). Otherwise the problem is called ill-posed.

We consider the discontinuous function:

$$E : X \to Y, f \mapsto E(f), \tag{3}$$

on the metric spaces X, Y with metrics given by the Euclidean norm. The problem of computing $E(f) \in Y$ for given $f \in X$ is ill-posed as the computed output does not continuously depend on the input, i.e. the **stability** statement from the definition of well-posed problems does not hold. We define a perturbation function as follows:

Definition 12. A perturbation of $f \in X$ is defined as the function $f_- : \mathbb{R}_{>0} \to X, \delta \mapsto f_\delta$ with $||f - f_\delta|| \leq \delta$ for all $\delta \in \mathbb{R}_{>0}$. In this case f is called the exact data, f_δ the perturbed data and δ the noise level (error, tolerance).

In this framework we define a regularization as follows:

Definition 13. For any $\epsilon \in \mathbb{R}_{>0}$, let:

$$R_\epsilon : X \to Y, f \mapsto R_\epsilon(f)$$

be a continuous function. The function R_ϵ is called a regularization if there exists a bijective, monotonic function $\epsilon = \alpha(\delta), \alpha : \mathbb{R}_{>0} \to \mathbb{R}_{>0}$ with:

$$\lim_{\delta \to 0} \alpha(\delta) = 0, \tag{4}$$

such that for any $f \in X$ and for any perturbation function f_- with $||f - f_\delta|| \leq \delta$ for all $\delta \in \mathbb{R}_{>0}$, the following property holds:

$$\lim_{\delta \to 0} R_{\alpha(\delta)}(f_\delta) = E(f) \tag{5}$$

The function α is called a *parameter choice rule*, ϵ is called the *regularization parameter* and R_α is called the *regularized solution* of E. The equation (5) is called the *convergence for noisy data* property of R_α. The pair (R_α, α) is called a regularization method for solving the ill-posed problem E if the equations (4) and (5) hold.

For our problem, we consider X the set I of coefficient vectors of squarefree polynomials $p(z, w) \in \mathbb{C}[z, w]$ of degree bounded by some natural number $m \in \mathbb{N} \setminus \{0\}$ and Y the set O of integer coefficient vectors of normalized Alexander polynomials. In addition, we let:

$$E : I \to O, f \mapsto E(f) \tag{6}$$

be the exact algorithm for computing the Alexander polynomial of a plane curve singularity. Since O is a discrete set, the function E is discontinuous. Therefore, the problem of computing the Alexander polynomial $E(f) \in O$ for given $f \in I$ is ill-posed.

For every $\epsilon \in \mathbb{R}_{>0}$, we denote by:

$$A_\epsilon : U \subset I \to O, p \mapsto A_\epsilon(p) \tag{7}$$

the symbolic-numeric algorithm that computes the ϵ-Alexander polynomial $A_\epsilon(p)$ for given $(p,\epsilon) \in I \times \mathbb{R}_{>0}$, as described in Section 3. This polynomial arises as the intersection of the sphere S_ϵ with the curve $\mathcal{C}$ defined by p. We notice that A_ϵ is a partial function, because it is not defined in case the intersection $S_\epsilon \cap \mathcal{C}$ has singularities. Still the function A_ϵ is continuous in its domain of definition denoted by U.

We wish to show that A_ϵ is a regularization function for every $(p,\epsilon) \in U \subset I \times \mathbb{R}_{>0}$. Therefore, from Definition 13 we need to find a parameter choice rule $\epsilon = \alpha(\delta)$ with property (4) and that satisfies equation (5). Consequently, the pair (A_α, α) would be a regularization method for solving the ill-posed Problem 1.

4.3 Convergence Results

In this subsection, we include the lemmas and the theorems that we formulate to prove the convergence for noisy data property of the algorithm A_ϵ considered in (7). In this subsection, we sketch the main steps of the proofs. A complete proof would be beyond the scope of this submission.

Remark 1. We denote with S_K the sphere centered in $(0,0)$ of radius K, and with L_K the K-link of the singularity $(0,0)$ of the plane complex algebraic curve specified by the polynomial $f(z,w) \in \mathbb{C}[z,w]$ with exact coefficients (i.e. integer or rational numbers). From Theorem 1 for the case $n=1$, we know that there exists K sufficiently small such that the K-link denoted L_K, which is defined as the image through stereographic projection of $Zeroes(f) \cap S_K$, coincide with the link of the singularity $(0,0)$. From Definition 4, L_K has no singularities. Since the stereographic projection is an homeomorphism, we obtain that $Zeroes(f) \cap S_K$ has no singularities. In the same setting, we formulate the following proposition: the intersection $Zeroes(f) \cap S_K$ has no singularities if and only if the equations $f(z,w) = |z|^2 + |w|^2 = det(M_f)(z,w) = 0$ have no common solutions. The proof of this proposition is beyond the scope of this submission.

First we set the general mathematical setting required for our study. Let $f(z,w)$ be arbitrary but fixed. For simplicity we denote $f_\delta(z,w) := g(z,w) \in \mathbb{C}[z,w]$ with $||g-f|| \le \delta$. Based on Remark 1, we take $K > 0$ such that the system:

$$f(z,w) = \det(M_f)(z,w) = 0 \tag{8}$$

has no common solution except for $(0,0)$ in the closed ball $B_K := \Big\{(z,w) \in \mathbb{C}^2 : \Big(|z|^2 + |w|^2\Big)^{1/2} \le K\Big\}$ of radius K around $(0,0) \in \mathbb{C}^2$. Thus the following relation holds:

$$f(0,0) = \det(M_f)(0,0) = 0, \tag{9}$$

and $B_K \cap Zeroes(f)$ has no singularities except for $(0,0)$.

To prove the convergence for noisy data property, we require a preliminary lemma.

Lemma 1. *There exists $N > 0$ such that for all $\delta > 0$, and for all g with $||g-f|| \le \delta$ there exists no zero for the system of polynomial equations determined by $g(z,w) = \det(M_g)(z,w) = 0$ whose length is greater than $\delta^{1/N}$ and less than K.*

To prove Lemma 1 we prove the equivalent statement:

$$\begin{gathered}\exists N > 0\ \forall \delta > 0\ \forall g : ||g-f|| \le 0\ \forall (z,w) : \\ g(z,w) = \det(M_g)(z,w) = 0 \text{ and} \\ \Big(|z|^2 + |w^2|\Big)^{1/2} \le K \Rightarrow \Big(|z|^2 + |w^2|\Big)^{1/2} \le \delta^{1/N}.\end{gathered} \tag{10}$$

We take $\delta > 0$ and g with $||g-f|| \le \delta$.

First, we define the set Z_δ of "special" zeroes of g :

$$\begin{aligned} Z_\delta = \Big\{\Big((z,w),g\Big) : ||g-f|| \le \delta, \\ g(z,w) = \det(M_g)(z,w) = 0, \\ \Big(|z|^2 + |w|^2\Big)^{1/2} \le K\Big\}. \end{aligned} \tag{11}$$

We introduce the function:

$$\begin{gathered}\tau : B_K \times I \to \mathbb{R}_{\ge 0} \\ \Big((z,w),g\Big) \mapsto \tau\Big((z,w),g\Big) = \Big(|z|^2 + |w|^2\Big)^{1/2}.\end{gathered} \tag{12}$$

By using the theorem on Euclidean extreme values of real-valued functions, we prove that τ attains its maximum and we define the monotonic, semialgebraic function:

$$\begin{gathered}\beta : \mathbb{R}_{>0} \to \mathbb{R}_{\ge 0} \\ \delta \mapsto \beta(\delta) = max\ \{\tau(a) : a \in Z_\delta\}.\end{gathered} \tag{13}$$

Secondly, we prove the convergence of β by using the theorem of Bolzano-Weierstrass on compact sets.

Finally, we show that the function β is bounded from above. We use the following theorem for estimating the rate of growth of a semialgebraic function of one variable:

Theorem 2. *([3]) Let $f : (a,\infty) \to \mathbb{R}$ be a semialgebraic function (not necessarily continuous). There exists $b \ge a$ and an integer $N \in \mathbb{N}$ such that $|f(x)| \le x^N$ for all $x \in (b,\infty)$.*

Moreover, we use the following theorem for ensuring the piecewise continuity of a semialgebraic function:

Theorem 3. *([14]) Let F be a real closed field and $f : F \to F$ be a semialgebraic function. Then, we can partition F into $I_1 \cup ... I_m \cup X$, where X is finite and I_j are pairwise disjoint open intervals with endpoints in $F \cup \{\pm\infty\}$ such that f is continuous on each I_j with $j \in \{1,...,m\}$ and $m \in \mathbb{N}$.*

We get that there exists $N \in \mathbb{N} \setminus \{0\},\ b \in \mathbb{R}_+$ such that:

$$\beta_r(\delta) \le \delta^{1/N},$$

for all $\delta < \eta = b^{-1}$, where β_r is the restriction of β to the first open interval.

We use Lemma 1 as a tool for proving the convergence for noisy data statement (5) and for ensuring the existence of a parameter choice rule (4) for A_ϵ. This convergence statement is given by the following theorem:

Theorem 4. *There exists $N > 0$ and $\eta \in \mathbb{R}_{>0}$ such that for all $\delta > 0$ with $\delta < \eta$, for all g with $||g-f|| \le \delta$ and for all $\epsilon \in [\delta^{1/N}, K]$, the following property holds: $A_\epsilon(g) = E(f)$.*

We prove Theorem 4 by constructing the isotopy:

$$\begin{array}{l} g_t : \mathbb{C}^2 \times [0,1] \rightarrow \mathbb{C} \\ (z,w) \rightarrow g_t(z,w) = tf(z,w) + (1-t)g(z,w), \end{array} \tag{14}$$

with g_t continuous function for all $0 \leq t \leq 1$, and $g_0 = g$, $g_1 = f$, and by showing that $A_\epsilon(g_t)$ is an ϵ-algebraic link based on Lemma 1.

From Theorem 4 it follows that $\epsilon = \delta^{1/N}$ is a parameter choice rule for A_ϵ, for which the convergence for noisy data statement (5) of A_ϵ holds. Still, this parameter choice rule depends on N which is unknown. The following lemma provides us with an upper bound for $\delta^{1/N}$ which is independent on N:

Lemma 2. *For all $N > 0$ there exists $\theta \in \mathbb{R}_+$ such that for all $\delta > 0$ with $\delta < \theta$, the inequality $\delta^{1/N} \leq \frac{1}{|ln\delta|}$ is true.*

We prove Lemma 2 by basic calculus and by using l'Hôpital rule. The preceding two lemmas allow us to formulate the following theorem concerning the existence of a parameter choice rule for A_ϵ which only depends on the given $\delta \in \mathbb{R}_+$:

Theorem 5. *The function $\alpha : \mathbb{R}_{>0} \rightarrow \mathbb{R}_{>0}, \alpha(\delta) = \frac{1}{|ln\delta|}$ is a parameter choice rule, i.e.*

$$\lim_{\delta \to 0} A_{\alpha(\delta)}(f_\delta) = E(f) \tag{15}$$

The theorem is true based on Lemma 1, Theorem 4 and Lemma 2.

Remark 2. The parameter choice rule indicates that the "degree of ill-posedness" is rather high (cf. with linear regularization theory [18], where $\alpha(\delta) = \delta^{1/2}$ frequently occurs). For fixed input instance f, the smallest function $\alpha : \mathbb{R}_{>0} \rightarrow \mathbb{R}_{>0}$ such that (noisy convergence) is true is equal to the function β from Lemma 1. The choice of α was done in order to ensure that α dominates β for every possible f. Here is an example that shows that a semi-algebraic parameter choice rule cannot be used as a choice rule.

Example 3. Let $n > 0$ be an integer. Let $f(z,w) = z^2 - w^{n+2}$. We consider the perturbation $g(z,w) = f_\delta(z,w) = z^2 - w^{n+2} + \delta w^2$, for $\delta \in (0,1)$. Then we have a special zero of (g, M_g) at $(z,w) = (0, \delta^{1/n})$. A closer analysis shows that the ϵ-link of g is the Hopf link for every sphere with radius less than $\delta^{1/n}$, while the link of f is equal to the torus link $(2, n+2)$. Consequently, $\beta(\delta) > \delta^{1/n}$ for this choice of f. Since n can be arbitrary, no function which is dominated by a function of the from $\delta \mapsto \delta^{1/m}$ for some m can be chosen as a parameter choice rule.

5. IMPLEMENTATION

5.1 A Library for Algebraic Curves

We implemented the symbolic-numeric algorithms for computing invariants of a plane complex algebraic curve described in Section 3 in the free library GENOM3CK [8]-GENus cOMputation of a plane Complex algebraiC Curve using Knot theory-written in the Axel free algebraic geometric modeler [21] and in the Mathemagix free computer algebra system [11], i.e. in C++ using Qt Script for Applications and OpenGL. By using Axel, we integrate symbolic, numeric and graphical capabilities into a single library. Together with its main functionality to compute the genus, the library performs operations in topology, algebraic geometry and knot theory. More information on GENOM3CK at: http://people.ricam.oeaw.ac.at/m.hodorog/software.html.

5.2 Test Experiments

We include several experiments performed with the library GENOM3CK in Axel. In Figure 8 we consider the plane complex algebraic curve defined by the squarefree polynomial $p(z,w) = z^3 - w^3$, with a singularity in the origin and the input parameter $\epsilon = 1.00$. From left to right, we visualize: (1) the link L_ϵ of the singularity computed as the intersection of two implicit surfaces S_1, S_2; (2) the two surfaces S_1, S_2; (3) the four surfaces $S_1, S_2, S_1 + S_2, S_1 - S_2$, which are all part of the Milnor fibration of the singularity.

The test experiments indicate the convergence for noisy data property of the regularization method as proved in Section 4. In Table 1 we consider several input curves defined by squarefree polynomials, which have the singularity in the origin. The first column indicates the defining polynomial of the curve, the second one the value for ϵ and the next columns contain the computed values for the ϵ-link, the ϵ-Alexander polynomial and respectively the ϵ-delta-invariant of the singularity. We emphasize that for the curves $\mathcal{C}$ and $\mathcal{D}$ defined by $p(z,w) = z^3 + z^2 - w^3$ and $\tilde{p}(z,w) = z^3 + z^2 - w^3 - 0.1w^2$ from Example 1, the singularity $(0,0)$ of $\mathcal{C}$ changes its type under small perturbations of $p(z,w)$. By using the algorithm `APPROXLINK` we observe that for $\epsilon = 0.25$ the link of $(0,0)$ of $\mathcal{C}$ coincide with the link of $(0,0)$ of $\mathcal{D}$.

6. CONCLUSION

We presented symbolic-numeric algorithms for computing invariants for each singularity of a plane complex algebraic curve: the link of each singularity, the Alexander polynomial, and the delta-invariant. We implemented the algorithms in a free library that combines graphical, numerical and symbolic capabilities. We employed regularization principles to handle the ill-posedness of the problem.

7. ACKNOWLEDGMENTS

Special thanks to Bernard Mourrain, who contributed to the implementation of the library and who offered significant software-development assistance and mathematical support. Many thanks to Julien Wintz, who contributed to the implementation of the library in its starting phase. Many thanks to Esther Klann, to Ronny Ramlau, and to other colleagues from the "Doctoral Program-Computational Mathematics" for their helpful comments. This work is supported by the Austrian Science Funds (FWF) under the grant W1214/DK9.

8. REFERENCES

[1] J. W. Alexander. Topological invariant of knots and links. *Transactions of the American Mathematical Society*, 30:275–306, 1928.

[2] D. J. Bates, C. Peterson, A. J. Sommese, and C. W. Wampler. Numerical computation of the genus of an irreducible curve within an algebraic set. To appear in *J. of Pure and Applied Algebra*.

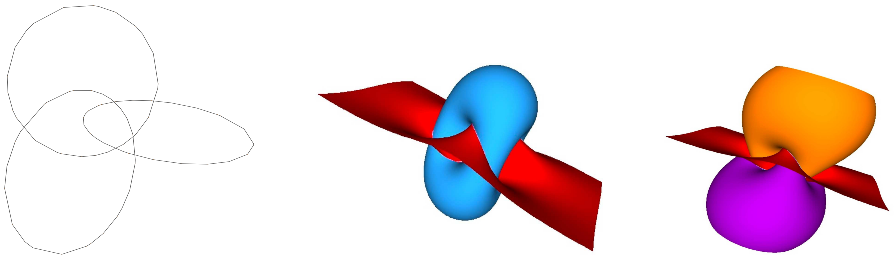


Figure 8: Link, Milnor fibration of the singularity $(0,0)$ of the plane complex algebraic curve defined by $z^3 - w^3$

Table 1: Evidence for the Convergence for Noisy Data Statement

Equation of the plane complex algebraic curve	$\epsilon \in \mathbb{R}_{>0}$	ϵ**-link**	ϵ**-Alexander polynomial**	ϵ**-delta-invariant**
$z^3 + z^2 - w^3$	1.5	3-knots link	$\Delta(t_1) = -t_1t_2t_3 + 1$	$\delta = 3$
$z^3 + z^2 - w^3$	0.25	Trefoil knot	$\Delta(t_1, t_2) = t_1^2 - t_1 + 1$	$\delta = 1$
$z^3 + z^2 - w^3 - 0.1w^2$	1.5	3-knots link	$\Delta(t_1) = -t_1t_2t_3 + 1$	$\delta = 3$
$z^3 + z^2 - w^3 - 0.1w^2$	0.25	Trefoil knot	$\Delta(t_1, t_2) = t_1^2 - t_1 + 1$	$\delta = 1$

[3] J. Bochnak, M. Coste, and M.-F. Roy. *Real Algebraic Geometry*. Ergebnisse der Math. Springer Verlag, Germany, 1998.

[4] K. Brauner. Zur geometrie der funktionen zweier veränderlichen:ii-iv. *Abh. Math. Sem. Hamburg*, 6:1–54, 1928.

[5] D. Cimasoni. Studying the multivariable Alexander polynomial by means of Seifert surfaces. *Bol. Soc. Mat. Mexicana (3)*, 10:107–115, 2004.

[6] M. de Berg, M. Krefeld, M. Overmars, and O. Schwarzkopf. *Computational geometry: algorithms and applications. Second edition*. Springer, Berlin, 2008.

[7] H. W. Engl, M. Hanke, and A. Neubauer. *Regularization of inverse problems*. Kluwer Academic Publishers Group, Dordrecht, Netherlands, 1996.

[8] M. Hodorog, B. Mourrain, and J. Schicho. GENOM3CK - A library for genus computation of plane complex algebraic curves using knot theory, December 2010. ACM SIGSAM Communications in Computer Algebra, vol. 44, issue 174, pp. 198-200, ISSN:1932-2240.

[9] M. Hodorog, B. Mourrain, and J. Schicho. A symbolic-numeric algorithm for computing the Alexander polynomial of a plane curve singularity. In T. Ida, V. Negru, T. Jebelean, D. Petcu, S. Watt, and D. Zaharie, editors, *Proceedings of the 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 21–28, 2010.

[10] M. Hodorog and J. Schicho. A symbolic-numeric algorithm for genus computation. In U. Langer and P. Paule, editors, *Numerical and Symbolic Scientific Computing: Progress and Prospects*. Springer Wien, Austria, 2011. To appear.

[11] V. D. J. Hoeven, G. Lecerf, and B. Mourrain. Mathemagix computer algebra system. http://www.mathemagix.org/www/main/index.en.html.

[12] C. Liang, B. Mourrain, and J. Pavone. Subdivision methods for 2d and 3d implicit curves. In B. Jüttler and R. Piene, editors, *Geometric Modeling and Algebraic Geometry*, pages 199–214. Springer, 2008.

[13] C. Livingston. *Knot theory*. Mathematical Association of America, U.S.A, 1993.

[14] D. Marker. *Model Theory: An Introduction*. Graduate Texts in Mathematics. Springer New York, United States of America, 2002.

[15] J. Milnor. *Singular points of complex hypersurfaces*. Princeton University Press and the University of Tokyo Press, New Jersey, 1968.

[16] B. Mourrain and J. Pavone. Subdivision methods for solving polynomial equations. *Symbolic Computation*, 44:292–306, 2009.

[17] S. Pérez-Díaz, J. R. Sendra, S. L. Rueda, and J. Sendra. Approximate parametrization of plane algebraic curves by linear systems of curves. *Computer Aided Geometric Design*, 27(2):212–231, February 2010.

[18] A. N. Tikhonov and V. A. Arsenin. *Solution of Ill-posed Problems*. V. H. Winston & Sons, Washington, D.C., 1977.

[19] R. J. Walker. *Algebraic Curves*. Springer-Verlag, New York, United States of America, 1978.

[20] F. Winkler. *Polynomial algorithms in computer algebra*. Springer-Verlag, Wien, New York, 1996.

[21] J. Wintz, S. Chau, L. Alberti, and B. Mourrain. Axel algebraic geometric modeler. http://axel.inria.fr/.

[22] M. Yamamoto. Classification of isolated algebraic singularities by their Alexander polynomials. *Topology*, 23:277–287, 1984.

Schönhage-Strassen Algorithm with MapReduce for Multiplying Terabit Integers

Tsz-Wo Sze
Yahoo! Cloud Platform
Sunnyvale, California, USA
tsz@yahoo-inc.com

ABSTRACT

We present *MapReduce-SSA*, an integer multiplication algorithm using the ideas from Schönhage-Strassen algorithm (SSA) on MapReduce. SSA is one of the most commonly used large integer multiplication algorithms. MapReduce is a programming model invented for distributed data processing on large clusters. MapReduce-SSA is designed for multiplying integers in terabit scale on clusters of commodity machines. As parts of MapReduce-SSA, two algorithms, *MapReduce-FFT* and *MapReduce-Sum*, are created for computing discrete Fourier transforms and summations. These mathematical algorithms match the model of MapReduce seamlessly.

Categories and Subject Descriptors

I.1.2 [**Computing Methodologies**]: Algorithms—*symbolic and algebraic manipulation*; F.2.1 [**Theory of Computation**]: Numerical Algorithms—*computation of fast Fourier transforms*; G.4 [**Mathematics of Computing**]: Mathematical Software

General Terms

Algorithms, distributed computing

Keywords

Integer multiplication, multiprecision arithmetic, fast Fourier transform, summation, MapReduce

1. INTRODUCTION

Integer multiplication is one of the most critical operations in arbitrary precision arithmetic. Beyond its direct applications, the existence of fast multiplication algorithms running in essentially linear time and the reducibility of other common operations such as division, square root, logarithm, etc. to integer multiplication motivate improvements in techniques for its efficient computation at scale [4, 8]. For terabit scale integer multiplication, the major application of it is the computation of classical constants, in particular, the computation of π [21, 1, 18, 7, 5].

The known, asymptotically fastest multiplication algorithms are based on fast Fourier transform (FFT). In practice, multiplication libraries often employ a mix of algorithms, as the performance and suitability of an algorithm vary depending on the bit range computed and on the particular machine where the multiplication is performed. While benchmarks inform the tuning of an implementation on a given computation platform, algorithms best suited to particular bit ranges are roughly ranked both by empirical and theoretical results. Generally, the naïve algorithm performs best in the lowest bit range, then the Karatsuba algorithm, then the Toom-Cook algorithm, and multiplication in the largest bit ranges typically employs FFT-based algorithms. For details, see [3, 13].

The Schönhage-Strassen algorithm (SSA) is one such FFT-based integer multiplication algorithm, requiring

$$O(N \log N \log\log N)$$

bit operations to multiply two N-bit integers [15]. There are other FFT-based algorithms asymptotically faster than SSA. In 2007, Fürer published an algorithm with running time

$$O(N(\log N)2^{\log^* N})$$

bit operations [11]. Using similar ideas from Fürer's algorithm, De et al. discovered another $O(N(\log N)2^{\log^* N})$ algorithm using modular arithmetic while the arithmetic of Fürer's algorithm is carried out over complex numbers [9]. As these two algorithms are relatively new, they are not found in common software packages and the bit ranges where they begin to outperform SSA are, as yet, unclear. SSA remains a dominant implementation in practice. See [12, 6] for details on SSA implementations.

Algorithms for multiplying large integers, including SSA, require a large amount of memory to store the input operands, intermediate results, and output product. For in-place SSA, the total space required to multiply to N-bit integers ranges from $8N$ to $10N$ bits; see §3.1. Traditionally, multiplication of terabit-scale integers is performed on supercomputers with software customized to exploit its particular memory and interconnect architecture [5, 7, 18]. Takahashi recently computed the square of a 2 TB integer with 5.15 trillion decimal digits in 31 minutes and 41 seconds using the 17.5 TB main memory on the T2K Open Supercomputer [17].

Most arbitrary precision arithmetic software packages assume a uniform address space on a single machine. The precision supported by libraries such as GMP and Magma is limited by the available physical memory. When memory is exhausted during a computation, these packages may have very poor performance or may not work at all. Such packages are not suited to terabit scale multiplication, as the memory available in most machines is insufficient by orders of magnitude. Other implementations such as *y-cruncher* [20] and *TachusPI* [2] use clever paging techniques to scale a single machine beyond these limits. By using the hard disk sparingly, its significantly higher capacity enables efficient computation despite its high latencies. According to Yee, y-cruncher is able to multiple integers with 5.2 trillion decimal digits using a desktop computer with only 96 GB memory in 41.6 hours [19].

In this paper, we present a distributed variety of SSA for multiplying terabit integers on commodity hardware using a shared compute layer. Our prototype implementation runs on *Hadoop* clusters. Hadoop is an open source, distributed computing framework developed by the Apache Software Foundation. It includes a fault-tolerant, distributed file system, namely *HDFS*, designed for high-throughput access to very large data sets in petabyte scale [16]. It also includes an implementation of *MapReduce*, a programming model designed for processing big data sets on large clusters [10].

Our environment is distinguished both from the supercomputer and single machine variants described in the preceding paragraphs. Unlike these settings, our execution environment assumes that hardware failures are common. The fixed computation plan offered by our host environment is designed for multi-tenant, concurrent batch processing of large data volumes on a shared storage and compute grid. As a sample constraint, communication patterns between processes are fixed and communicated only at their conclusion, to ensure execution plans remain feasible should a machine fail. While the problem of memory remains a central and familiar consideration of the implementation, some of the challenges posed and the advantages revealed by our programming paradigm and execution environment are unique. We learned that not only SSA, but FFT and other arbitrary precision routines like summation match the MapReduce programming model seamlessly. More specifically, the FFT matrix transposition, which is traditionally difficult in preserving locality, becomes trivial in MapReduce.

The rest of the sections are organized as follows. More details on computation environment are given in §2. SSA is briefly described in §3. A new algorithm, *MapReduce-SSA*, is presented in §4. The conclusion is in §5.

2. COMPUTATION ENVIRONMENT

In this section, we first describe our cluster configurations, and then give a short introduction of MapReduce.

2.1 Cluster Configurations

At Yahoo!, Hadoop clusters are composed of hundreds or thousands of commodity machines. All the machines in the same cluster are collocated in the same data center. Each machine typically has

- 2 quad-core CPUs,
- 6 GB or more memory, and
- 4 hard drives.

The machines are connected with a two-level network topology shown in Figure 1. There are 40 machines connected to a rack switch by 1-gigabit links. Each rack switch is connected to a set of core switches with 8-gigabit or higher aggregate bandwidth.

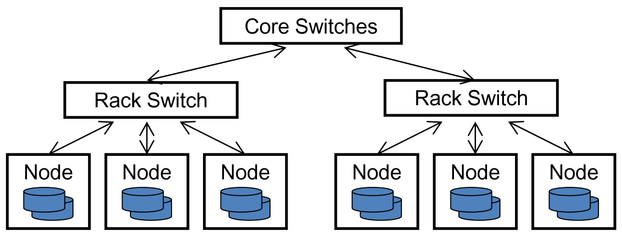


Figure 1: Network topology

For terabit scale integer multiplication, the data involved in the matrix transposition step in parallel FFT is terabyte scale. Small clusters with tens of machines are inadequate to perform such operation, even if individual machines are equipped with high performance CPUs and storage, since the network bandwidth is a bottleneck of the system. The benefits of adding machines to a cluster are twofold: it increases the number of CPUs and the size of storage, and also increases the aggregate network bandwidth. From our experience, clusters with five hundred machines are sufficient to perform terabit scale multiplication.

2.2 MapReduce

In the MapReduce model, the input and the output of a computation are lists of key-value pairs. Let k_t be key types and v_t be value types for $t = 1, 2, 3$. The user specifies two functions,

$$\begin{aligned} \text{map}: &\quad (k_1, v_1) \longrightarrow \text{list}\langle k_2, v_2\rangle, \quad \text{and} \\ \text{reduce}: &\quad (k_2, \text{list}\langle v_2\rangle) \longrightarrow \text{list}\langle k_3, v_3\rangle. \end{aligned}$$

When a *job* starts, the MapReduce framework launches *map tasks*. A map task transforms one or more inputs in (k_1, v_1) to intermediate outputs in (k_2, v_2) according to map$(\,\cdot\,)$. Once all map tasks have completed, the framework begins a process called *shuffle*. It puts all intermediate values with the same key into a list, launches *reduce tasks* and sends the keys with the corresponding list to the reduce tasks. A reduce task uses reduce$(\,\cdot\,)$ to process one or more of the key-list pairs in $(k_2, \text{list}\langle v_2\rangle)$ and outputs key-value pairs in (k_3, v_3). The job is finished when all reduce tasks are completed. Figure 2 shows a diagram of the MapReduce model.

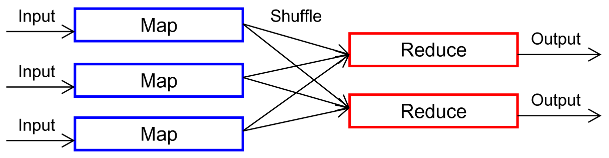


Figure 2: MapReduce model

The number of map tasks is at least one but the number of reduce tasks could possibly be zero. For a map-only job, shuffle is unnecessary and is not performed. The output from the map tasks becomes the output of the job.

The MapReduce framework included in Hadoop is highly parallel, locality-aware, elastic, flexible and fault tolerant.

Operators routinely add new machines and decommission failing ones without affecting running jobs. Hadoop provides a flexible API and permits users to define various components, including input/output formats that define how data is read and written from HDFS. Hardware failures are common, so the framework must also detect and gracefully tolerate such events. As a MapReduce application, recovery is transparent as long as it conforms to the assumptions and conventions of the framework.

3. SCHÖNHAGE-STRASSEN

We describe SSA briefly in this section. For more detailed discussions, see [15, 8, 3].

The goal is to compute

$$p \quad = \quad ab, \tag{1}$$

where $a, b \in \mathbb{Z}$ are the input integers and p is the product of a and b. Without loss of generality, we may assume $a > 0$ and $b > 0$. Suppose a and b are N-bit integers for N a power of two. Write

$$N = KM = 2^k M,$$

where K and M are also powers of two. Let

$$A(x) \stackrel{\mathsf{def}}{=} \sum_{i=0}^{K-1} a_i x^i \qquad \text{for } 0 \le a_i < 2^M \qquad \text{and}$$

$$B(x) \stackrel{\mathsf{def}}{=} \sum_{i=0}^{K-1} b_i x^i \qquad \text{for } 0 \le b_i < 2^M$$

be polynomials such that $A(x), B(x) \in \mathbb{Z}[x]$, $a = A(2^M)$ and $b = B(2^M)$. There are M bits in each coefficient a_i and b_i. The number of coefficients in each polynomial $A(x)$ and $B(x)$ is K. Let

$$D \stackrel{\mathsf{def}}{=} 2K = 2^{k+1}.$$

The strategy is to compute the product polynomial

$$P(x) \stackrel{\mathsf{def}}{=} A(x)B(x) \stackrel{\mathsf{def}}{=} \sum_{i=0}^{D-1} p_i x^i \qquad \text{for } 0 \le p_i < 2^{2M+k}.$$

Then, we have $p = P(2^M)$. Consider polynomials A, B and P as D-dimensional tuples,

$$\begin{aligned} \mathbf{a} &\stackrel{\mathsf{def}}{=} (0, \ldots, 0, a_{K-1}, \ldots, a_0), \\ \mathbf{b} &\stackrel{\mathsf{def}}{=} (0, \ldots, 0, b_{K-1}, \ldots, b_0), \qquad \text{and} \\ \mathbf{p} &\stackrel{\mathsf{def}}{=} (p_{D-1}, \ldots, p_0). \end{aligned}$$

Then $\mathbf{p}$ is the cyclic convolution of $\mathbf{a}$ and $\mathbf{b}$,

$$\mathbf{p} = \mathbf{a} \times \mathbf{b}.$$

By the convolution theorem,

$$\mathbf{a} \times \mathbf{b} \quad = \quad \mathrm{dft}^{-1}(\mathrm{dft}(\mathbf{a}) * \mathrm{dft}(\mathbf{b})), \tag{2}$$

where $*$ denotes componentwise multiplication, $\mathrm{dft}(\cdot)$ and $\mathrm{dft}^{-1}(\cdot)$ respectively denote the discrete Fourier transform and its inverse. All the operations on the right hand side of equation (2) are performed over the ring $\mathbb{Z}/(2^n + 1)\mathbb{Z}$. The integer $2^n + 1$ is called *the Schönhage-Strassen modulus.* The exponent n must satisfy the following conditions,

(C_1) $D \mid 2n$, and

(C_2) $n \ge 2M + k$.

Thus, n is chosen to be the smallest integer satisfying both conditions (C_1) and (C_2). At last, the componentwise multiplications are computed by recursive calls to the integer multiplication routine, and dft (or dft^{-1}) is calculated by forward (or backward) FFT.

There are some well-known improvements on the original SSA. For example, condition (C_1) can be relaxed to $D \mid 4n$ using the $\sqrt{2}$ trick; see [12].

3.1 Memory Requirement

For multiplying two N-bit integers, the total input size is $2N$ bits. The output size is $2N$ bits as well. The values of $\mathrm{dft}(\mathbf{a})$ and $\mathrm{dft}(\mathbf{b})$ in equation (2) are the intermediate results. We have

$$\mathrm{dft}(\mathbf{a}) \in \left(\frac{\mathbb{Z}}{(2^n + 1)\mathbb{Z}} \right)^D,$$

a D-dimension tuple. By condition (C_2), the size is

$$|\,\mathrm{dft}(\mathbf{a})| \ge nD \ge (2M + k)D > 4MK = 4N.$$

It is possible to choose the SSA parameter such that the DFT size is within the $(4N, 5N]$ interval. Therefore, the total space required for the multiplication is in $(12N, 14N]$ if the buffers for the input, the intermediate results and the output are separated. For an in-place implementation, i.e. the buffers are reused, the total required space decreases to $(8N, 10N]$. As an example, the required memory is from 1 TB to 1.25 TB for multiplying two 1-terabit integers with an in-place implementation; see also §4.4 and Table 2.

4. MapReduce-SSA

In our design, we target on input integers in terabit scale. SSA is a recursive algorithm. The first level call has to process multi-terabit data. The parameters are selected such that the data in the recursive calls is only in megabit or gigabit scale. Therefore, the recursive calls can be executed in a single machine locally. MapReduce framework is used to handle the first level call so that the computation is distributed across the machines and executed in parallel.

4.1 Data Model

In order to allow accessing terabit integers efficiently, an integer is divided into sequences of records. An integer x is represented as a D-dimensional tuple

$$\mathbf{x} = (x_{D-1}, x_{D-2}, \ldots, x_0) \in \mathbb{Z}^D$$

as illustrated in §3. It is customary to call the components of $\mathbf{x}$ *the digits of* x. Let

$$D = IJ. \tag{3}$$

where $I > 1$ and $J > 1$ are powers of two. For $0 \le i < I$, define a J-dimensional tuple

$$\mathbf{x}^{(i)} \stackrel{\mathsf{def}}{=} (x_{(J-1)I+i}, x_{(J-2)I+i}, \ldots, x_i) \in \mathbb{Z}^J \tag{4}$$

so that

$$\begin{pmatrix} \mathbf{x}^{(0)} \\ \mathbf{x}^{(1)} \\ \vdots \\ \mathbf{x}^{(I-1)} \end{pmatrix} = \begin{pmatrix} x_{(J-1)I} & x_{(J-2)I} & \cdots & x_0 \\ x_{(J-1)I+1} & x_{(J-2)I+1} & \cdots & x_1 \\ \vdots & \vdots & \ddots & \vdots \\ x_{(J-1)I+(I-1)} & x_{(J-2)I+(I-1)} & \cdots & x_{I-1} \end{pmatrix}.$$

We call $(\mathbf{x}^{(0)}, \ldots, \mathbf{x}^{(I-1)})^t$ *the* (I, J)*-format of* $\mathbf{x}$.

Each $\mathbf{x}^{(i)}$ is represented as a sequence of J records and each record is a key-value pair, where the key is the index and the value is the value of the digit as shown in Table 1. There are I sequences for the entire tuple $\mathbf{x}$.

Record #	<Key, Value>
0	$<i, \quad x_i>$
1	$<J+i, \quad x_{J+i}>$
$\vdots$	$\vdots$
$J-1$	$<(J-1)I+i, \quad x_{(J-1)I+i}>$

Table 1: The representation of $\mathbf{x}^{(i)}$

We give a few definitions in the following. A tuple

$$\mathbf{x} = (x_{D-1}, \ldots, x_0)$$

is *normalized* if $0 \le x_t < 2^M$ for all $0 \le t < D$. The input tuples of the algorithm are assumed to be normalized and the output tuple returned is also normalized. Define the *left component-shift* operator $\ll$, for $m \ge 0$,

$$\mathbf{x} \ll m \overset{\text{def}}{=} (x_{D-m-1}, x_{D-m-2}, \ldots, x_0, \underbrace{0, \ldots, 0}_{m}). \qquad (5)$$

By definition, $\mathbf{x} \ll 0$ is a no-op. Let $\mathbf{y} = (y_{D-1}, \ldots, y_0)$. Define the *addition with carrying* operator $\oplus$ as below,

$$\mathbf{x} \oplus \mathbf{y} \overset{\text{def}}{=} (s_{D-1} \bmod 2^M, \ldots, s_0 \bmod 2^M), \qquad (6)$$

where $s_{-1} = 0$ and $s_t = x_t + y_t + \left\lfloor \frac{s_{t-1}}{2^M} \right\rfloor$ for $0 \le t < D$.

4.2 Algorithms

Using the notations in §3, SSA consists of four steps,

S1: two forward FFTs, $\hat{\mathbf{a}} \overset{\text{def}}{=} \text{dft}(\mathbf{a})$ and $\hat{\mathbf{b}} \overset{\text{def}}{=} \text{dft}(\mathbf{b})$;

S2: componentwise multiplication, $\hat{\mathbf{p}} \overset{\text{def}}{=} \hat{\mathbf{a}} * \hat{\mathbf{b}}$;

S3: a backward FFT, $\mathbf{p} = \text{dft}^{-1}(\hat{\mathbf{p}})$; and

S4: carrying, normalizing $\mathbf{p}$.

S1 first reads the two input integers, and then runs two forward FFTs in parallel. Forward FFTs are performed by the *MapReduce-FFT* algorithm described in §4.2.2. It is clear that the componentwise multiplication in S2 can be executed in parallel. Notice that the digits in $\hat{\mathbf{a}}$ and $\hat{\mathbf{b}}$ are relatively small in size. For terabit integers, the size of the digits is only in the scale of megabits. Thus, multiplications with these digits can be computed by a single machine. We discuss componentwise multiplication in §4.2.3. In S3, the backward FFT, which is very similar to forward FFT, is again calculated by MapReduce-FFT. The carrying in S4 can be done by *MapReduce-Sum* as shown in §4.2.4.

4.2.1 Parallel-FFT

There is a well-known algorithm, called *parallel-FFT*, for evaluating a DFT in parallel. As before, let

$$\hat{\mathbf{a}} \overset{\text{def}}{=} (\hat{a}_{D-1}, \hat{a}_{D-2}, \ldots, \hat{a}_0) \overset{\text{def}}{=} \text{dft}(\mathbf{a}).$$

By the definition of DFT,

$$\hat{a}_j = \sum_{i=0}^{D-1} a_i \zeta^{ij} \qquad \text{for } 0 \le i < D, \qquad (7)$$

where ζ is a principle Dth root of unity in $\mathbb{Z}/(2^n+1)\mathbb{Z}$. We may take

$$\zeta = 2^{2n/D} \pmod{2^n + 1}.$$

Rewrite the summation, for all $0 \le j_0 < J$ and $0 \le j_1 < I$,

$$\begin{aligned} \hat{a}_{j_1 J + j_0} &= \sum_{i_0=0}^{I-1} \sum_{i_1=0}^{J-1} \zeta^{(i_1 I + i_0)(j_1 J + j_0)} a_{i_1 I + i_0} \\ &= \sum_{i_0=0}^{I-1} \zeta_I^{i_0 j_1} \left(\zeta^{i_0 j_0} \sum_{i_1=0}^{J-1} \zeta_J^{i_1 j_0} a_{i_1 I + i_0} \right), \end{aligned}$$

where $\zeta_I \overset{\text{def}}{=} \zeta^J$ and $\zeta_J \overset{\text{def}}{=} \zeta^I$ are principle Ith and Jth roots of unity in $\mathbb{Z}/(2^n+1)\mathbb{Z}$, respectively. For $0 \le i < I$, let

$$\mathbf{a}^{(i)} \overset{\text{def}}{=} (a_{(J-1)I+i}, a_{(J-2)I+i}, \ldots, a_i)$$

and

$$\widehat{\mathbf{a}^{(i)}} \overset{\text{def}}{=} \left(\widehat{a^{(i)}}_{J-1}, \widehat{a^{(i)}}_{J-2}, \ldots, \widehat{a^{(i)}}_0 \right) \overset{\text{def}}{=} \text{dft}(\mathbf{a}^{(i)}). \qquad (8)$$

These I DFTs with J points can be calculated in parallel. For $0 \le i < I$ and $0 \le j < J$, define

$$\begin{aligned} z_{jI+i} &\overset{\text{def}}{=} \zeta^{ij} \, \widehat{a^{(i)}}_j, & (9) \\ \mathbf{z}^{[j]} &\overset{\text{def}}{=} (z_{jI+(I-1)}, z_{jI+(I-2)}, \ldots, z_{jI}), & (10) \\ \widehat{\mathbf{z}^{[j]}} &\overset{\text{def}}{=} \text{dft}(\mathbf{z}^{[j]}). & (11) \end{aligned}$$

Notice that we use $[\cdot]$ in equation (10) in order to emphasize the difference between $\mathbf{z}^{[j]}$ and $\mathbf{z}^{(j)}$. The J DFTs with I points in equation (11) can be calculated in parallel. Finally,

$$\hat{a}_{j_1 J + j_0} = \sum_{i_0=0}^{I-1} \zeta_I^{i_0 j_1} z_{j_0 I + i_0} = \widehat{z^{[j_0]}}_{j_1}.$$

4.2.2 MapReduce-FFT

We may carry out parallel-FFT on MapReduce. For forward FFT, the input $\mathbf{a}$ is in (I, J)-format — it is divided into $\mathbf{a}^{(i)}$ for $0 \le i < I$ as in §4.1. There are I map and J reduce tasks in a job. The map tasks execute I parallel J-point FFTs as shown in equation (8), while the reduce tasks run J parallel I-point FFTs as shown in equation (11). For $0 \le j < J$, the output from reduce task j is

$$\hat{\mathbf{a}}^{(j)} = (\hat{a}_{(I-1)J+j}, \hat{a}_{(I-2)J+j}, \ldots, \hat{a}_j) = \widehat{\mathbf{z}^{[j]}}.$$

The output $\hat{\mathbf{a}}$ is written in (J, I)-format. The mapper and reducer algorithms are shown in Algorithms 1 and 2. Note that equation (9) computed in *(f.m.3)* indeed is a bit-shifting since ζ is a power of two. Note also that the transposition of the elements from $\zeta^{ij}\widehat{\mathbf{a}^{(i)}}$ in equation (9) to $z^{[j]}$ in equation (10) is accomplished by the shuffle process, i.e. the transition from map tasks to reduce tasks.

Algorithm 1 (Forward FFT, Mapper).

(f.m.1) read key i, value $\mathbf{a}^{(i)}$*;*

(f.m.2) calculate a J-point DFT by equation (8);

(f.m.3) componentwise bit-shift by equation (9);

(f.m.4) for $0 \le j < J$, emit key j, value (i, z_{jI+i}).

Algorithm 2 (Forward FFT, Reducer).

(f.r.1) receive key j, list $[(i, z_{jI+i})]_{0\le i<I}$;

(f.r.2) calculate an I-point DFT by equation (11);

(f.r.3) write key j, value $\widehat{\mathbf{z}^{[j]}}$.

Backward FFT is similar to forward FFT except that (1) it uses ζ^{-1} instead of ζ in the computation, and (2) it has an extra scale multiplication by $\frac{1}{D}$. In backward FFT, the roles of I and J are interchanged. The input is $\hat{\mathbf{p}}$ in (J, I)-format and the output $\mathbf{p}$ is in (I, J)-format, where $\mathbf{p} = \text{dft}^{-1}(\hat{\mathbf{p}})$. There are J map tasks executing J parallel I-point FFTs and I reduce tasks running I parallel J-point FFTs. We skip the detailed equations and provide brief mapper and reducer algorithms in Algorithms 3 and 4. Similar to *(f.m.3)*, we may perform *(b.m.3)* and *(b.r.3)* by bit-shifting since ζ^{-1} and D are also powers of two.

Algorithm 3 (Backward FFT, Mapper).

(b.m.1) read key j, value $\hat{\mathbf{p}}^{(j)}$;

(b.m.2) calculate an I-point DFT inverse;

(b.m.3) perform a componentwise bit-shifting;

(b.m.4) emit results.

Algorithm 4 (Backward FFT, Reducer).

(b.r.1) receive the corresponding map outputs;

(b.r.2) calculate a J-point DFT inverse;

(b.r.3) componentwise bit-shift for scaling by $\frac{1}{D}$;

(b.r.4) write key i, value $\mathbf{p}^{(i)}$.

We end the section by discussing the data size involved in MapReduce-FFT. Suppose the input is an integer with 1 TB (i.e. $N = 2^{43}$) and, say, $D = 2^{22}$. Then, the exponent in the Schönhage-Strassen modulus is

$$n = \frac{5D}{2} = 10,485,760.$$

The size of each component in the tuple is 1.25 MB. If

$$I = J = 2^{11} = 2048,$$

each task, map or reduce, processes a 2048-dimensional tuple with 2.5 GB data uniformly in all FFTs. If I and J are distinct, the machines running the tasks with smaller count have to process more data. Suppose $I = 4096$ and $J = 1024$. In the forward FFTs, each map task processes only 1.25 GB data but each reduce task has to process 5 GB data. On the other hand, each map and reduce task respectively processes 5 GB and 1.25 GB data in the backward FFT. See also Table 3 in §4.4.

4.2.3 Componentwise Multiplication

Componentwise multiplication can be easily done by a map-only job with J map tasks and zero reduce tasks. The mapper j reads $\hat{\mathbf{a}}^{(j)}$ and $\hat{\mathbf{b}}^{(j)}$, computes

$$\hat{\mathbf{p}}^{(j)} \stackrel{\text{def}}{=} \hat{\mathbf{a}}^{(j)} * \hat{\mathbf{b}}^{(j)} \tag{12}$$

over $\mathbb{Z}/(2^n+1)\mathbb{Z}$ and then writes $\hat{\mathbf{p}}^{(j)}$. Componentwise multiplication indeed can be incorporated in Algorithm 3 to eliminate the extra map-only job. We replace Step *(b.m.1)* with the following.

Algorithm 5 (Componentwise-Mult).

(b.m.1.1) read key j, value $(\hat{\mathbf{a}}^{(j)}, \hat{\mathbf{b}}^{(j)})$;

(b.m.1.2) compute $\hat{\mathbf{p}}^{(j)}$ by equation (12).

Note that a careful implementation can avoid loading both $\hat{\mathbf{a}}^{(j)}$ and $\hat{\mathbf{b}}^{(j)}$ in the memory at the same time. We may read, multiply and deallocate the digits pair-by-pair.

4.2.4 Carrying

Since $\mathbf{p} = (p_{D-1}, \ldots, p_0)$ is the cyclic convolution of $\mathbf{a}$ and $\mathbf{b}$, we have

$$0 \le p_t < 2^{2M+k} \qquad \text{for } 0 \le t < D.$$

The tuple $\mathbf{p}$ is not normalized and the carries may affect one or more following components. The *carrying* operation is to normalize $\mathbf{p}$. It is straightforward to perform carrying sequentially but non-trivial to execute it in parallel efficiently. We reduce carrying to a summation of three normalized tuples. The summation can be done by *MapReduce-Sum* given in §4.2.5.

It is obvious that SSA is inefficient for $k \ge M$, so we only have to consider $k < M$. In practice, k is much smaller than M. For $0 \le t < D$, write

$$p_t = p_{t,2}2^{2M} + p_{t,1}2^M + p_{t,0} \tag{13}$$

such that $0 \le p_{t,s} < 2^M$ for $s = 0, 1, 2$. Recall that $\ll$ and $\oplus$ are the left component-shift and the addition with carrying operators defined in equations (5) and (6). Let

$$\mathbf{p}_{[s]} \stackrel{\text{def}}{=} (p_{D-1,s}, \ldots, p_{0,s}) \ll s \tag{14}$$

be D-dimensional normalized tuples. The dropped digits are

$$p_{D-1,1} = p_{D-1,2} = p_{D-2,2} = 0.$$

If these digits are not all zeros, the integer product $p \ge 2^{2N}$, which is absurd. The sum

$$\tilde{\mathbf{p}} \stackrel{\text{def}}{=} \mathbf{p}_{[0]} \oplus \mathbf{p}_{[1]} \oplus \mathbf{p}_{[2]} \tag{15}$$

is the normalized form of $\mathbf{p}$. With equations (4), (13) and (14), it is not hard to see that the tuples

$$\mathbf{p}_{[0]}{}^{(i)}, \quad \mathbf{p}_{[1]}{}^{((i+1) \bmod I)} \quad \text{and} \quad \mathbf{p}_{[2]}{}^{((i+2) \bmod I)}$$

can be constructed by rearranging the digits of $\mathbf{p}^{(i)}$. The steps for splitting $\mathbf{p}^{(i)}$ into three tuples can be incorporated in Algorithm 4. We replace Step *(b.r.4)* with the following.

Algorithm 6 (Splitting).

(b.r.4.1) construct the tuples $\mathbf{p}_{[0]}{}^{(i)}$, $\mathbf{p}_{[1]}^{((i+1) \bmod I)}$ and $\mathbf{p}_{[2]}^{((i+2) \bmod I)}$ from $\mathbf{p}^{(i)}$ by equations (4), (13) and (14);

(b.r.4.2) write key i, value $(\mathbf{p}_{[0]}{}^{(i)}, \mathbf{p}_{[1]}{}^{((i+1) \bmod I)}, \mathbf{p}_{[2]}{}^{((i+2) \bmod I)})$.

4.2.5 MapReduce-Sum

Let $m > 1$ be an integer constant, which is much smaller then N. For $0 \le s < m$, let

$$\mathbf{y}_{[s]} \overset{\mathsf{def}}{=} (y_{D-1,s}, \dots, y_{0,s})$$

be m normalized tuples. Extending the definition (6) of addition with carrying, define *sum with carrying*

$$\mathbf{y}_{[0]} \oplus \cdots \oplus \mathbf{y}_{[m-1]} \overset{\mathsf{def}}{=} (s_{D-1} \bmod 2^M, \dots, s_0 \bmod 2^M),$$

where $s_{-1} = 0$ and, for $0 \le t < D$,

$$s_t = y_{t,0} + \cdots + y_{t,m-1} + \left\lfloor \frac{s_{t-1}}{2^M} \right\rfloor. \tag{16}$$

If $\mathbf{y}_{[s]}$'s are considered as integers $y_{[s]}$ as in §3, then $\mathbf{y}_{[0]} \oplus \cdots \oplus \mathbf{y}_{[m-1]}$ is the normalized representation of the integer sum

$$\sum_{s=0}^{m-1} y_{[s]} \pmod{2^{DM}}.$$

The classical addition algorithm is a sequential algorithm because the carries in the higher digits depend on the carries in the lower digits. It is not suitable to our environment since the entire D-dimensional tuple is in terabyte scale. Processing it with a single machine takes too much time.

An observation is that it is unnecessary to have the entire digit sequence for determining the carries $\lfloor \frac{s_{t-1}}{2^M} \rfloor$ in equation (16). The carries can be obtained using a much smaller data set. For $0 \le t < D$, let

$$\begin{aligned} z_t &\overset{\mathsf{def}}{=} y_{t,0} + \cdots + y_{t,m-1}, &(17)\\ r_t &\overset{\mathsf{def}}{=} z_t \bmod 2^M, &(18)\\ c_t &\overset{\mathsf{def}}{=} \left\lfloor \frac{z_t}{2^M} \right\rfloor &(19) \end{aligned}$$

be the componentwise sums, the remainders and the intermediate carries, respectively. Note that c_t may not equals to $\lfloor \frac{s_t}{2^M} \rfloor$ since, when c_t's are added to the digits, overflow may occur. Then carrying is required again in such case. Define the *difference*

$$d_t = \begin{cases} 2^M - r_t, & \text{if } 2^M - r_t < m; \\ m, & \text{otherwise.} \end{cases} \tag{20}$$

Notice that $0 < d_t \le m$. We also have

$$0 \le c_t \le \left\lfloor \frac{s_t}{2^M} \right\rfloor < m. \tag{21}$$

The last inequality is due to the fact that the input tuples $\mathbf{y}_{[s]}$'s are normalized. The sizes of the tuples $(c_{D-1}, \cdots, c_0)$ and $(d_{D-1}, \cdots, d_0)$ are in $O(D)$, given m a constant by assumption. Similar to [12, equation (3)], we have

$$D \le \sqrt{8N}. \tag{22}$$

The data size is reduced from $O(N)$, for classical addition algorithm, to $O(D) = O(\sqrt{N})$; i.e. from terabits to megabits. So these two tuples can be sequentially processed in a single machine efficiently. For $0 \le t < D$, let

$$c'_t = \begin{cases} c_t + 1, & \text{if } t > 0 \text{ and } c'_{t-1} \ge d_t; \\ c_t, & \text{otherwise.} \end{cases} \tag{23}$$

Then, $c'_0, \dots, c'_{D-1}$ are the carries as shown below.

Theorem 7. *Let*

$$y_t = \begin{cases} r_0, & \text{if } t = 0; \\ (r_t + c'_{t-1}) \bmod 2^M, & \text{if } 1 \le t < D. \end{cases} \tag{24}$$

Then,

$$(y_{D-1}, \dots, y_0) = \mathbf{y}_{[0]} \oplus \cdots \oplus \mathbf{y}_{[m-1]}.$$

Proof. We prove by induction that $c'_t = \lfloor \frac{s_t}{2^M} \rfloor$ for $0 \le t < D$. Clearly,

$$c'_0 = c_0 = \left\lfloor \frac{z_0}{2^M} \right\rfloor = \left\lfloor \frac{s_0}{2^M} \right\rfloor.$$

Assume $c'_t = \lfloor \frac{s_t}{2^M} \rfloor$ for $0 \le t < D - 1$. By equation (21), $c'_t < m$. Then,

$$s_{t+1} = z_{t+1} + \left\lfloor \frac{s_t}{2^M} \right\rfloor = c_{t+1}2^M + r_{t+1} + c'_t \tag{25}$$

by equations (16), (18) and (19). Since m is much smaller than 2^M by assumption, we have

$$0 \le r_{t+1} + c'_t < 2^M + m < 2^{M+1}.$$

Then,

$$\left\lfloor \frac{r_{t+1} + c'_t}{2^M} \right\rfloor = \begin{cases} 1, & \text{if } r_{t+1} + c'_t \ge 2^M; \\ 0, & \text{otherwise.} \end{cases} \tag{26}$$

Suppose $d_{t+1} = m$. By equation (20), $2^M - r_{t+1} \ge m$. We have

$$r_{t+1} + c'_t = r_{t+1} + \left\lfloor \frac{s_t}{2^M} \right\rfloor < r_{t+1} + m \le 2^M.$$

Divide equation (25) by 2^M,

$$\left\lfloor \frac{s_{t+1}}{2^M} \right\rfloor = c_{t+1} + \left\lfloor \frac{r_{t+1} + c'_t}{2^M} \right\rfloor = c_{t+1} = c'_{t+1}$$

by equations (23) and (26). Suppose $d_{t+1} < m$. Then, equation (23) becomes

$$c'_{t+1} = \begin{cases} c_{t+1} + 1, & \text{if } r_{t+1} + c'_t \ge 2^M; \\ c_{t+1}, & \text{otherwise;} \end{cases}$$

by substituting $d_{t+1} = 2^M - r_{t+1}$. By equations (26) and (25),

$$c'_{t+1} = c_{t+1} + \left\lfloor \frac{r_{t+1} + c'_t}{2^M} \right\rfloor = \left\lfloor \frac{s_{t+1}}{2^M} \right\rfloor.$$

Finally, by equation (24),

$$y_0 = r_0 \equiv z_0 \equiv s_0 \pmod{2^M}$$

and, for $1 \le t < D$,

$$y_t \equiv r_t + c'_{t-1} \equiv z_t + \left\lfloor \frac{s_{t-1}}{2^M} \right\rfloor \equiv s_t \pmod{2^M}.$$

The theorem follows. □

We present MapReduce-Sum below. All tuples are represented in the (I, J)-format described in §4.1. Two jobs are involved in MapReduce-Sum. The first job for componentwise summation (Algorithm 8) is a map-only job, which has I map and zero reduce tasks. The mapper i reads input tuples $\mathbf{y}_{[s]}{}^{(i)}$ for $0 \le s < m$, and writes the remainders $\mathbf{r}^{(i)}$, the intermediate carries $\mathbf{c}^{(i)}$ and the differences $\mathbf{d}^{(i)}$.

Algorithm 8 (Componentwise Sum, Mapper).

```
(s.m.1)  read key $i$, value $({\mathbf{y}_{[0]}}^{(i)}, \ldots, {\mathbf{y}_{[m]}}^{(i)})$;
(s.m.2)  compute $\mathbf{z}^{(i)}$ by equation (17);
(s.m.3)  compute $\mathbf{r}^{(i)}$ by equation (18);
(s.m.4)  compute $\mathbf{c}^{(i)}$ by equation (19);
(s.m.5)  compute $\mathbf{d}^{(i)}$ by equation (20);
(s.m.6)  write key $i$, value $(\mathbf{r}^{(i)}, \mathbf{c}^{(i)}, \mathbf{d}^{(i)})$.
```

The second job for carrying (Algorithms 9 and 10) has a single map task and I reduce tasks. The mapper reads all the intermediate carries and all the differences, and then evaluates the actual carries. The reducers receive the actual carries from the mapper, read the corresponding remainders and then compute the final output. For single-map jobs, we may use the *null key*, denoted by $\emptyset$, as the input key.

Algorithm 9 (Carrying, Mapper).

```
(c.m.1)  read key $\emptyset$, value $\left\{(i, \mathbf{c}^{(i)}, \mathbf{d}^{(i)}) : 0 \le i < I\right\}$;
(c.m.2)  compute $\mathbf{c}'$ by equation (23);
(c.m.3)  for $0 \le i < I$, emit key $i$, value $\mathbf{c}'^{((i-1) \bmod I)}$.
```

Algorithm 10 (Carrying, Reducer).

```
(c.r.1)  receive key $i$, singleton list $[\mathbf{c}'^{((i-1) \bmod I)}]$;
(c.r.2)  read $\mathbf{r}^{(i)}$;
(c.r.3)  compute $\mathbf{y}^{(i)}$ by equation (24);
(c.r.4)  write key $i$, value $\mathbf{y}^{(i)}$.
```

4.3 Summary

The input of MapReduce-SSA is two integers a and b, which are represented as normalized tuples $\mathbf{a}$ and $\mathbf{b}$ in (I, J)-format. The output is also in (I, J)-format, a normalized tuple $\tilde{\mathbf{p}}$, which represents an integer p such that $p = ab$. MapReduce-SSA has the following sequence of jobs:

J1: two concurrent forward FFT jobs (Alg. 1 & 2);

J2: a backward FFT job with componentwise multiplication and splitting (Alg. 3 with 5, & 4 with 6);

J3: a componentwise summation job (Alg. 8);

J4: a carrying job (Alg. 9 & 10).

4.4 Choosing Parameters

In SSA, once D, the dimension of the FFTs, is fixed, all other variables, including the exponent n in the Schönhage-Strassen modulus $2^n + 1$, are fixed consequently. The standard choice is $D \approx \sqrt{N}$.

In MapReduce-SSA, there is an additional parameter I, defined in equation (3). It determines the number of sequences in the integer representation, the numbers of tasks in the jobs and the memory required in each task. Therefore, the choice of I depends on the cluster capacity, i.e. the numbers of map and reduce tasks supported, and the available memory in each machine.

Denote the number of map and reduce tasks in the forward FFTs by $N_{f,m}$ and $N_{f,r}$, and the number of map and reduce tasks in the backward FFT by $N_{b,m}$ and $N_{b,r}$. Then,

$$N_{f,m} = N_{b,r} = I \qquad \text{and} \qquad N_{f,r} = N_{b,m} = J.$$

Excluding the memory overhead from the system software and the temporary variables, let $M_{f,m}$ and $M_{f,r}$ be the number of bits of the required memory in map and reduce tasks in forward FFT, and $M_{b,m}$ and $M_{b,r}$ be the number of bits of the required memory in map and reduce tasks in backward FFT. Then,

$$M_{f,m} = M_{b,r} = Jn \qquad \text{and} \qquad M_{f,r} = M_{b,m} = In.$$

We always choose $I \le J$ so that $M_{b,m} = In \le Jn$, in order to have more memory available for running Algorithm 5 in the map tasks of the backward FFT. In Tables 2, 3 and 4, we show some possible choices of the parameters D and I, and the corresponding values of some other variables for $N = 2^{40}$, 2^{43} and 2^{46}. The memory requirement is just 0.56 GB per task for $N = 2^{40}$. For $N = 2^{46}$, it requires 12 GB per task.

$N = 2^{40}$	J	n	In	Jn
$D = 2^{19}$, $I = 2^{9}$	2^{10}	$16.5D$	0.52GB	1.03GB
$D = 2^{20}$, $I = 2^{10}$	2^{10}	$4.5D$	0.56GB	0.56GB
$D = 2^{21}$, $I = 2^{10}$	2^{11}	$1.5D$	0.38GB	0.75GB
$D = 2^{22}$, $I = 2^{11}$	2^{11}	$0.5D$	0.50GB	0.50GB

Table 2: Possible parameters for $N = 2^{40}$

$N = 2^{43}$	J	n	In	Jn
$D = 2^{20}$, $I = 2^{10}$	2^{10}	$32.5D$	4.06GB	4.06GB
$D = 2^{21}$, $I = 2^{10}$	2^{11}	$8.5D$	2.13GB	4.25GB
$D = 2^{22}$, $I = 2^{11}$	2^{11}	$2.5D$	2.50GB	2.50GB
$D = 2^{23}$, $I = 2^{11}$	2^{12}	D	2.00GB	4.00GB

Table 3: Possible parameters for $N = 2^{43}$

$N = 2^{46}$	J	n	In	Jn
$D = 2^{22}$, $I = 2^{11}$	2^{11}	$16.5D$	16.5GB	16.5GB
$D = 2^{23}$, $I = 2^{11}$	2^{12}	$4.5D$	9.0GB	18.0GB
$D = 2^{24}$, $I = 2^{12}$	2^{12}	$1.5D$	12.0GB	12.0GB
$D = 2^{25}$, $I = 2^{12}$	2^{13}	$0.5D$	8.0GB	16.0GB

Table 4: Possible parameters for $N = 2^{46}$

4.5 A Prototype Implementation

The program *DistMpMult* is a prototype implementation, which includes *DistFft*, *DistCompSum* and *DistCarrying* as subroutines. DistFft implements Algorithms 1, 2, 3 and 4 with Algorithms 5 and 6, while DistCompSum implements Algorithm 8 and DistCarrying implements Algorithms 9 and 10. We have verified the correctness of the implementation by comparing the outputs from DistMpMult to the outputs of the GMP library.

Since DistFft is notably more complicated than DistCompSum and DistCarrying, we have measured the performance of DistFft on a 1350-node cluster. Each node has 6 GB memory, and supports 2 map tasks and 1 reduce task. The cluster is installed with Apache Hadoop version 0.20. It is a shared cluster, which is used by various research projects in Yahoo!. Figure 3 shows a graph of the elapsed time t against the input integer size N in base-2 logarithmic scales. The elapsed time includes the computation of two parallel forward FFTs and then a backward FFT. From the range $2^{32} \le N \le 2^{36}$, the elapsed time fluctuates between 2 to 4 minutes because of the system overheads. The utilization is extremely low in these cases. From the range $2^{36} \le N \le 2^{40}$, the graph is almost linear with slope $\approx 0.8 < 1$, which indicates that the cluster is still underutilized since FFT is an essentially linear time algorithm. Unfortunately, the 1350-node cluster is unable to run DistFft for $N \ge 2^{41}$ due to the memory limitation configured in the cluster, so that it limits the aggregated memory usage of individual jobs.

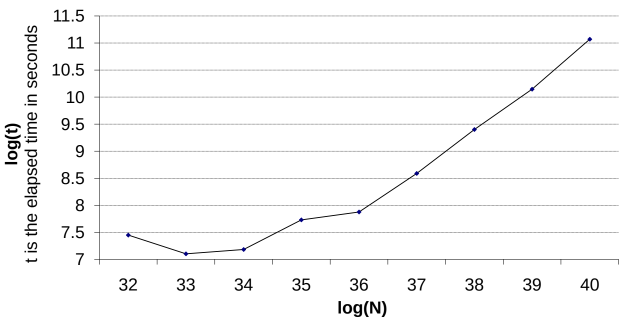


Figure 3: Actual running time of DistFft on a 1350-node cluster

For $N = 2^{40}$, we choose $D = 2^{20}$ and $I = 2^{10}$. Then, $J = I$ and

$$n = 4.5D = 4,718,592$$

as shown in Table 2. In a forward FFT job, the average durations of a map, a shuffle and a reduce task are 157, 223 and 90 seconds, respectively. Although the total is only 7.83 minutes, the the job duration is 15.38 minutes due to scheduling and other overheads. In a backward FFT job, the average durations for a map, a shuffle and a reduce task are 251, 483 and 77 seconds, and the job duration is 18.17 minutes. The itemized details are shown in Table 5. It shows that the computation is I/O bound. The CPU intensive computation only contributes 10% to 15% of the elapsed time excluding the system overheads.

Since there are already benchmarks showing that Hadoop is able to scale adequately on manipulating 100-TB data on a large cluster [14], we estimate that DistFft will be able to process integers with $N = 2^{46}$ in some suitable clusters. Such integers have 21 trillion decimal digits and require 8 TB space. A suitable cluster may consists of 4200 nodes with 16 GB memory in each machine or 2100 nodes with 32 GB memory in each machine according to the parameters suggested in Table 4.

$N = 2^{40}$		Step	Duration
Forward FFT	Mapper (Alg. 1)	*(f.m.1)*	12s
		(f.m.2)	24s
		(f.m.3)	1s
		(f.m.4)	120s
		Total	157s
	Shuffle		223s
	Reducer (Alg. 2)	*(f.r.1)*	19s
		(f.r.2)	22s
		(f.r.3)	49s
		Total	90s
Backward FFT	Mapper (Alg. 3)	*(b.m.1)*	89s
		(b.m.2)	70s
		(b.m.3)	2s
		(b.m.4)	90s
		Total	251s
	Shuffle		483s
	Reducer (Alg. 4)	*(b.r.1)*	20s
		(b.r.2)	20s
		(b.r.3)	24s
		(b.r.4)	13s
		Total	77s

Table 5: Average duration for each step in MapReduce-FFT with $N = 2^{40}$

5. CONCLUSION

An integer multiplication algorithm, *MapReduce-SSA*, is designed for multiplying integers in terabit scale on clusters with many commodity machines. It employs the ideas from Schönhage-Strassen algorithm in the MapReduce model. We first introduce a data model which allows efficiently accessing terabit integers in parallel. We show *forward/backward MapReduce-FFT* for calculating DFT and the inverse DFT, and describe how to perform componentwise multiplications and carrying operations. Componentwise multiplication is simply incorporated in the backward FFT step. The carrying operation is reduced to a summation, which is computed by *MapReduce-Sum*. A MapReduce-SSA computation involves five jobs — two concurrent jobs for two forward FFTs, a job for a backward FFT, and two jobs for carrying. In addition, we discuss the choices of the parameters and the impacts on cluster capacity and memory requirements.

We have developed a program, *DistMpMult*, which is a prototype implementation of the entire MapReduce-SSA algorithm. It includes *DistFft*, *DistCompSum* and *DistCarrying* as subroutines for calculating DFT/inverse DFT, componentwise summation and carrying, respectively. All routines run on Apache Hadoop clusters. The correctness of the implementation has been verified by comparing the outputs from DistMpMult to the outputs of the GMP library. Our experiments demonstrate that DistFft is able to process terabit integers efficiently using a 1350-node cluster with 6 GB memory in each machine. With the supporting data from benchmarks on Hadoop, we estimate that DistFft is able to process integers with 8 TB in size, or, equivalently, 21 trillion decimal digits, on some large clusters.

6. REFERENCES

[1] F. Bellard. Pi computation record, 2009. `http://bellard.org/pi/pi2700e9/announce.html`.

[2] F. Bellard. Tachuspi, 2009. `http://bellard.org/pi/pi2700e9/tpi.html`.

[3] D. J. Bernstein. Multidigit multiplication for mathematicians, 2001. Available at `http://cr.yp.to/papers/m3.pdf`.

[4] D. J. Bernstein. Fast multiplication and its applications. Number 44 in Mathematical Sciences Research Institute Publications, pages 325–384. Cambridge University Press, 2008.

[5] J. M. Borwein, P. B. Borwein, and D. H. Bailey. Ramanujan, modular equations, and approximations to pi or how to compute one billion digits of pi. 96(3):201–219, 1989.

[6] R. P. Brent and P. Zimmermann. *Modern Computer Arithmetic*. Number 18 in Cambridge Monographs on Computational and Applied Mathematics. Cambridge University Press, Cambridge, United Kingdom, 2010.

[7] D. Chudnovsky and G. Chudnovsky. The computation of classical constants. In *Proc. Nat. Acad. Sci. USA*, volume 86, pages 8178–8182. Academic Press, 1989.

[8] R. Crandall and C. Pomerance. *Prime Numbers: A Computational Perspective*. Springer-Verlag, New York, NY, 2001.

[9] A. De, P. P. Kurur, C. Saha, and R. Saptharishi. Fast integer multiplication using modular arithmetic. In *Proceedings of the 40th annual ACM Symposium on Theory of Computing*, pages 499–506. ACM, 2008.

[10] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 137–150. USENIX Association, 2004.

[11] M. Fürer. Faster integer multiplication. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 57–66. ACM, 2007.

[12] P. Gaudry, A. Kruppa, and P. Zimmermann. A GMP-based implementation of Schönhage-Strassen's large integer multiplication algorithm. In *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation*, pages 167–174. ACM, 2007.

[13] D. E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley, Reading, MA, 1997.

[14] O. O'Malley and A. C. Murthy. Winning a 60 second dash with a yellow elephant, 2009. Available at `http://sortbenchmark.org/Yahoo2009.pdf`.

[15] A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7:281–292, 1971.

[16] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The Hadoop Distributed File System. In *26th IEEE Symposium on Massive Storage Systems and Technologies*, 2010.

[17] D. Takahashi, 2010. Personal communication.

[18] D. Takahashi. Parallel implementation of multiple-precision arithmetic and 2,576,980,370,000 decimal digits of π calculation. *Parallel Comput.*, 36:439–448, August 2010.

[19] A. J. Yee, 2010. Personal communication.

[20] A. J. Yee. y-cruncher – a multi-threaded pi-program, 2010. `http://www.numberworld.org/y-cruncher/`.

[21] A. J. Yee and S. Kondo. 5 trillion digits of pi - new world record, 2010. `http://www.numberworld.org/misc_runs/pi-5t/announce_en.html`.

An improvement in the lattice construction process of Approximate Polynomial GCD over Integers

[Extended Abstract]

Kosaku Nagasaka (Kobe University, Japan)
nagasaka@main.h.kobe-u.ac.jp

ABSTRACT

We compute an approximate greatest common divisor (GCD) of co-prime polynomials over integers by changing their coefficients slightly over integers so that the input polynomials still remain over integers. In this paper, we give an improved algorithm with a new lattice construction process by which we can restrict the range of perturbations in some cases.

Categories and Subject Descriptors

I.1 [**Symbolic and algebraic manipulation**]: Misc.

General Terms

Algorithms

Keywords

Approximate Polynomial GCD, Numerical Polynomial GCD

1. INTRODUCTION

In this paper, we improve an algorithm to compute the following approximate GCD over integers. For this problem, there are only a limited number of studies by the author[2, 3] and von zur Gathen and Shparlinski[5, 4].

Definition 1. (Approximate GCD Over Integers)
Let $f(\vec{x})$ and $g(\vec{x})$ be polynomials in variables $\vec{x} = x_1, \ldots, x_\ell$ over $\mathbb{Z}$, and let ε be a small positive integer. If they satisfy $f(\vec{x}) = t(\vec{x})h(\vec{x}) + \Delta_f(\vec{x})$, $g(\vec{x}) = s(\vec{x})h(\vec{x}) + \Delta_g(\vec{x})$ and $\varepsilon = \max\{\|\Delta_f\|, \|\Delta_g\|\}$ for some polynomials $\Delta_f, \Delta_g \in \mathbb{Z}[\vec{x}]$, then we say that the above polynomial $h(\vec{x})$ is an **approximate GCD over integers**. We also say that $t(\vec{x})$ and $s(\vec{x})$ are **approximate cofactors over integers**, and we say that their **tolerance** is ε. ($\|p\|$ denotes a suitable norm of $p(\vec{x})$.)$\triangleleft$

1.1 Approx. GCD by Lattice Basis Reduction

We review the known result[2, 3] briefly. Let $f(\vec{x})$ and $g(\vec{x})$ have total degrees $n = \mathrm{tdeg}(f)$ and $m = \mathrm{tdeg}(g)$, respectively. We call the following mapping $\mathcal{S}_r(f, g)$ the subresultant mapping of $f(\vec{x})$ and $g(\vec{x})$ of order r.

$$\mathcal{S}_r(f,g): \begin{array}{ccc} \mathcal{P}_{m-r-1} \times \mathcal{P}_{n-r-1} & \rightarrow & \mathcal{P}_{n+m-r-1} \\ (s(\vec{x}), t(\vec{x})) & \mapsto & s(\vec{x})f(\vec{x}) + t(\vec{x})g(\vec{x}) \end{array}$$

where $r = 0, \ldots, \min\{n, m\} - 1$ and $\mathcal{P}_d$ denotes the set of polynomials in variables $x_1, \ldots, x_\ell$, of total degree d. We construct the coefficient vector $\vec{p}$ of polynomial $p(\vec{x})$ by the lexicographic ascending order. To see the number of elements of a coefficient vector, we define the notation: $\beta_{d,r} = \binom{d-r+\ell}{\ell}$. The number of terms $x_1^{i_1} \cdots x_\ell^{i_\ell}$ satisfying $i_1 + \cdots + i_\ell \leq d$ is denoted by $\beta_{d,0}$. The k-th convolution matrix $C_k(f)$ is defined to satisfy $C_k(f)\vec{p} = \overrightarrow{fp}$ for any polynomial $p(\vec{x})$ of total degree $k-1$, where $\vec{p} \in \mathbb{Z}^{\beta_{k-1,0} \times 1}$ and $C_k(f) \in \mathbb{Z}^{\beta_{n+k-1,0} \times \beta_{k-1,0}}$. We have the matrix representation of the subresultant mapping: $Syl_r(f, g) = (C_{m-r}(f) \;\; C_{n-r}(g))$ of size $(\beta_{n+m-1,r}) \times (\beta_{m-1,r} + \beta_{n-1,r})$, satisfying

$$\mathcal{S}_r(f,g): \begin{array}{ccc} \mathcal{P}_{m-r-1} \times \mathcal{P}_{n-r-1} & \rightarrow & \mathcal{P}_{n+m-r-1} \\ (\;\vec{s}\;\vec{t}\;)^t & \mapsto & \overrightarrow{sf+tg}^t = Syl_r(f,g)(\;\vec{s}\;\vec{t}\;)^t. \end{array}$$

By computing $s(\vec{x})$ and $t(\vec{x})$ such that the mapping is not injective, we can compute candidates of approximate GCD. Hence, finding the approximate GCD is reduced to finding short vectors by the well-known LLL algorithm[1]. For this, we construct the lattice generated by row vectors of $\mathcal{L}(f, g, r, c)$ which is defined as the following matrix where r denotes the order of the subresultant mapping.

$$\mathcal{L}(f,g,r,c) = (E_{\beta_{n-1,r}+\beta_{m-1,r}} \mid c \cdot Syl_r(f,g)^t)$$

where E_i denotes the identity matrix of size $i \times i$ and $c \in \mathbb{Z}$. The size of $\mathcal{L}(f, g, r, c)$ is $(\beta_{n-1,r} + \beta_{m-1,r}) \times (\beta_{n-1,r} + \beta_{m-1,r} + \beta_{n+m-1,r})$. However, the short vectors found are only candidate cofactors $t(\vec{x})$ and $s(\vec{x}) \in \mathbb{Z}[\vec{x}]$ such that $s(\vec{x})f(\vec{x}) + t(\vec{x})g(\vec{x}) \approx 0$, and $f(\vec{x})$ and $g(\vec{x})$ may not be divisible by $h(\vec{x})$. To compute an approximate GCD from the candidate cofactors, we apply the LLL algorithm again to the following matrix $\mathcal{H}(f, g, r, c, t, s)$ of size $(\beta_{r+1,0} + 1) \times (\beta_{n,0} + \beta_{m,0} + \beta_{r+1,0} + 1)$.

$$\mathcal{H}(f,g,r,c,t,s) = \left(E_{\beta_{r+1,0}+1} \left| \begin{array}{cc} c \cdot \vec{f}^{\,t} & c \cdot \vec{g}^{\,t} \\ c \cdot C_{r+2}(-t)^t & c \cdot C_{r+2}(s)^t \end{array} \right. \right)$$

where $\vec{f}$ and $\vec{g}$ denote the coefficient vectors of $f(\vec{x})$ and $g(\vec{x})$, respectively.

For example, we compute an approximate GCD of the following very simple polynomials for easy understanding.

$$\begin{array}{rcl} f(x) & = & 28x^2 - x\underline{-14} & = & (7x+5)(4x-3)+1, \\ g(x) & = & 42x^2 + 65x\underline{+27} & = & (7x+5)(6x+5)+2. \end{array}$$

We construct the following matrix $\mathcal{L}(f,g,r,c)$ with $r=0$ and $c=1$, and apply the LLL algorithm to the lattice generated by row vectors of $\mathcal{L}(f,g,r,c)$.

$$\left(\begin{array}{cccc|cccc} 1&0&0&0&-14&-1&28&0\\ 0&1&0&0&0&-14&-1&28\\ 0&0&1&0&27&65&42&0\\ 0&0&0&1&0&27&65&42 \end{array}\right) \to \left(\begin{array}{cccc|cccc} \underline{-5}&\underline{-6}&\underline{-3}&\underline{4}&-11&2&0&0\\ 4&4&2&-3&-2&-11&-3&-14\\ 6&6&3&-4&-3&-3&28&0\\ 4&5&2&-3&-2&-25&-4&14 \end{array}\right).$$

We take the first row vector as candidate cofactors since the right hand part of the first row is the smallest. We construct the following matrix $\mathcal{H}(f,g,r,c,t,s)$ with $c=1$ and apply the LLL algorithm, to compute an approximate GCD.

$$\left(\begin{array}{ccc|cccccc} 1&0&0&-14&-1&28&27&65&42\\ 0&1&0&-3&4&0&5&6&0\\ 0&0&1&0&-3&4&0&5&6 \end{array}\right) \to \left(\begin{array}{ccc|cccccc} 0&1&0&-3&4&0&5&6&0\\ 0&0&1&0&-3&4&0&5&6\\ 1&\underline{-5}&\underline{-7}&1&0&0&2&0&0 \end{array}\right).$$

Hence, we get $7x+5$ as an approximate polynomial GCD over integers and $4x-3$ and $6x+5$ as approximate cofactors. We note that there are more complicated examples, some lemmas and techniques for decreasing the computing-time (see [2, 3]) though we omit them here.

2. DIGITS-WISE LATTICE

The algorithms introduced in [2, 3] work well according to the numerical experiments therein. However, they can not detect any approximate GCD of the following polynomials.

$$\begin{array}{rclcl} f(x) &=& 28x^2 - x\underline{-5} &=& (7x+5)(4x-3)+10,\\ g(x) &=& 42x^2+65x\underline{+45} &=& (7x+5)(6x+5)+20. \end{array}$$

Although this is very difficult in general since the tolerance is $10\sqrt{5}$ in 2-norm, using our digits-wise lattice we can detect it if we assume that all the coefficients have a priori error on only the limited number of digits. The above pair of polynomials is the case since they have a priori error on the second digit only. One may think that this seems a odd example. However, this is important for simplifying algebraic expressions, computing with multi-precision integers with a priori error (e.g. each element of the array has error) and so on.

We construct the following matrix similar to $\mathcal{L}(f,g,r,c)$ and consider the lattice generated by their row vectors.

$$\left(\begin{array}{cccc|cccccccc} 1&0&0&0&0&-5&0&-1&2&8&0&0\\ 0&1&0&0&0&0&0&-5&0&-1&2&8\\ 0&0&1&0&4&5&6&5&4&2&0&0\\ 0&0&0&1&0&0&4&5&6&5&4&2\\ 0&0&0&0&-1&10&0&0&0&0&0&0\\ 0&0&0&0&0&0&-1&10&0&0&0&0\\ 0&0&0&0&0&0&0&0&-1&10&0&0\\ 0&0&0&0&0&0&0&0&0&0&-1&10 \end{array}\right) \begin{array}{c} \\ \\ \\ \\ \star\\ \star\\ \star\\ \star \end{array}$$

There are two differences between this matrix and $\mathcal{L}(f,g,r,c)$: 1) each column represents only a digit of coefficients, 2) extra row vectors (last 4 rows) represent carry and borrow digits. In this case, we assume that only the second digit has a priori error hence we multiple the columns corresponding to the second digit by 100 as penalty terms. The LLL algorithm gives the following result for the lattice generated by row vectors of this scaled matrix.

$$\left(\begin{array}{cccc|cccccccc} 0&-1&2&-1&9&0&9&0&2&0&-7&0\\ \underline{-5}&\underline{-6}&\underline{-3}&\underline{4}&-11&0&2&0&0&0&0&0\\ -5&2&1&2&7&0&15&0&3&0&14&0\\ 10&0&0&0&-5&0&-1&0&28&0&0&0\\ -1&-4&-1&1&-4&0&0&100&0&-100&-7&0\\ -6&-3&-2&2&-6&0&-2&100&-12&100&0&0\\ 5&2&1&-2&2&0&-4&0&5&0&-3&200\\ -4&1&-1&1&-2&-500&-2&-100&-9&0&7&0 \end{array}\right)$$

We take the second row vector as candidate cofactors and construct the following matrix similar to $\mathcal{H}(f,g,r,c,t,s)$ as above.

$$\left(\begin{array}{ccc|cccccccccccc} 1&0&0&0&-5&0&-1&2&8&4&5&6&5&4&2\\ 0&1&0&0&-3&0&4&0&0&0&5&0&6&0&0\\ 0&0&1&0&0&0&-3&0&4&0&0&0&5&0&6\\ 0&0&0&-1&10&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&-1&10&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&-1&10&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&-1&10&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&-1&10&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&-1&10 \end{array}\right) \begin{array}{c} \\ \\ \\ \star\\ \star\\ \star\\ \star\\ \star\\ \star \end{array}$$

As above, we multiple the columns corresponding to the second digit by 100 as penalty terms. The LLL algorithm gives the following result for the lattice generated by row vectors of this scaled matrix.

$$\left(\begin{array}{ccc|cccccccccccc} 1&\underline{-5}&\underline{-7}&1&0&0&0&0&0&2&0&0&0&0&0\\ -1&5&-3&-1&0&3&0&-4&0&-2&0&-5&0&-6&0\\ -1&-5&7&2&0&-4&0&0&0&-7&0&-6&0&0&0\\ 1&-1&-2&0&-200&0&100&2&0&4&0&5&-100&3&0\\ 0&-4&3&1&200&-3&500&1&200&-2&0&-1&100&2&-200\\ 0&-4&2&1&200&-2&-200&1&-200&-2&0&-1&-400&1&200\\ 0&4&3&-1&-200&1&-300&1&200&2&0&4&-100&2&-200\\ 0&-3&0&1&-100&-1&-200&0&0&-2&500&-2&200&0&0\\ 0&0&-4&0&0&1&200&-2&400&0&0&-2&0&-3&600 \end{array}\right)$$

Hence, we get $7x+5$ as an approximate polynomial GCD over integers and $4x-3$ and $6x+5$ as approximate cofactors.

3. REMARKS

The digits-wise lattice is based on the preliminary presentation about computing approximate GCD of integers (not polynomials) by the author in Research Institute for Mathematical Sciences, Kyoto University in 2010. The example above treats only 2 digits for each coefficients. It can be used for more number of digits easily. Moreover, in this paper, we use the decimal base only but this lattice can be extended to any base numbers (e.g. computing with multi-precision erroneous integers). Theoretical treatments are now in progress.

4. REFERENCES

[1] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261(4):515–534, 1982.

[2] K. Nagasaka. Approximate polynomial gcd over integers. *ACM CCA (ISSAC 2008 Poster Session)*, 42(3):124–126, 2008.

[3] K. Nagasaka. Approximate polynomial gcd over integers. *J. Symbolic Comput.*, (accepted).

[4] J. von zur Gathen, M. Mignotte, and I. E. Shparlinski. Approximate polynomial gcd: Small degree and small height perturbations. *J. Symbolic Comput.*, 45(8):879–886, 2010.

[5] J. von zur Gathen and I. E. Shparlinski. Approximate polynomial gcd: small degree and small height perturbations. In *LATIN 2008: Theoretical informatics*, volume 4957 of *Lecture Notes in Comput. Sci.*, pages 276–283. Springer, Berlin, 2008.

Acknowledgments

This work was supported in part by Japanese Ministry of Education, Culture, Sports, Science and Technology under Grant-in-Aid for Young Scientists, KAKENHI (22700011).

Stokes phenomenon : graphical visualization and certified computation

F. Richard-Jung
L.J.K.
Université de Grenoble
38041 Grenoble - France
Francoise.Jung@imag.fr

ABSTRACT

We present an automatic treatment of the Stokes phenomenon for the computation and the graphical representation of solutions of linear ODEs in the neighborhood of an irregular singularity in the complex plane. More precisely, given a basis of formal solutions, our programs detect the Stokes rays, compute the associated Stokes matrices, compute the formal monodromy, combine these matrices to specify the continuation of any actual solution given by linear combination in one sector to the whole Riemann surface of the logarithm. The graphical representation permits to visualize the results and to compare them with the use of convergent series.
In the last part of the paper we report on a first attempt to obtain certified values of the Stokes constants.

Categories and Subject Descriptors

G.1.7 [**Numerical Analysis**]: Ordinary Differential Equations—*Convergence and stability*; I.1.2 [**Symbolic and Algebraic Manipulation**]: Algorithms—*Algebraic algorithms*

General Terms

Algorithms, Theory

Keywords

Ordinary Differential Equations, Stokes matrices, graphical visualization, Computer Algebra.

1. INTRODUCTION

This paper deals with the computation and the graphical representation of complex functions of the complex variable, solutions of linear differential equations with polynomial coefficients.
Since the 80's, algorithms of Computer Algebra have been developed to find a basis of formal solutions of these equations in the neighborhood of singularities [2]. If the singularity is irregular, these solutions contain in general divergent series, but in particular thanks to the theory of Gevrey asymptotics and multisummability [8, 9, 10, 11], it is possible to associate to each divergent series appearing in a formal solution an analytic function, asymptotic to the series in some sector. In case of k-summability, this leads to effective algorithms of summation [19, 20], and so we can compute actual solutions of the ODE on **sectors** (whose vertex is the singular point). In order to continue one particular solution on a large sector, beyond the critical directions, a control of the Stokes phenomenon is necessary.
The sectors in which these actual solutions are defined are overlapping one another, in such a way that the same basis of formal solutions gives rise to two basis of actual solutions on the intersection of two adjacent sectors. The line, which is the bissector of the intersection, is called a Stokes ray, and the matrix which express the change of basis is called the associated Stokes matrix.
It is well known that the problem of computing the entries of the Stokes matrices is closed since the publication of [22], where Écalle's accelero-summation process is used to show that the entries of the Stokes matrices may be approximated up to n decimal digits in time $O(n \log^4 n \log\log n)$. Neverthless this theoretical study does not lead yet to a practical implementation.
In [6], we have proposed a completely different approach for the numerical computation of the Stokes matrices. The algorithm is based on the analysis of the singularities of the Borel transforms of the divergent series, and needs to sum only **convergent** series. In the current version, it works under the hypotheses that the equation is of single rank k (where k is any positive rational number) and the Borel transforms do not have irregular singularities and do not have many singularities aligned on a half line issued from the origin (this last point is a technical, practical limitation, not a serious one), but it allows in the Borel plane any polar, ramified or logarithmic singularities. It is implemented in our new version of Desir package, written in Maple [18].
The objective of this paper is double.
First we want to show how to use these matrices in an automatic way to continue a particular solution on the whole Riemann surface of the logarithm near the singularity. For this purpose, we introduce and compute the matrix of formal monodromy. The graphical representation in the complex plane is used to visualize the results and some examples, classical or less studied, are treated.
Secondly we will explain how to replace the analytic continuation of convergent series, performed in [6] by the use of Padé approximants, by a certified method of summation. And so we will obtain certified Stokes constants.

2. GRAPHICAL REPRESENTATION OF COMPLEX FUNCTIONS

Along this paper, we will use a graphical tool, well adapted to represent multivalued complex functions and to compare different numerical methods used to compute them.
The first implementation of such a graphical toolbox was dedicated to solutions of linear differential equation with singularities and is described in [16]. A new version of the 2D functionalities is now written in MAPLE [17].
The principle of the representation is the following: it consists in plotting the image under the considered function f of a circle or a circular arc around the singularity, in general 0. The color is used to associate a point in the domain and its image: each point $f(x)$ is plotted with a color corresponding to the argument of x.
As the studied functions are in general multi-valued, we consider them in the neighborhood of 0 as functions on the Riemann surface of the logarithm and points in the domain are represented by their Euler coordinates, whereas the image points are computed in cartesian coordinates. So we have to manipulate two types of colored points : $(\rho, \theta, color)$ and $(x, y, color)$. Two functions `draw_rhotheta` and `draw_xy` are associated to plot lists of such points. The function `create_circle`(ρ, θ^-, θ^+, `nbpoints`) creates an arc of circle around 0, of radius ρ, as a list of `nbpoints` of modulus ρ and argument between θ^- and θ^+.

Example: we consider the special Bessel function J_ν, with $\nu = \frac{1}{4}$. This function is known by MAPLE and can be computed in the complex plane, so that 0 is a branch point and the negative real axis is the branch cut. The values on the branch cut are assigned such that the function is continuous in the direction of increasing argument (equivalently, from above).

```
> circ1:=create_circle(2.,-3.1,Pi,50):

> draw_rhotheta(circ1,"Domain");
```

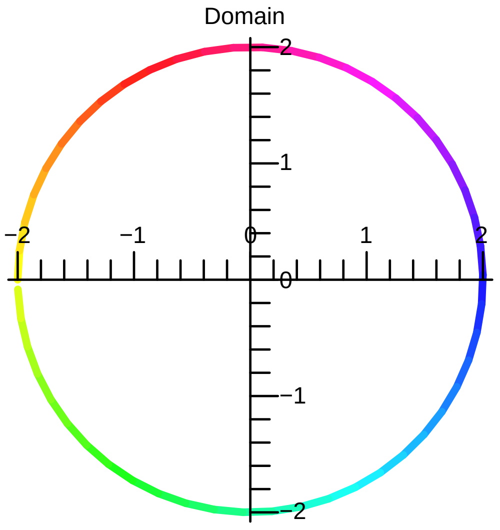


Figure 1: Domain (circle of radius 2.)

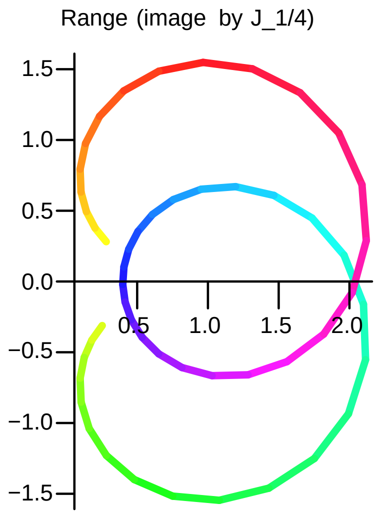


Figure 2: Range (image by $J_{1/4}$)

```
> seq1:=map(proc(pp) local x,y;
  x:=pp[1]*(cos(pp[2])+I*sin(pp[2]));
  y:=BesselJ(nu,x);
  [Re(y),Im(y),pp[3]] end proc,circ1):

> draw_xy(seq1,"Range (image by J_1/4)");
```

In fact, this function can be extended by analytic continuation on the whole Riemann surface of the logarithm. It is solution of the linear ODE

$$x^2y''(x) + xy'(x) + (x^2 - \nu^2)y(x) = 0. \qquad (1)$$

At the origin, this equation admits a basis of solutions of the form:

$$f_1(x) = x^{-1/4}(1 - \frac{3}{2}x^2 + O(x^3)),$$

$$f_2(x) = x^{1/4}(1 - \frac{1}{5}x^2 + O(x^3)).$$

It is well known that the series f_1 and f_2 are convergent (and their convergence radii are infinite).
In this basis, we express $J_\nu = \frac{1}{2^\nu\Gamma(\nu+1)}f_2$. Such a relation can be found either in [1], or by the series expansion at the origin of J_ν in MAPLE.
This expression proves that $J_{\frac{1}{4}}(xe^{2\mathbf{i}\pi}) = \mathbf{i}J_{\frac{1}{4}}(x)$.
So we obtain (Fig. 3) the image of the arc of circle of radius 2., the argument describing the range $[-2\pi, 4\pi]$, with the following commands:

```
> circ2:=create_circle(2.,-2*Pi,4*Pi,50):

> Order:=21:
> seq2:=compute_solution(sol0,
  <0,1/(2^nu*GAMMA(nu+1))>,circ2,
  linestyle=2):

> draw_xy([op(seq1),op(seq2)],"Turning 3 times"):
```

Here we use the function
`compute_solution(sol,combli,circ,options)`: it computes the image of the list of points `circ` by the function defined as the fixed linear combination `combli` of the solutions in `sol` (basis of solutions obtained by the software DESIR, [14, 18]). The last parameter `options` is optional: it permits to change the line style or the line thickness. If the series appearing in

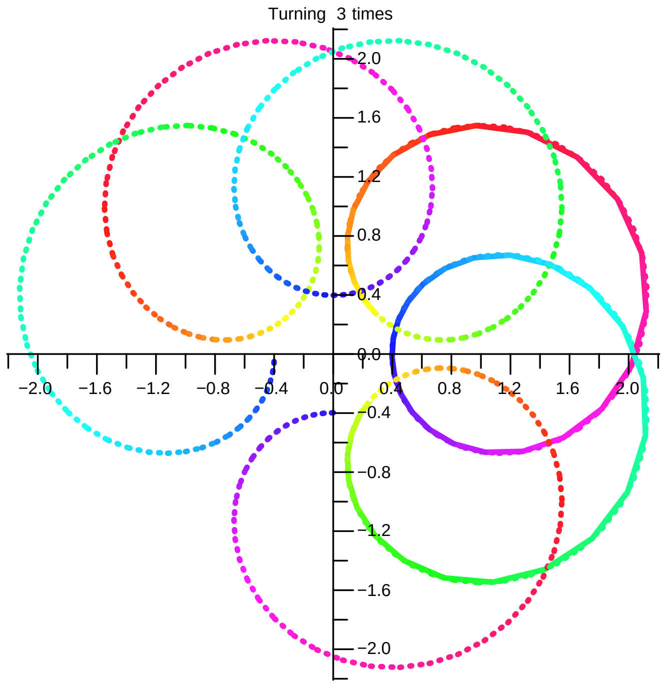


Figure 3: Image of a multiple arc of circle

`sol` are convergent, they are truncated at the order fixed by the MAPLE global variable `Order`.
So, on this figure, the series f_2 is truncated at order 21 and the image is plotted with dotted line. The curve obtained is superposed with the one obtained with the internal BesselJ function of MAPLE on the first circle. This permits to conclude that the result is the same for $|arg(x)| < \pi$.
Of course, the "complete" image would be obtained by a rotation of 4 turns around the origin.

3. STOKES PHENOMENON: THE PROBLEM

Let us explain this well-known phenomenon in the case of a linear differential equation of rank one at the origin.
Let (E) be such a linear homogeneous ODE, with polynomial coefficients, of order d. Suppose that it has an irregular singularity of rank one at 0 and that a basis of formal solution is $\hat{y}_j(x) = exp(\frac{\omega_j}{x})x^{\lambda_j}\hat{\varphi}_j(x)$, $j = 1, \ldots, d$, $\omega_j, \lambda_j \in \mathbb{C}$ and $\hat{\varphi}_j \in x\mathbb{C}[[x]]$.
Consider the series $\hat{\varphi}_i$. It is solution of a linear ODE (E_1) with polynomial coefficients, whose other solutions are $exp(\frac{\omega_j - \omega_i}{x})x^{\lambda_j - \lambda_i}\hat{\varphi}_j(x)$. The Newton polygon of this equation (E_1) has a unique edge of non null slope (namely 1), so $\hat{\varphi}_i$ is 1-summable. Its Borel transform $\mathcal{B}\hat{\varphi}_i$ is convergent in the neighborhood of the origin and the singularities of $B\hat{\varphi}_i$ are among $\omega_i - \omega_j$. This means precisely that $\hat{\varphi}_i$ is 1-summable in all directions, but the singular directions $d_{\theta_j}; \theta_j = arg(\omega_i - \omega_j)$.
Let d_θ be a nonsingular direction. The Laplace transform of $\mathcal{B}(\hat{\varphi}_i)$

$$\varphi_i(x) = \int_{d_\theta} exp(\frac{-\zeta}{x})\mathcal{B}\hat{\varphi}_i(\zeta)d\zeta$$

defines an analytic function in the sector $|arg(x) - \theta| < \frac{\pi}{2}$[1]. The product $y_i(x) = exp(\frac{\omega_i}{x})x^{\lambda_i}\varphi_i(x)$ is then an actual solution of the initial ODE (E) in the same sector.
By moving the direction of summation θ between two consecutive singular directions θ_{k-1} and θ_k, we obtain an analytic continuation on the sector $\theta_{k-1} - \frac{\pi}{2} < arg(x) < \theta_k + \frac{\pi}{2}$. The function so obtained is defined as the sum of $\hat{\varphi}_i$ on this sector.
We can do the same with a direction of summation varying between θ_k and θ_{k+1}, but (in general) the function obtained on the sector $\theta_k - \frac{\pi}{2} < arg(x) < \theta_{k+1} + \frac{\pi}{2}$ is not an analytic continuation of the previous one.
In other words, if we are interested in the calculation of one particular solution of the ODE (E), we have to represent it as linear combination of the basis solutions in **some precise sector**, taking into account all the singular directions of the series appearing in the basis solutions. In general, this linear combination is not valid in the adjacent sectors.
We illustrate this phenomenon, known as the Stokes phenomenon, on the example of the previous section.
At infinity, the equation (1) has an irregular singularity and admits as basis of formal solutions, with $z = 1/x$:

$$\hat{g}_1(z) = \frac{e^{-\mathbf{i}/z}}{\sqrt{z}}\hat{\varphi}_1(z); \hat{\varphi}_1(z) = z\left(1 + \frac{3\mathbf{i}}{32}z - \frac{105}{2048}z^2 + O(z^3)\right);$$

$$\hat{g}_2(z) = \frac{e^{\mathbf{i}/z}}{\sqrt{z}}\hat{\varphi}_2(z); \hat{\varphi}_2(z) = z\left(1 - \frac{3\mathbf{i}}{32}z + \frac{105}{2048}z^2 + O(z^3)\right).$$

The series $\hat{\varphi}_1$ and $\hat{\varphi}_2$ are divergent, but 1-summable. Their Borel transforms are convergent, with respectively only one singularity at $2\mathbf{i}$ and $-2\mathbf{i}$. The Laplace transforms of $B\hat{\varphi}_1$ and $B\hat{\varphi}_2$ can be defined in all directions, but respectively $\mathbf{i}\mathbb{R}^+$ and $\mathbf{i}\mathbb{R}^-$. An actual solution of the differential equation can then be defined on the sector $]-\pi, \pi[$ by linear combination of the sums obtained with a direction of summation varying in $]-\frac{\pi}{2}, \frac{\pi}{2}[$. If we compute the sums with a direction of summation beyond $\frac{\pi}{2}$, and combine them with the same coefficients, we obtain an other solution, which is not an analytic continuation of the first one. We illustrate this fact on the following figure (Fig. 4).
It has been obtained with the following sequence of instructions:

```
>  circ3:=create_circle(2.,-Pi,5*Pi/4,50):
>  seq3:=compute_solution(sol_inf,
   <exp(3/8*I*Pi)/sqrt(2*Pi),
   exp(-3/8*I*Pi)/sqrt(2*Pi)>,circ3):
                    4
                    4
>  draw_xy([op(seq2),op(seq3)],"At infinity"):
```

Here `sol_inf` is the basis of solutions computed by DESIR in the neighborhood of infinity, i.e. the two solutions $\hat{g}_1$ and $\hat{g}_2$. Thus, the call to `compute_solution` sums the two series appearing in $\hat{g}_1$ and $\hat{g}_2$ and combine them with the fixed

[1]An open sector on the Riemann surface of logarithm is defined as a set $\{0 < |x| < R, \alpha < arg(x) < \beta\}$, for some fixed values $R > 0$, $\alpha, \beta \in \mathbb{R}$. In all the paper, we focus on the limits on the argument, on the aperture of the sector $\beta - \alpha$ and forget the radius R. The sentence "this function is defined on the sector $\alpha < argx < \beta$" means "there exists a radius R such that the function is defined on the sector $\{0 < |x| < R, \alpha < arg(x) < \beta\}$".

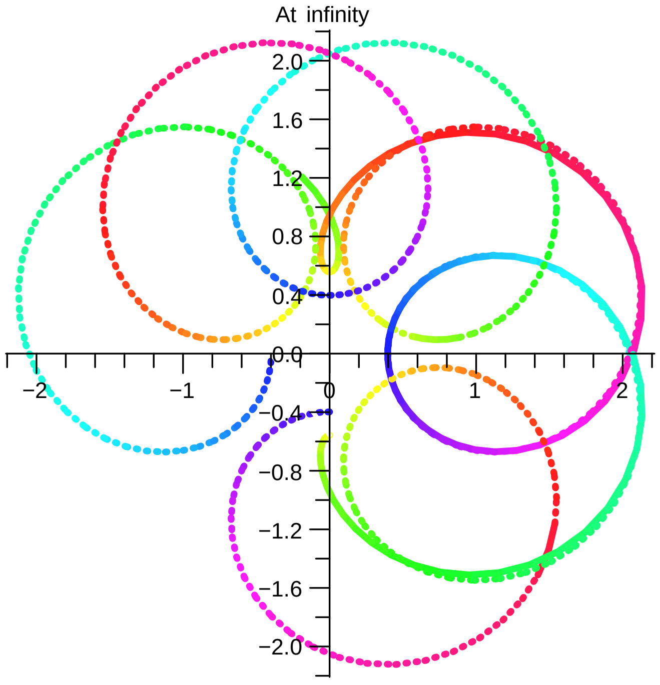


Figure 4: Image using divergent series

linear combination giving J_ν on the sector $]-\pi,\pi[$:

$$J_\nu(x)=\frac{1}{\sqrt{2\pi}}e^{\mathbf{i}(\frac{\nu}{2}+\frac{1}{4})\pi}\hat{g}_1(1/x)+\frac{1}{\sqrt{2\pi}}e^{-\mathbf{i}(\frac{\nu}{2}+\frac{1}{4})\pi}\hat{g}_2(1/x).$$

The summation method used is the most basic one: the so called "sommation des astronomes" or summation at the smallest term [15] (here four terms are used). We think that it is suffisant to highlight our purpose. The curve thus obtained (`seq3`) is plotted in solid line. We see that it follows the "exact" curve (`seq2`) for arguments of x between $-\pi$ and π, but that it is not the case for argument of x between π and $\frac{5\pi}{4}$.

4. STOKES MATRICES: A SOLUTION

Let us return to the equation (E). Consider three consecutive singular directions $\theta_{k-1},\theta_k,\theta_{k+1}$. Let y_i^- be the actual solutions obtained by summation of φ_i with a direction of summation varying between θ_{k-1} and θ_k and y_i^+ be the actual solutions obtained by summation of φ_i with a direction of summation varying between θ_k and θ_{k+1}. The two sets of solutions are then defined on the sector $\theta_k-\frac{\pi}{2}<\arg(x)<\theta_k+\frac{\pi}{2}$. By definition, the Stokes matrix associated to the Stokes line θ_k is the transition matrix S^{θ_k} which express the change of basis from the basis y_i^- to the basis y_i^+. In this way, knowing the coordinates of a particular solution in the basis y_i^- (i.e. on the sector $]\theta_{k-1}-\frac{\pi}{2},\theta_k+\frac{\pi}{2}[$), we can transport them into the basis y_i^+ (i.e. into the sector $]\theta_k-\frac{\pi}{2},\theta_{k+1}+\frac{\pi}{2}[$).
The difficult point is to compute these matrices.
Some computations have been made "by hand" on classical examples (Airy equation [16], hypergeometric confluent equation [10], confluent generalized hypergeometric equations [3]). In [22], Écalle's accelero-summation process is used in all generality to show that the entries of the Stokes matrices may be approximated up to n decimal digits in time $O(n\log^4 n\log\log n)$. But this theoretical study does not lead yet to a practical implementation.
We proposed in [6] an alternative approach, which is valid in the current version in more restricted cases, but which leads to explicit and completely automatic calculations. Indeed, the Stokes constants can be numerically computed by the analysis of the singularities of the Borel transform of the divergent series. Using formal and numerical tools in the Borel plane, introduced in [7] and [5], we showed how to obtain the Stokes matrices for a large class of differential equations. We refer to this reference for the details.
This leads to the function `StokesMatrices`, written in MAPLE. In the current version, it takes two arguments: a list (basis) of formal solutions, and a list of two integers, fixing the order of the Padé approximants used to sum the convergent series appearing in the process. The output is a description of the Stokes phenomenon as a list of objects $[arg,M\_arg]$: arg is the argument of a Stokes ray and M_arg is the corresponding Stokes matrix.
For example, for the Bessel equation, the result is the following:

$$\left[[-\pi/2,\begin{bmatrix}1 & 0\\ 6.5\,10^{-11}-1.414213562\,\mathbf{i} & 1\end{bmatrix}],\right.$$

$$\left.[\pi/2,\begin{bmatrix}1 & -6.5\,10^{-11}-1.414213562\,\mathbf{i}\\ 0 & 1\end{bmatrix}]\right].$$

On this elementary example, the result is known exactly: the Stokes constant is $-\sqrt{2}\mathbf{i}$ [21].
In this paper, our goal is to show how these matrices can be used, in combination with others – the matrix of formal monodromy and the matrices of actual monodromy – to compute any particular solution on $\mathbb{C}_\bullet$, the whole Riemann surface of logarithm. And particularly, that it can now be done in an easy manner by a user, not expert in divergent series, Stokes phenomenon or Ecalle's alien calculus.
In this paper, we start from the following data: a list of increasing arguments in $[-\pi,\pi[$ defining the critical directions $\theta_1,\ldots,\theta_p$ and the corresponding Stokes matrices computed relatively to the formal basis $\hat{y}_i$. We put $\theta_0=\theta_p-2\pi$ and $\theta_{p+1}=\theta_1+2\pi$. Implicitely, this defines the following sectors of summation: $]\theta_0,\theta_1[,\ ]\theta_1,\theta_2[,\ldots,]\theta_p,\theta_{p+1}[$, numbered respectively $0,\ 1,\ldots,p$. Then we have defined a function (in MAPLE) which calculates automatically, for a particular solution f, a suitable linear combination, depending on the sector.

For this purpose, we need to define and calculate the matrices of formal and actual monodromy.

4.1 Formal monodromy and actual monodromy

In this section, we do no assumption on the formal solutions constituting the basis. They can be of the most general form:

$$e^{Q(1/t)}t^\lambda\hat{\varphi}(t),\hat{\varphi}\in\mathbb{C}[[t]][\log t],x=t^n,n\in\mathbb{N}^*.$$

Let $(\hat{y}_1,\ldots,\hat{y}_d)$ be a basis of such formal solutions. By definition, the formal monodromy relative to this basis is the matrix $\hat{M}$ (of order d and with complex elements) such that

$$\left(\hat{y}_1(xe^{2\mathbf{i}\pi}),\ldots,\hat{y}_d(xe^{2\mathbf{i}\pi})\right)=(\hat{y}_1(x),\ldots,\hat{y}_d(x))\,\hat{M}.$$

Let $(y_1, \dots, y_d)$ a basis of analytic solutions of (E). The actual monodromy relative to this basis is the matrix M (of order d and with complex elements) such that

$$\left(y_1(xe^{2\mathbf{i}\pi}), \dots, y_d(xe^{2\mathbf{i}\pi})\right) = (y_1(x), \dots, y_d(x))\, M.$$

The knowledge of the formal monodromy and of the Stokes matrices permits to compute the matrices of actual monodromy relative to the basis obtained by summation in all the sectors $]\theta_j, \theta_{j+1}[, j = 0 \dots, p$. Let $(y_1^j, \dots, y_d^j)$ the actual solutions obtained by summation with a direction of summation varying in the sector numbered j and their analytic continuation on $\mathbb{C}_\bullet$. By definition, in the sector $]\theta_1 - \frac{\pi}{2}, \theta_1 + \frac{\pi}{2}]$, we have:

$$\left(y_1^0(x), \dots, y_d^0(x)\right) = \left(y_1^1(x), \dots, y_d^1(x)\right) S^{\theta_1}.$$

On the sector $]\theta_p - \frac{\pi}{2}, \theta_p + \frac{\pi}{2}]$, we have:

$$\left(y_1^0(x), \dots, y_d^0(x)\right) = (y_1^p(x), \dots, y_d^p(x))\, S^{\theta_p} \dots S^{\theta_1}.$$

As $(y_1^p(x), \dots, y_d^p(x)) = \left(y_1^0(xe^{-2i\pi}), \dots, y_d^0(xe^{-2i\pi})\right) \hat{M}$, the monodromy relative to the basis $(y_1^0, \dots, y_d^0)$ is the product

$$\hat{M} S^{\theta_p} \dots S^{\theta_1}.$$

For example, for the Bessel equation, we have:

- near the origin, the formal monodromy in the basis (f_1, f_2) is
$$\hat{M}_0 = \begin{pmatrix} -\mathbf{i} & 0 \\ 0 & \mathbf{i} \end{pmatrix}.$$
But f_1 and f_2 are convergent series, so they define analytic functions on $\mathbb{C}_\bullet$. In the corresponding basis (f_1, f_2), the actual monodromy and the formal monodromy are the same: $M_0 = \hat{M}_0$.

- near infinity, the formal monodromy in the basis $(\hat{g}_1, \hat{g}_2)$ is
$$\hat{M}_\infty = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}.$$
The matrix of monodromy in the basis (g_1^0, g_2^0) is $M_\infty^0 = \hat{M}_\infty S^{\frac{\pi}{2}} S^{\frac{-\pi}{2}}$. The matrix of monodromy in the basis (g_1^1, g_2^1) is
$$M_\infty^1 = \hat{M}_\infty S^{\frac{3\pi}{2}} S^{\frac{\pi}{2}} = S^{\frac{-\pi}{2}} \hat{M}_\infty S^{\frac{\pi}{2}}.$$
The matrix of monodromy in the basis (g_1^2, g_2^2) is
$$M_\infty^2 = \hat{M}_\infty S^{\frac{5\pi}{2}} S^{\frac{3\pi}{2}} = S^{\frac{\pi}{2}} S^{\frac{-\pi}{2}} \hat{M}_\infty.$$

4.2 A variable linear combination on $\mathbb{C}_\bullet$

Precisely, we define the function
`combli_variable(nsector, combli, Monodromy, MStokes)`.
Input:

- `nsector` is an integer, defining a sector of summation (as explained above);

- `combli` is a vector (of dimension d), it represents the components of the studied solution f in the basis $(y_1^{\mathtt{nsector}}, \dots, y_d^{\mathtt{nsector}})$;

- `Monodromy` is the matrix of formal monodromy;

- `MStokes` is the list $[[\theta_j, S^{\theta_j}]]$.

Output: a function (denoted `comb_var` below), which has one argument θ and returns a vector: the coefficients of a linear combination, which permits to calculate f by summation of the $\hat{\varphi}_j$'s in the direction θ.

The definition of this function is done in two steps:

1. First, we compute for each sector 0, 1, …, p two data: the linear combination $combli_j$ valid on $]\theta_j - \frac{\pi}{2}, \theta_{j+1} + \frac{\pi}{2}[$ (the components of f in the basis $(y_1^j, \dots, y_d^j)$) and the actual monodromy M_j in the basis $(y_1^j, \dots, y_d^j)$. The vector $combli_{nsector}$ is equal to `combli` and the others are obtained by application of the Stokes matrices (if $j >$`nsector`) or their inverse (if $j <$`nsector`) to `combli`.
The matrix M_0 is computed as the product
$\hat{M} S^{\theta_p} \dots S^{\theta_1}$. For $j > 0$, the matrix M_j is computed using the same principle, and taking into account the fact that $S^{\theta_k + 2\pi} = \hat{M}^{-1} S^{\theta_k} \hat{M}$. So, for example, $M_1 = S^{\theta_1} \hat{M} S^{\theta_p} \dots S^{\theta_2}$.
This first step is independent of θ.

2. The second step depends on θ. Let $\tilde{\theta}$ and $k \in \mathbb{Z}$ be such that $\theta = \tilde{\theta} + 2k\pi$ and $-\pi < \tilde{\theta} \leq \pi$. Of course, $\tilde{\theta}$ belongs to some sector $]\theta_j, \theta_{j+1}]$. Using the first step, we find the linear combination valid on $]\theta_j - \frac{\pi}{2}, \theta_{j+1} + \frac{\pi}{2}[$, and in order to obtain a suitable linear combination for θ, we apply M_i k times to this linear combination (if $k < 0$, this means that we apply the inverse of M_i).
In other words, we apply on the sector $]\theta_j, \theta_{j+1}]$, the linear combination valid on $]\theta_j - \frac{\pi}{2}, \theta_{j+1} + \frac{\pi}{2}[$.
This linear combination is the result returned by `comb_var(`θ`)`.

Let us see the result on the Bessel function.

```
> MStokes:=StokesMatrices(sol_inf,[8,8]);
```

$$MStokes := \left[[-\pi/2, \begin{bmatrix} 1 & 0 \\ 6.5\,10^{-11} - 1.414213562\,\mathbf{i} & 1 \end{bmatrix}], [\pi/2, \begin{bmatrix} 1 & -6.5\,10^{-11} - 1.414213562\,\mathbf{i} \\ 0 & 1 \end{bmatrix}] \right]$$

```
> M_inf:=monodromy(sol_inf);
```

$$M\_inf := \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$

```
> combli_Bessel:=combli_variable(1,
  <sqrt(1/(2*Pi))*exp(I*(nu/2+1/4)*Pi),
  sqrt(1/(2*Pi))*exp(-I*(nu/2+1/4)*Pi)>,
  M_inf,MStokes):
> combli_Bessel(0):evalf(%);
```

$$\begin{bmatrix} 0.1526686012 + 0.3685746074\,i \\ 0.1526686012 - 0.3685746074\,i \end{bmatrix}$$

For 0, the result (denoted by $combli_1$) is the linear combination given on the sector 1, namely valid on $]-\pi, \pi[$.

```
> combli_Bessel(Pi/2):evalf(%);
```

$$\begin{bmatrix} 0.1526686012 + 0.3685746074\,\mathbf{i} \\ 0.1526686012 - 0.3685746074\,\mathbf{i} \end{bmatrix}$$

For $\frac{\pi}{2}$, we obtain the same result.

```
> combli_Bessel(3*Pi/4):evalf(%);
```

$$\begin{bmatrix} -0.3685746073 + 0.1526686011\,\mathbf{i} \\ 0.1526686012 - 0.3685746074\,\mathbf{i} \end{bmatrix}$$

For $\frac{3\pi}{4}$, the result is $S^{\frac{\pi}{2}} combli_1$.

```
> combli_Bessel(-Pi/2):evalf(%);
```

$$\begin{bmatrix} 0.1526686012 + 0.3685746074\,\mathbf{i} \\ -0.3685746073 - 0.1526686011\,\mathbf{i} \end{bmatrix}$$

For $-\frac{\pi}{2}$, the result is $(S^{-\frac{\pi}{2}})^{-1} combli_1$.

We are now ready to compute $J_{\frac{1}{4}}(x)$ using a combination of the solutions $\hat{g}_1$ and $\hat{g}_2$, which depends on the sector where x is. For this purpose, we call the same function `compute_solution` with a function as second argument (in place of a vector).

```
> seq4:=compute_solution(sol_inf,
  combli_Bessel,circ2):
                                4
                                4
> draw_xy([op(seq2),op(seq4)],"Bessel variable"):
```

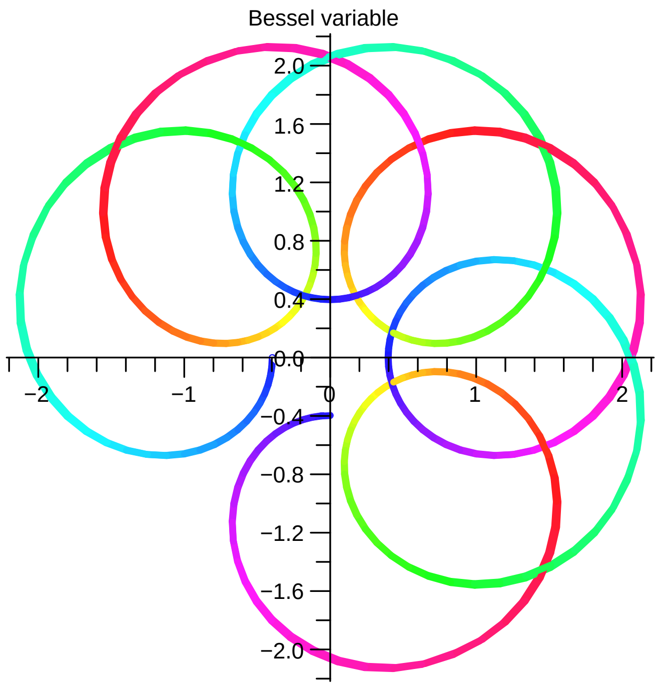


Figure 5: Image using divergent series and Stokes matrices

The validity of the computation can be seen on the fact that the two curves are exactly superposed, it is impossible to distinguish them, even using a zoom (for example near the origin, by specifying the graphical window $[-1,1] \times [-1,1]$ as optional argument in the function `draw_xy`).

```
> draw_xy([op(seq2),op(seq4)],"Zoom",
  -1,1,-1,1);
```

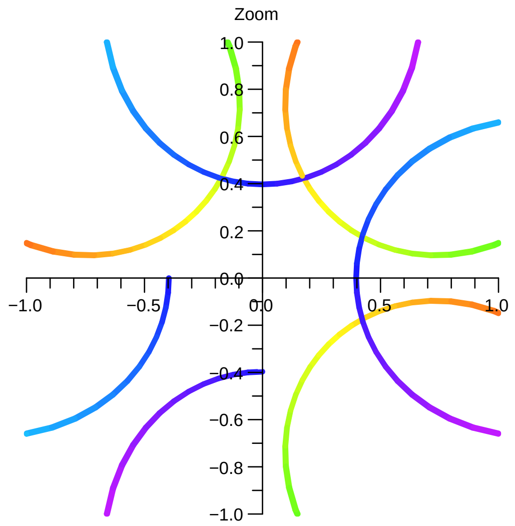


Figure 6: Zoom near the origin

Two other classical examples, the Airy equation and the exponential integrals have been studied with the same tools, but the corresponding figures can not been inserted here by lack of place. Both the Bessel function $J_{\frac{1}{4}}$ and the exponential integral present an actual monodromy. In both cases, we have compared sucessfully the result obtained with convergent series near the regular singularity (at 0 for the first one, at infinity for the second one) and the result obtained by summation of divergent 1-summable series near the irregular singularity (at infinity for the first one, at 0 for the second one). For the Airy equation, on one hand the study is facilitated by the fact that the actual monodromy is trivial, on the other hand it is complicated by the fact that the divergent series at infinity are 3/2-summable (in the variable $z = 1/x$).

We claim that our programs are **not** dedicated to special functions and run on a lot of linear differential equations. To illustrate this point, we will now run them on a "new" example (rather simple, but less studied) [[23], p 43].

5. A NEW EXAMPLE

Consider the equation

$$x^3 y''(x) - y(x) = 0. \qquad (2)$$

The origin is an irregular singularity and using the software DESIR,we compute as basis of solutions:

$$\hat{g}_1(x) = e^{\frac{-2}{\sqrt{x}}} x^{3/4} \left(1 + 3/16\sqrt{x} - \tfrac{15}{512}\,x + O\left(x^{3/2}\right)\right)$$
$$\hat{g}_2(x) = e^{\frac{2}{\sqrt{x}}} x^{3/4} \left(1 - 3/16\sqrt{x} - \tfrac{15}{512}\,x + O\left(x^{3/2}\right)\right)$$

The series appearing in $\hat{g}_1$ and $\hat{g}_2$ are 1/2 summable in the x-variable, 1-summable in the u-variable, with $u = \sqrt{x}$. The infinity point is a regular singularity. We find the following basis of solutions:

$$f_1(z) = 2 + z + 1/6\, z^2 + \frac{1}{72}\, z^3 + \frac{1}{1440}\, z^4 + O\left(z^5\right),$$

$$f_2(z) = z^{-1}\Bigg(\ln(z)\,(z + 1/2\, z^2 + 1/12\, z^3 + \frac{1}{144}\, z^4) +$$

$$1 - 3/4\,z^2 - \frac{7}{36}\,z^3 + O\left(z^4\right)\Big)$$

The series appearing in f_1 and f_2 are entire functions (the radii of convergence of the series are infinite).

We compute the Stokes matrix in the basis $(\hat{g}_1, \hat{g}_2)$,

$$\texttt{MStokes} := [[0, \begin{bmatrix} 1 & 1.\,10^{-10} + 2.0\,\mathbf{i} \\ 0 & 1 \end{bmatrix}]]$$

and the matrix of formal monodromy at 0 in the same basis:

$$\texttt{M0} := \begin{bmatrix} 0 & -\mathbf{i} \\ -\mathbf{i} & 0 \end{bmatrix}$$

So, we have all the matrices that permit to continue a particular solution around 0.
Precisely, we define the sector 0 as $]-2\pi, 0[$, and the sector 1 as $]0, 2\pi[$. A particular solution will be specified by summation in the sector 0, i.e. by the linear combination valid for $arg(x) \in]-3\pi, \pi[$[2] or by summation in the sector 1, i.e. by the linear combination valid for $arg(x) \in]-\pi, 3\pi[$.
Our goal is now to compare the results obtained with the divergent series at 0 with the results obtained with the convergent series at infinity.
For this purpose, we need to do the connection between 0 and infinity. **Only at this point, in order to control the previous results**, we will consider special functions and in particular the knowledge about the Bessel functions of integer order I and Y. Indeed:

$$F_1 = \sqrt{x}\texttt{BesselI}\left(1, 2\frac{1}{\sqrt{x}}\right) \text{ and } F_2 = \sqrt{x}\texttt{BesselY}\left(1, \frac{-2\,\mathbf{i}}{\sqrt{x}}\right)$$

are linearly independant solutions of the equation (2). A complete study of these two functions leads to the definition of a matrix of connection M between the basis (f_1, f_2) and the basis (g_1^0, g_2^0):

$$f_1 = \frac{-\mathbf{i}}{\sqrt{\pi}} g_1^0 + \frac{1}{\sqrt{\pi}} g_2^0.$$

$$f_2 = \frac{\sqrt{\pi}}{2}\left((1 - C1)g_1^0 - \mathbf{i}(1 + C1)g_2^0\right); \quad C1 = -1 + \frac{1 - 2\gamma}{\pi}.$$

This allows us to specify any particular solution expressed in the basis (f_1, f_2) as linear combination of the solutions (g_1^0, g_2^0).
Let $F = f_1 + f_2$. We represent F on the circle of radius 0.2, with arguments describing $[-3\pi, 3\pi]$ (seq5). First using the convergent series at infinity (Fig. 7):

```
> seq5F_inf:=compute_solution(sol_inf,
  <1,1>,seq5):
```

$$21$$
$$21$$

```
> draw_xy(seq5F_inf,"F with convergent series,
  3 loops around 0");
```

Then we represent the same function using the divergent series at 0 (Fig. 8). We compute the coordinates of F in the basis (g_1^0, g_2^0) and use them to specify F at 0 in the sector 0.

```
> <a,b>:=simplify(MatrixVectorMultiply(M,
  <1,1>));
```

[2]The aperture of the sectors of validity is imposed by the fact that the series are 1/2-summable.

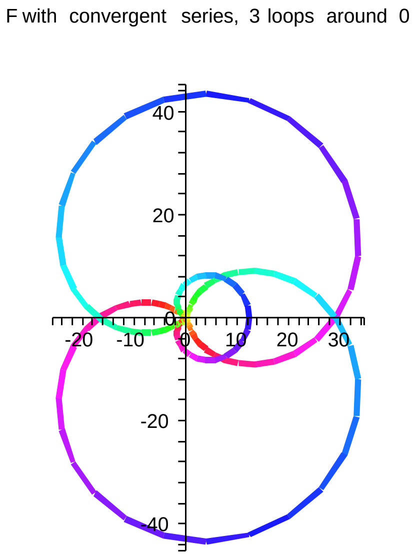


Figure 7: Image using convergent series

$$< a, b > := \begin{bmatrix} \frac{-3\,\mathbf{i} + 2\,\pi + 2\,\mathbf{i}\gamma}{2\sqrt{\pi}} \\ \frac{3 - 2\,\gamma}{2\sqrt{\pi}} \end{bmatrix}$$

```
> combli_F:=combli_variable(0,<a,b>,M0,
  MStokes):
> seq5F_0:=compute_solution(sol0,
  combli_F,seq5,linestyle=2):
```

$$8$$
$$8$$

```
> draw_xy(seq5F_0,"F with divergent series,
  3 loops around 0");
```

Finally we verify using a zoom (Fig. 9) that the two curves are merged:

```
> draw_xy([op(seq5F_0),op(seq5F)],
  "Comparison of the curves on a zoom",
  -5.,5.,-5.,5.);
```

6. CERTIFIED STOKES CONSTANTS

In this section, we will explain how to obtain certified numerical values for the Stokes constants.
The heart of the computation of the Stokes constants (with our approach) is to perform the **connection between two regular singularities**.
We consider a series $\hat{\varphi}$, solution of a linear ODE with polynomial coefficients $(E1)$ and convergent at the origin.[3]
In general, the origin is a regular singularity of the equation $(E1)$.
We consider ω a finite non null singularity of $(E1)$ and assume that

- it is a regular singularity;

[3]In the current version of the algorithm, this series is the Borel transform of a divergent series, appearing in a formal solution of the considered basis. In a futur version, it could be replaced by a series solution near a finite singularity of the Fourier transform of the initial equation.

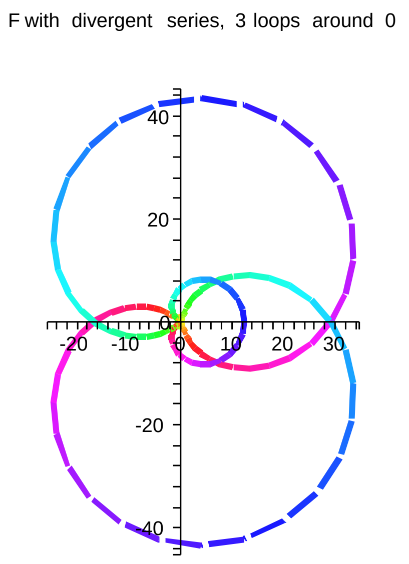


Figure 8: Image using divergent series and Stokes matrices

- a basis of solutions near ω can be obtained as $[\hat{f}(x), \hat{f}(x)\ln(x-\omega)+\hat{g}(x)]$
- the series $\hat{\varphi}$ is convergent on the disk $|x|<|\omega|$ and the series $\hat{f}$ and $\hat{g}$ are convergent on the disk $|x-\omega|<|\omega|$.

The goal is to express $\hat{\varphi}(x) = \lambda\hat{f}(x)+\mu\big(\hat{f}(x)\ln(x-\omega)+\hat{g}(x)\big)$ and to obtain a numerical approximation of the coefficients λ and μ. Indeed, the Stokes constant (to be put in the Stokes matrix associated to the Stokes ray of argument $arg(\omega)$) is exactly $2\mathbf{i}\pi\mu$.
To do this, we just have to obtain a certified evaluation of $\hat{\varphi}$, $\hat{f}$ and $\hat{g}$ at two points on the segment $[0,\omega]$ (or more generally in the intersection of the disks of convergence).
As the origin and ω are not regular points for $(E1)$, the current procedures available in GFUN[4], and more precisely in NUMGFUN [12], for the analytic continuation of holonomic functions can not be used.
Neverthless, it is possible to obtain the wished result in the following way.
Consider $\hat{\varphi}(x) = \sum_{n\geq 0}\varphi_n x^n$. The coefficients φ_n satisfy a nonsingular homogeneous recurrence equation with polynomial coefficients. Assuming that it is also reversible, from [13] we can obtain constants $A\in\mathbb{R}^+$, $\kappa\in\mathbb{Q}$, $\alpha\in\overline{Q}^{*+}$ and a function ϕ such that

$$\forall n\in\mathbb{N},\quad |\varphi_n|\leq An!^{\kappa}\alpha^n\phi(n); \tag{3}$$

with $\phi(n)=e^{o(n)}$. The corresponding algorithms are implemented in the function `bound_rec`, available in NUMGFUN.
Let v be the majorant series. Then it is possible to obtain a closed form of the tail of the series $v_{N;}(x)=\sum_{n\geq N}v_n x^n$, which gives a majoration of $|\varphi_{N;}(x)|$ by evaluating $v_{N;}(|x|)$.
Consider $\hat{f}(x)=\sum_{n\geq 0}f_n x^n$. The coefficients f_n satisfy the same type of recurrence equation and we obtain a numerical value of $\hat{f}(x)$ with a guaranted precision in the same manner.

[4]http://algo.inria.fr/libraries/papers/gfun.html

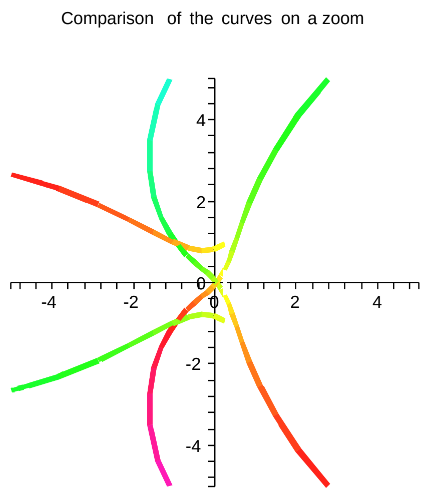


Figure 9: Zoom near the origin

The case of $\hat{g}$ is more complicated.
An attempt to treat this case is done in [12], but is not yet implemented.
The first idea was to obtain a differential equation with polynomial coefficients satisfied by the series. This could be done, knowing a differential equation satisfied by log and using the functions `'diffeq*diffeq'` and `'diffeq+diffeq'` of the package GFUN. Unfortunately, this leads to a differential equation of higher order, introduces apparent singularities, and the computations did not succeed for a simple example.
Let $\hat{g}(x)=\sum_{n\geq 0}g_n x^n$. The coefficients g_n satisfy again a recurrence equation, but this one is not homogeneous, it presents a right hand side depending on the coefficients f_n.
Precisely, we are now treating the case where the indicial equation of $(E1)$ near ω admits two roots with integer difference $\overline{k}$ (potentially null). Thus the sequence g_n is defined by its first terms $g_0,\ldots,g_{\overline{k}-1},0$ and a recurrence of the following form:

$$\sum_{j=0}^{s}p_j(n)g_{n+j} = -\sum_{j=0}^{s}p'_j(n)f_{n+j},\quad n\geq\overline{k}-s+1,$$

where p_j are polynomial in n, p'_j is the derivative of p_j and $p_s(n)$ has constant sign (it is a constant multiple c of $(n+s)(n+s-\overline{k})$).
Suppose that we have obtained a majorant sequence for f_n of the type : $A\alpha^n\phi(n)$.[5] We put $|g_n|=\alpha^n b_n$, so that

$$|c|(n+s)(n+s-\overline{k})b_{n+s}\leq\sum_{j=0}^{s-1}|p_j(n)|\alpha^{j-s}b_{n+j}$$

$$+A\sum_{j=0}^{s}|p'_j(n)|\alpha^{j-s}|\phi(n+j)|.$$

Now assume that $\phi(n)$ is a polynomial, we can bound all the modules by replacing the coefficients of the polynomials

[5]As the series $\hat{f}$ is convergent, we can assume that $\kappa\leq 0$ in the majoration (3), and replace it by 0, for simplicity. It is then possible to take κ into account, by putting $|g_n|=\alpha^n n!^{\kappa}b_n$ and obtaining a majorant sequence for b_n.

by their modules. Hence we obtain a recurrence equation with polynomial coefficients for a majorant of the sequence b_n. Finaly, from this recurrence, we can construct an homogeneous one (function `rectohomrec` of GFUN), on which we apply the previous process in order to obtain a majoration of the tail $b_{N;}(|x|)$, and so we get an evaluation of $\hat{g}(x)$ with guaranted precision.
This method has been succesfully tested on the Bessel equation. The equation $(E1)$ is of order 2, with 0 and $2\mathbf{i}$ as regular singularity. The functions $\hat{\varphi}$, $\hat{f}$ and $\hat{g}$ have been evaluated at $0.9\mathbf{i}$ and $1.1\mathbf{i}$, by summing 80 terms in the series, which guarantes a precision of 10^{-11}. So the Stokes constant is guaranted with a precision of 10^{-10}, and indeed we verify that we obtain $-\sqrt{2}\mathbf{i}$ with a precision of 10^{-12}.
Moreover this method has been successfully tested on the prolate spheroidal wave equation [4]. We can now guarantee that the Stokes constant associated to the first eigenvalue, whose numerical value is known in the tables as 0.319, is null up to a precision of 10^{-6}, but also improve the numerical value of this eigenvalue: for this new value 0.31900005514689, the Stokes constant is null up to a precision of 10^{-14}.

7. CONCLUSION

In this paper, we proved that we have now the tools to perform automatic computations with divergent series around an irregular singularity and to graphically visualize the results. So we will pursue our efforts to enlarge the class of equations, for which we are able to compute the Stokes matrices. The next step in this direction will be to treat equations which present aligned singularities in the Borel plane (for example, it is the case of the confluent generalized hypergeometric equations, for particular values of the parameters).
In the last section, we focused on the computation of the Stokes constants with guaranted precision. This question will also lead to further development in future work.

8. REFERENCES

[1] M. Abramowitz and I. Stegun. *Handbook of Mathematical Functions*. Dover Publications, New-York, 1964.

[2] J. Della Dora, C. Di Crescenzo, and E. Tournier. An algorithm to obtain formal solutions of a linear homogeneous differential equation at an irregular point. In J. Calmet, editor, *European Computer Algebra Conference, EUROCAM'82, April, 1982*, volume 144 of *Lecture Notes in Computer Science*, pages 273–280, Marseille, France, Nov. 1982. Springer.

[3] A. Duval and C. Mitschi. Matrices de Stokes et groupe de Galois des équations hypergéométriques confluentes généralisées. *Pacific Journal of Mathematics*, 138(1), 1989.

[4] F. Fauvet, J.-P. Ramis, F. Richard-Jung, and J. Thomann. Stokes phenomenom for the prolate spheroidal wave equation. *Appl. Numer. Math.*, 60(12):1309–1319, Dec. 2010.

[5] F. Fauvet, F. Richard-Jung, and J. Thomann. Algorithms for the splitting of formal series; applications to alien differential calculus. In J.-G. Dumas, editor, *Transgressive Computing 2006, April, 2006*, pages 231–246, Grenade, Espagne, Apr. 2006.

[6] F. Fauvet, F. Richard-Jung, and J. Thomann. Automatic computation of Stokes matrices. *Numer. Algorithms*, 50(2):179–213, Feb. 2009.

[7] F. Fauvet and J. Thomann. Formal and numerical calculations with resurgent functions. *Numerical Algorithms*, 40(4), Dec. 2005.

[8] M. Loday-Richaud. Introduction à la multisommabilité. *Gazette des Mathématiciens*, SMF(44), Apr. 1990.

[9] B. Malgrange. Sommation des séries divergentes. *Expo. Math.*, 13:163–222, 1995.

[10] J. Martinet and J.-P. Ramis. *Computer algebra and differential equations (E. Tournier ed.)*, chapter Théorie de Galois différentielle et resommation. Academic Press, 1987.

[11] J. Martinet and J.-P. Ramis. Elementary acceleration and multisummability. *Ann. Inst. H. Poincaré Phys.Théor.*, 54(4):331–401, 1991.

[12] M. Mezzarobba. NumGfun: a package for numerical and analytic computation with D-finite functions. In S. M. Watt, editor, *ISSAC 2010: Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation, 25-28 July 2010, Munich, Germany*, pages 139–146. ACM, 2010.

[13] M. Mezzarobba and B. Salvy. Effective bounds for P-recursive sequences. *Journal of Symbolic Computation*, 45(10):1075–1096, 2010.

[14] E. Pflügel. On the latest version of DESIR-II. *Theor. Comput. Sci.*, 187(1-2):81–86, Nov. 1997.

[15] J.-P. Ramis and R. Schäfke. Gevrey separation of fast and slow variables. *Nonlinearity*, 9(2):353–384, 1996.

[16] F. Richard-Jung. *Représentations graphiques de solutions d'équations différentielles dans le champ complexe*. PhD thesis, Université Louis Pasteur, Sept. 1988.

[17] F. Richard-Jung. An implicit differential equation with Maple and Irondel. In V. G. Ghanza, E. W. Mayr, and E. V. Vorozhtsov, editors, *7th International Workshop on Computer Algebra in Scientific Computing, CASC'04, July, 2004*, volume 4770, pages 383–397, Saint Petersbourg, Russie, 2004.

[18] F. Richard-Jung. DESIR (new version, including automatic computation of Stokes matrices). Software LJK, http://www-ljk.imag.fr/CASYS/LOGICIELS/, May 2009.

[19] J. Thomann. Resommation des séries formelles solutions d'équations différentielles linéaires ordinaires du second ordre dans le champ complexe au voisinage de singularités irrégulières. *Numer. Math.*, 58:503–535, 1990.

[20] J. Thomann. Procédés formels et numériques de sommation de séries solutions d'équations différentielles. *Expo. Math.*, 13:223–246, 1995.

[21] E. Tournier, editor. *Computer Algebra and Differential Equation, CADE'92*, volume 193 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, Apr. 1994. 267 pages.

[22] J. van der Hoeven. Efficient accelero-summation of holonomic functions. *JSC*, 42(4):389–428, 2007.

[23] R. B. White. *Asymptotic Analysis of Differential Equations*. Imperial College Press, 2005.

Extended companion matrix for approximate GCD

Paola Boito
XLIM - DMI
Université de Limoges - CNRS
123 avenue Albert Thomas
87060 Limoges CEDEX
paola.boito@unilim.fr

Olivier Ruatta
XLIM - DMI
Université de Limoges - CNRS
123 avenue Albert Thomas
87060 Limoges CEDEX
olivier.ruatta@unilim.fr

ABSTRACT

We study a variant of the univariate approximate GCD problem, where the coefficients of one polynomial $f(x)$are known exactly, whereas the coefficients of the second polynomial $g(x)$may be perturbed. Our approach relies on the properties of the matrix which describes the operator of multiplication by $g$in the quotient ring $\mathbb{C}[x]/(f)$. In particular, the structure of the null space of the multiplication matrix contains all the essential information about $\mathrm{GCD}(f,g)$. Moreover, the multiplication matrix exhibits a displacement structure that allows us to design a fast algorithm for approximate GCD computation with quadratic complexity w.r.t. polynomial degrees.

Categories and Subject Descriptors

G.0 [**Mathematics of Computing**]: General
; G.1.2 [**Mathematics of Computing**]: Numerical Analysis—*Approximation*; G.1.3 [**Mathematics of Computing**]: Numerical Analysis—*Numerical Linear Algebra*; I.1.2 [**Computing Methodologies**]: Symbolic and Algebraic Manipulations—*Algorithms*

General Terms

Theory; Algorithms

Keywords

Approximate greatest common divisor; Approximate geometry; Symbolic-numeric computation; Structured matrices

1. INTRODUCTION

The approximate polynomial greatest common divisor (denoted as AGCD) is a central object of symbolic-numeric computation. The main difficulty of the problem comes from the fact that is no universal notion of AGCD. One can find different approaches and different notions for AGCD. We will not give a review of all the existing work on this subject, but we will recall one of the most popular approaches to show how our work brings a different point of view on the problem.

The main approach to the computation of an AGCD consists in considering two univariate polynomials whose coefficients are known with uncertainty. This uncertainty can be the result of the fact that the polynomials have floating point coefficients coming from previous computation (and so are subject to round-off errors). The most frequently adopted formulation is related to semi-algebraic optimization : given $\tilde{f}$ and $\tilde{g}$ two approximate polynomials, find two polynomials f and g such that $\|\tilde{f}-f\|$ and $\|\tilde{g}-g\|$ are small (lower than a given tolerance for instance) and such that the degree of $\gcd(f,g)$ is maximal. That is, one looks for the most singular system close to the input $(\tilde{f},\tilde{g})$. An ε-gcd is obtained if the conditions $\|\tilde{f}-f\| < \varepsilon$ and $\|\tilde{g}-g\| < \varepsilon$ are satisfied. One can try to compute the tolerance on the perturbation of the input polynomial thanks to direct computation (for instance from a jump on singular values of particular matrix for instance). This last approach has received a great interest following the work of Zeng using Sylvester like matrices ([22]) following several previous work [3], [4], [7], [8], [16], [17] and [18].

Here, we consider a slightly different problem. One of the polynomials, say f, is known exactly (it is the result of an exact model) and the second one, say g, is an approximate polynomial (result of measures or previous approximation for instance). This case occurs in applications such as model checking (to compare results of an exact model and measures). There are many other instances of such a problem, such as simplification of fractions when one of the polynomial is known exactly but the other one is not.

We give a example of such a situation. When modeling an electromagnetic filter, one might want to parametrize its behavior with respect to the frequency. But one may need to do so even if there are singularities and to do so one may use Padé approximations of the electromagnetic signal at each point as a function of the frequency. In some cases of interest, one can know all the singularities and so compute an exact polynomial called characteristic. Padé approximations are computed independently for each point by a numerical process and denominators may have a non trivial gcd with the "characteristic" polynomial. The denominators are not known exactly. So, in order to identify unwanted common factors in denominators one has to compute approximate gcds between an exact and non exact polynomials.

This AGCD problem can also be interpreted as an optimization problem. Given f exactly and $\tilde{g}$ approximately, compute a polynomial g close to $\tilde{g}$ such that g has a maximal degree gcd with f. Our approach takes advantage of the asymmetry of the problem and of the structure of the quotient algebra $\mathbb{C}[x]/(f(x))$ (more accurately, of the displacement rank of the multiplication operator in this algebra). So, we address the following problem :

Problem 1 Let $f(x) \in \mathbb{C}[x]$ a given polynomial and $g(x)$ another polynomial. Find $\tilde{g}(x)$ close to $g(x)$ (in a sense that will be explained) such that $f(x)$ and $\tilde{g}(x)$ have a gcd of maximal degree.

This may be also an interesting approach when one has two polynomials, one known with high confidence and another with worse accuracy. This approach may take advantage of this asymmetry which would not be possible for classical framework based on Sylvester or Bézout matrices.

All the previous methods allow to deform the two input polynomials. They give a symmetrical role to those polynomials. If one of them is known with high confidence, the previous methods cannot take into account this information. Here, the polynomial used to construct the quotient algebra is supposed to be exact or known with higher confidence than the one use to compute the multiplication matrix. Furthermore, just as in [17] and [18] we can translate our problem to an optimization problem. This is important in order to be able to design the numerical part of our algorithm. Both the approach and the algorithm seem new since we address a new problem.

The proposed algorithm is "fast" since the exponent of its complexity is better than the classical linear algebra exponent in the degree of the input polynomials.

Organisation of the paper: The second section is devoted to some basic result on algebra needed after, the third section gives an algebraic method for gcd based on linear algebra, the fourth section recalls the Barnett's formula allowing to compute the multiplication matrix without division, the fifth gives the displacement rank structure of the multiplication matrix, the sixth describes the final algorithm and experiments before finishing with conclusions and perpectives.

2. EUCLIDIAN STRUCTURE AND QUOTIENT ALGEBRA

In this section, we recall basic algebraic results needed to understand the principle of our approach. All material in this section can be found (even in the non reduced case and in the multivariate setting) in [19] for instance.

Assume that $\mathbb{K}$ is an algebraically closed field (here we think about $\mathbb{C}$). Let $f(x)$ and $g(x) \in \mathbb{K}[x]$ and assume that $f(x) = f_d \prod_{i=1}^{d}(x-\zeta_i)$ and that $\zeta_i \neq \zeta_j$ for all $i \neq j$ in $\{1,\ldots,d\}$. Let $\mathbb{A} = \mathbb{K}[x]/(f)$ and $\pi : \mathbb{K}[x] \longrightarrow \mathbb{A}$ be the natural projection. For $i \in \{1,\ldots,d\}$, we define $L_i(x) = \frac{\prod_{j\neq i}(x-\zeta_j)}{\prod_{j\neq i}(\zeta_i-\zeta_j)}$, the ith Lagrange polynomial associated with $\{\zeta_1,\ldots,\zeta_d\}$. Clearly, since $\deg(L_i) < \deg(f)$, we have $\pi(L_i) = L_i$, for all $i \in \{1,\ldots,d\}$. Let $\mathbb{A}^* = \mathrm{Hom}_{\mathbb{K}}(\mathbb{A},\mathbb{K})$ be the usual dual space of $\mathbb{A}$. For all $i \in \{1,\ldots,d\}$, we define $\mathbf{1}_{\zeta_i} : \mathbb{A} \longrightarrow \mathbb{K}$ by $\mathbf{1}_{\zeta_i}(p) = p(\zeta_i)$ for all $p \in \mathbb{A}$. The following lemma is obvious from the definition of the polynomials L_i that for i and $j \in \{1,\ldots,d\}$, we have $L_i(\zeta_j) = \left\{\begin{array}{l} 1 \text{ if } i=j \\ 0 \text{ else}\end{array}\right.$.

This implies that the set $\{L_1,\ldots,L_d\}$ is a basis of $\mathbb{A}$. A well known fact is that the set $\{\mathbf{1}_{\zeta_1},\ldots,\mathbf{1}_{\zeta_d}\}$ form a basis $\mathbb{A}^*$ dual of the basis $\{L_1,\ldots,L_d\}$ of $\mathbb{A}$. As a corollary, we have the Lagrange interpolation formula : Each $p \in \mathbb{A}$ can be written $p(x) = \sum_{i=1}^{d}\mathbf{1}_{\zeta_i}(p)L_i(x)$. A direct consequence is that if we choose $\{L_1,\ldots,L_d\}$ as a basis of $\mathbb{A}$, for all $g \in \mathbb{K}[x]$, the remainder $\pi(g)$ resulting of the Euclidean division of g by f is given by $(g(\zeta_1),\ldots,g(\zeta_d))$ in the basis $\{L_1,\ldots,L_d\}$, i.e. $r = \sum_{i=1}^{d} g(\zeta_i)L_i(x)$. In other word, divide g by f is equivalent to evaluate g at the roots of f.

The general philosophy of this last assertion will allow us to make a lot of proof in a very simple way. For example, it is very easy to see the different operation in $\mathbb{A}$ using this representation. Let g and h be two elements in $\mathbb{A}$, then we have $g+h = \sum_{i=1}^{d}(g(\zeta_i)+h(\zeta_i)) * L_i(x)$ and $g*h = \sum_{i=0}^{d}(g(\zeta_i) * h(\zeta_i))L_i(x)$ in $\mathbb{A}$. This allows us to avoid the use of the section of the surjection π. In fact, the Lagrange polynomials $L_1,\ldots,L_d$ reveal a deeper structure on the algebra $\mathbb{A}$: The polynomials $L_1,\ldots,L_d$ are the idempotents of $\mathbb{A}$, i.e. $L_i * L_j = \left\{\begin{array}{l} L_i \text{ if } i=j \\ 0 \text{ otherwise.}\end{array}\right.$.

Thanks to this description of the quotient algebra, it is easy to derive algorithms for both polynomial solving and gcd computation even though the problems are of very different nature.

Remark that we have expressed everything in the monomial basis since it is the most widely used basis to express polynomials. We can use other bases. A particular basis is the Chebyshev basis where all results are exactly the same since it is a graded basis.

3. AN ALGEBRAIC ALGORITHM FOR GCD COMPUTATION

To first give an idea on how to exploit the section above in order to design algorithm for gcd, We recall a classical method for polynomial solving (see [6] for instance). Proofs are given for the sake of completeness and because very similar ideas will lead us to the AGCD computation.

3.1 Roots via eigenvalues

Let $f(x) = \sum_{i=0}^{d} f_i x^i \in \mathbb{C}[x]$ be a polynomial of degree d. Then we consider the matrix of the multiplication by x in $\mathbb{C}[x]/(f)$. Its matrix in the monomial basis $1,\ldots,x^{d-1}$ is the following:

$$\mathrm{Frob}(f) = \begin{pmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^{d-1}\end{pmatrix} \begin{array}{c} \begin{pmatrix} 1 & x & x^2 & \cdots & x^{d-1} \end{pmatrix} \\ \begin{pmatrix} 0 & 0 & 0 & \cdots & -\frac{f_0}{f_d} \\ 1 & 0 & 0 & \cdots & -\frac{f_1}{f_d} \\ 0 & 1 & 0 & \cdots & -\frac{f_2}{f_d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & -\frac{f_{d-1}}{f_d} \end{pmatrix}\end{array}$$

well known as the Frobenius companion matrix associated with f.

Proposition 1 *Let $f(x) \in \mathbb{C}[x]$ be polynomial of degree d with d distinct roots $\mathcal{Z}(f) = \{z_1,\ldots,z_d\}$, then the eigenvalues of* $\mathrm{Frob}(f)$ *are the roots of $f(x)$, i.e.* $\mathrm{Spec}(\mathrm{Frob}(f)) = \{z_1,\ldots,z_d\}$.

Then, to compute the roots of $f(x)$ one can compute the eigenvalues of its Frobenius companion matrix. This is the object of the method proposed (reintroduced) by Edelman and Murakami [9] and revisited by Fortune [10] and many others trying to use the displacement structure of the companion matrix. In fact, often, the author realized that the monomial basis of the quotient algebra is not the most suitable one and proposed to express the matrix of the same linear application but in other basis. In the case of the Chebyshev basis this algorithm was already known by Barnett [2] and Cardinal later [6].

In the next section, we will also take advantage of the structure of the quotient algebra to design an algorithm for gcd computation mainly using linear algebra (eigenvalues are used in theory and never computed).

3.2 Structure of quotient and gcd

Let $f(x)$ and $g(x) \in \mathbb{K}[x]$ such that they are both monic. As above, we denote $\mathbb{A} = \mathbb{K}[x]/(f)$ and $d = \deg(f)$. We denote $\{\zeta_1, \dots, \zeta_d\}$ the set of roots of $f(x)$ and we assume that $f(x)$ is squarefree, i.e. $\zeta_i \neq \zeta_j$ if $i \neq j$. We define $\mathcal{M}_g : \left\{ \begin{array}{l} \mathbb{A} \longrightarrow \mathbb{A} \\ h \longmapsto \pi(gh) \end{array} \right.$ where $\pi(p) \in \mathbb{A}$ denote the remainder of $p(x) \in \mathbb{K}[x]$ divided by $f(x)$. We denote M_g the matrix of $\mathcal{M}_g$ in the monomial basis $1, x, \dots, x^{d-1}$ of $\mathbb{A}$ but other bases can be used. A matrix representing the map $\mathcal{M}_g$ is called an extended companion matrix.

Proposition 2 *The eigenvalues of* $\mathcal{M}_g$ *are* $\{g(\zeta_1), \dots, g(\zeta_d)\}$.

Proof. It is a direct corollary of the proposition 1 since if we write the matrix of this linear map in the Lagrange basis associated with $\{\zeta_1, \dots, \zeta_d\}$, then it is

$$\begin{pmatrix} g(\zeta_1) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & g(\zeta_d) \end{pmatrix}$$

and gives the wanted result. □

Trivially, we have:

Corollary 1 *We have* $\operatorname{corank}(\mathcal{M}_g) = \deg(f) - \operatorname{rank}(\mathcal{M}_g) = \deg(\gcd(f, g))$.

The i-th column of the matrix M_g contains the coefficients of $\pi(x^{i-1} * g(x))$.

Let $p_1, \dots, p_l$ be a basis of $\operatorname{Ker}(M_g)$ and let $P_1(x), \dots, P_l(x)$ be the corresponding polynomials. First we remark that $\operatorname{Ann}_{\mathbb{A}}(g) = \{P(x) \in \mathbb{A} | P(x) * g(x) = 0\}$ is an ideal of $\mathbb{A}$.

Lemma 1 *The ideal* $\operatorname{Ann}_{\mathbb{A}}(g)$ *is a principal ideal.*

Proof. Let us define

$$v(x) = \prod_{\zeta \in \mathcal{Z}(f) \setminus (\mathcal{Z}(f) \cap \mathcal{Z}(g))} (x - \zeta).$$

For all $h \in \operatorname{Ann}_{\mathbb{A}}(g)$ it is clear that $\mathcal{Z}(h) \supset \mathcal{Z}(f) \setminus (\mathcal{Z}(f) \cap \mathcal{Z}(g))$ and then v divides h. Furthermore $v \in \operatorname{Ann}_{\mathbb{A}}(g)$ since in the Lagrange basis

$$v(x) * g(x) = \sum_{i=1}^{d} v(\zeta_i) g(\zeta_i) * L_i(x) = 0.$$

This shows that $\operatorname{Ann}_{\mathbb{A}}(g) = (v)$. □

To compute $v(x)$, we built the matrix with columns formed by $p_1, \dots, p_l$ and we make a triangulation by operating only on the columns. This way we obtain the polynomial of minimal degree as a linear combination of $P_1(x), \dots, P_l(x)$ and it is easily seen that this $v(x)$ up to a multiplicative scalar factor.

Lemma 2 *The first column of a column echelon form of the matrix* K_g *built from a basis of* $\operatorname{Ker}(M_g)$ *is the generator of* $\operatorname{Ann}_{\mathbb{A}}(g)$*, i.e. it is the vector of the coefficients of* $v(x)$ *up to a scalar multiplication.*

Proof. Since the columns of a column echelon form of the matrix K_g are linearly independent, they form a basis of $\operatorname{Ann}_{\mathbb{A}}(g)$ as $\mathbb{K}$-vector space. So $v(x)$ is a linear combination of the polynomials associated with those columns. The polynomial associated with the column echelon form of K_g have all different degree (because it is an echelon form) and so $v(x)$ is a linear combination of those polynomial. Because $v(x)$ as the lowest degree possible, it is a scalar multiple of the polynomial associated with the first column. □

Proposition 3 $\gcd(f(x), g(x)) = \frac{f(x)}{v(x)}$.

Proof. By construction, we have $v(x) * g(x) = 0 \bmod f(x)$ and so $v(x)$ divide $f(x)$. We also have $\gcd(\frac{f(x)}{v(x)}, g(x)) = \gcd(f(x), g(x))$ since the roots of $\frac{f(x)}{v(x)}$ are the root of $f(x)$ where $g(x)$ vanishes. Since $\deg(\frac{f(x)}{v(x)}) = \deg(\gcd(f(x), g(x))$ we have the wanted result. □

In all this section, we did not care if the polynomials are known in monomial or Chebyshev basis for instance. In fact, in order to have an algebraic algorithm, we only need to be able to perform Euclidean division and this is always the case if the polynomial basis is graduated (as for monomial, Chebyshev, most of the orthogonal bases).

4. BEZOUTIAN AND BARNETT'S FORMULA

A classical matricial formulation of resultant is given by the Bézout matrix. In this part, we recall the construction of the Bézout matrix and a special factorization of the multiplication matrix expressed in the monomial basis. This factorization is called Barnett's formula (see [2]). The Barnett's formula allows to build the classical extended companion matrix without using Euclidean division and only stable numerical computations. Furthermore, this factorization reveals that the extended companion matrix has a special rank structure and we will use this fact later to design a fast algorithm to compute AGCD.

Definition 1 *Let* f *and* $g \in \mathbb{C}[x]$ *of degree* m *and* n *respectively (with* $n \geqslant m$*), we denote* $\Theta_{f,g}(x, y) = \frac{f(x)g(y) - f(y)g(x)}{x - y} = \sum_{i,j} \theta_{i,j} x^i y^j = \sum_{j=0}^{m-1} \kappa_{f,g,j}(x) y^j$. *The Bézout matrix associated with* f *and* g *is* $B_{f,g} = \left(\ \theta_{i,j} \ \right)_{i,j \in \{0, \dots, m-1\}}$.

Remark that since $\Theta_{f,g}(x, y) = \Theta_{f,g}(y, x)$ the matrix $B_{f,g}$ is symmetric. The polynomials $\kappa_{f,g,j}(x)$ are univariate polynomials of degree at most $m - 1$. One particular case of interest is when $f = 1$. In this case the Bézout matrix has a Hankel structure, i.e. $\theta_{i,j} = \theta_{i-1,j+1}$. In this case we denote

$H_{g,i}(x) = \kappa_{1,g,i}(x)$ for $i \in \{0, \dots, m-1\}$ which are called the Horner polynomials.

Proposition 4 *Let $i \in \{0, \dots, m-1\}$, the polynomial $H_{g,i}(x) = c_{1,m-i} + \dots + c_{1,m}x^i$ has degree i and since they have different degree, they form a basis of $\mathbb{C}[x]/(g)$. Furthermore, $\Theta_{1,g}(x, y) = \sum_{i=0}^{m-1} H_{g,m-i}(x)y^i$.*

Corollary 2 *The matrix $B_{1,g}$ is the basis conversion from the Horner basis $H_0, \dots, M_{m-1}$ to the monomial basis $1, x, \dots, x^{m-1}$ of $\mathbb{C}[x]/(g)$.*

This leads us to the following theorem, known as Barnett's formula (see [2]):

Theorem 1 *Let M_g be the multiplication matrix associated with g in $\mathbb{C}[x]/(f)$ in the monomial basis, we have:*

$$M_g = B_{f,g}B_{1,f}^{-1}.$$

Proof. We have $\Theta_{f,g}(x, y) = f(x)\frac{g(y)-g(x)}{x-y} + g(x)\frac{f(x)-f(y)}{x-y}$ and so $g(x)\frac{f(x)-f(y)}{x-y} \equiv \Theta_{f,g}(x, y)$ in $\mathbb{C}[x, y]/(f(x))$. So, for each $i \in \{0, \dots, \deg(f)-1\}$, we have $\Theta_{f,g,i}(x) \equiv g(x)\Theta_{1,f,i}(x)$. This last equality means that $B_{f,g}$ is the matrix of the multiplication by $g(x)$ in $\mathbb{C}[x]/(f)$. The result follows directly from this fact. $\square$

The Barnett's formula reveals the rank structure of the multiplication matrix. Furthermore, this formula is already known if we choose Chebyshev basis instead of monomial basis to express the polynomials and the matrices have exactly the same nature.

5. STRUCTURED MATRICES AND ASYMPTOTICALLY FAST ALGORITHMS

In this section, we briefly recall some basics on displacement structured matrices and related algorithms. We are especially interested in two forms of displacement structure: Toeplitz-like and Cauchy-like structure. See [15] for a more detailed treatment of this topic.

5.1 Displacement structure

Given an integer n and a complex number ϑ with $|\vartheta| = 1$, define the circulant matrix

$$Z_n^\vartheta = \begin{pmatrix} 0 & & & & \vartheta \\ 1 & 0 & & & \\ & 1 & \ddots & & \\ & & \ddots & \ddots & \\ & & & 1 & 0 \end{pmatrix} \in \mathbb{C}^{n\times n}.$$

Next, define the *Toeplitz-like displacement operator* as the linear operator

$$\nabla_T : \mathbb{C}^{m\times n} \longrightarrow \mathbb{C}^{m\times n}$$

$$\nabla_T(A) = Z_m^1 A - AZ_n^\vartheta.$$

A matrix $A \in \mathbb{C}^{m\times n}$ is said to be *Toeplitz-like* if $\nabla_T(A)$ is a small rank matrix (where "small" means small with respect to the matrix size). The number $\alpha = \operatorname{rank}(\nabla(A))$ is called the *displacement rank* of A. If A is Toeplitz-like, then there exist (non-unique) *displacement generators* $G \in \mathbb{C}^{m\times\alpha}$ and $H \in \mathbb{C}^{\alpha\times n}$ such that

$$\nabla_T(A) = GH.$$

Other formally different (but essentially equivalent) definitions of the Toeplitz-like displacement operator are found in the literature; as a consequence, the exact displacement rank of a given matrix may vary slightly depending on the definition that has been chosen.

Toeplitz-like structure is a generalization of Toeplitz structure: indeed, it is easy to see that Toeplitz matrices are also Toeplitz-like. Moreover, note that displacement structure is preserved under inversion: the inverse of a Toeplitz-like matrix is again Toeplitz-like, with the same displacement rank. Sums and products of Toeplitz-like matrices are also Toeplitz-like, even though the displacement rank may increase.

More examples of Toeplitz-like matrices include:

- Toeplitz-block matrices, such as the Sylvester matrix associated with two given polynomials;
- Bézout matrices (which can be defined via sums of products of Sylvester matrices, see e.g. [2]),
- the multiplication matrix M_f (which is a product of Bézoutians).

In particular, M_f has Toeplitz-like displacement rank equal to 2, regardless of its size, with respect to the displacement operator defined above.

A similar definition holds for *Cauchy-like* structure; here the relevant displacement operator is

$$\nabla_C : \mathbb{C}^{m\times n} \longrightarrow \mathbb{C}^{m\times n}$$
$$\nabla_C(A) = D_1A - AD_2,$$

where D_1 and D_2 are diagonal matrices of appropriate size with disjoint spectra. A matrix A is said to be Cauchy-like if $\nabla_C(A)$ has small rank. One can then define notions of Cauchy-like displacement generators and Cauchy-like displacement rank.

5.2 Fast solution of displacement structured linear systems

Gaussian elimination with partial pivoting (GEPP) is a well-known and reliable algorithm that computes the solution of a linear system. Its arithmetic complexity for an $n \times n$ matrix is asymptotically $\mathcal{O}(n^3)$. But if the system matrix exhibits displacement structure, it is possible to apply a variant of GEPP with complexity $\mathcal{O}(n^2)$. The main idea consists in operating on displacement generators rather than on the whole matrix; see [11] for a detailed description of the algorithm (which will be denoted as GKO in the following).

Strictly speaking, the GKO algorithm performs GEPP (or, equivalently, computes the PLU factorization) for Cauchy-like matrices. However, several authors have pointed out (see [11], [13], [20]) that Toeplitz-like matrices can be stably and cheaply transformed into Cauchy-like matrices; the same is true for displacement generators.

Consider, for instance, the case of square matrices, with $\vartheta = 1$. Recall that the Fourier matrix F of size $n \times n$, which defines the discrete Fourier transform, is such that $F_{jk} = \frac{1}{\sqrt{n}}e^{2\pi i(j-1)(k-1)}$. We have the following result (taken from [11]):

Proposition 5 *Let A be an $n \times n$ Toeplitz-like matrix with generators G and H. Denote by D_0 the matrix $\mathrm{diag}(1, e^{\pi i/n}, \ldots, e^{(n-1)\pi i/n})$ and let F be the Fourier matrix of size $n \times n$. Then the matrix $FAD_0^{-1}F^H$ is Cauchy-like, of the same displacement rank as A, with respect to the displacement operator defined by $D_1 = \mathrm{diag}(1, e^{2\pi i/n}, \ldots, e^{2\pi i(n-1)/n})$ and $D_2 = \mathrm{diag}(e^{\pi i/n}, e^{3\pi i/n}, \ldots, e^{(2n-1)\pi i/n})$. Its Cauchy-like generators can be computed as $\hat{G} = FG$ and $\hat{H}^H = FD_0H^H$.*

Here M^H denotes the transpose conjugate of a given matrix M.

Generalization to the case of $m \times n$ rectangular matrices is possible. In this case, the parameter ϑ should be chosen so that the spectra of D_1 and D_2 are well separated (see [1] and [5]).

We also point out that the GKO algorithm can be adapted to pivoting techniques other than partial pivoting ([12], [21]). This is especially useful in case of instability due to internal growth of generator entries. A Matlab implementation of the GKO algorithm that takes into account several pivoting strategies is found in the package DRSolve described in [1].

In our implementation, we use the pivoting strategy proposed in [12]. At each step, the displacement generators G and B are redefined, so that the columns of the new first generator $\tilde{G}$ are orthogonal. This result can be achieved by computing the QR factorization $G = QR$, where Q has size $n \times 2$, and defining new generators $\tilde{G} = Q$ and $\tilde{H} = RH$. This ensures good conditioning of $\tilde{G}$. One then performs pivoted Gaussian elimination on the matrix column corresponding to the column of $\tilde{H}$ with maximum 2-norm. Such a technique involves both row and column permutations; it is not equivalent to complete pivoting, which would be too expensive, but numerical results show that it generally allows a good choice of pivots and reduces the growth of generator entries. Error analysis for this pivoting strategy shows that the backward error essentially depends on the Cauchy-like displacement operator and on the magnitude of the computed upper triangular factor (but not on the magnitude of the generators).

6. A STRUCTURED APPROACH TO AGCD COMPUTATION

We propose here an algorithm that exploits the algebraic and displacement structure of the multiplication matrix to compute the AGCD of two given polynomials with real coefficients (as defined in Section 1).

6.1 Rank estimation

It has been pointed out in Section 3 that the rank deficiency of the multiplication matrix equals the gcd degree. In an approximate setting, a relevant notion is the *approximate rank* (or ε-rank): recall that a matrix Mhas ε-rank k if there exists a matrix $\tilde{M}$ of exact rank k such that $\left\|\tilde{M} - M\right\|_2 \leq \varepsilon$. The AGCD degree can then be estimated using the approximate rank of the multiplication matrix. Observe that the technique of computing the AGCD degree via approximate rank of a resultant (e.g., Sylvester) matrix is often exploited in the literature: see for instance [7], [8], [16], [5].

Here we use the structured pivoted LU decomposition to estimate the approximate rank of the multiplication matrix. Recall that M_g has a Toeplitz-like structure with displacement rank 2; it can then be transformed into a Cauchy-like matrix $\hat{M}_g$ as described in Section 5.2. Fast pivoted Gauss elimination yields a factorization $\hat{M}_g = P_1LUP_2$, where L is a square, nonsingular, lower triangular matrix with diagonal entries equal to 1, U is upper triangular and $P_{1,2}$ are permutation matrices. Inspection of the diagonal entries (or of the row norms) of U allows to estimate the approximate rank of $\hat{M}_g$ and, therefore, of M_g.

Developing a reliable and numerically robust factorization-based strategy to compute the AGCD degree is not an easy task. As far as the LU factorization is concerned, we mention that Gauss elimination with complete pivoting is rank-revealing in the exact case, but in principle it might not detect near rank-deficiency. Since we cannot expect the pivoting strategy we use to perform better than complete pivoting, we conclude that our strategy cannot ensure an *a priori* certification of approximate rank, nor of AGCD degree. There is, however, a relationship between GKO pivots and distance from a rank-deficient matrix. Indeed at any GKO step we have

$$|a| \leq \left\|\tilde{U}\right\|_2 \leq k^{3/2}\rho\,|a|\,,$$

where $\widetilde{U} \in \mathbb{C}^{k\times k}$ is the trailing matrix block that has not yet been factorized, a is the current pivot and ρ is a positive constant depending on the Cauchy-like displacement operator. This bound can be derived directly using the results presented in [12]; see also [5].

We propose in the future to give a more detailed theoretical and numerical analysis of the effectiveness of our degree estimation strategy; the proposed numerical experiments are a first step in this direction.

6.2 Minimization of a quadratic functional

Let us suppose that:

- the polynomial $f(x) = \sum_{j=0}^{n} f_j x^j$ is exactly known,
- the polynomial $g(x) = \sum_{j=0}^{m} g_j x^j$ is approximately known and may be perturbed, so that we consider its coefficients as variables,
- the AGCD degree is known.

Then we can reformulate the problem of AGCD computation as the minimization of a quadratic functional. Indeed, recall that the cofactor $v(x)$ with respect to $f(x)$ is defined by the "shortest" vector (i.e., the vector with the maximum number of trailing zeros) that belongs to the null space of M_g. We assume $v(x)$ to be monic; we denote its degree as k and we have

$$M_g v = M_g \cdot \begin{pmatrix} v_0 \\ \vdots \\ v_{k-1} \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 0 \end{pmatrix}.$$

Also observe that the entries of M_g are linear functions of the coefficients of $g(x)$. Then the equation $M_g v = 0$ can be rewritten as $\mathcal{F}(g,v)$=0, where the functional $\mathcal{F}$ is defined as

$$\mathcal{F}:\mathbb{C}^{m+1}\times\mathbb{C}^k\longrightarrow\mathbb{R}_+$$
$$\mathcal{F}(g,v)=\|M_g v\|_2^2.$$

For a preliminary study of the problem, we have chosen to solve the equation $\mathcal{F}(g,v)$=0 by means of Newton's method, applied so as to exploit structure. Denote by $z=[g_0,\dots,g_m,v_0,\dots,v_{k-1}]^T$ the vector of unknowns; then each Newton step has the form

$$z^{(j+1)}=z^{(j)}-J(g^{(j)},v^{(j)})^{\dagger}M_{g^{(j)}}v^{(j)}.$$

In particular, notice that the Jacobian matrix associated with $\mathcal{F}$ is an $n\times(m+k+1)$ Toeplitz-like matrix of displacement rank 3. This property allows to compute a solution of the linear system $J(g^{(j)},v^{(j)})y=M_{g^{(j)}}v^{(j)}$ in a fast way; therefore, the arithmetic complexity of each iteration is quadratic w.r.t. the degree of the input polynomials.

We propose in the future to take into consideration other optimization methods in the quasi-Newton family, such as BFGS.

6.3 Computation of displacement generators

In order to perform fast factorization of the multiplication matrix M_g, we need to compute Toeplitz-like displacement generators. It turns out that the range of $\nabla(M_g)$ is spanned by the first and last column of the displaced matrix, and the columns of indices from 2 to $n-1$ are multiples of the first one. Therefore, it suffices to compute a few rows and columns of M_g in order to obtain displacement generators. This can be done in a fast and stable way by using Barnett's formula. If we denote e_j the j-th vector of the canonical basis of $\mathbb{C}^n$, then the computation of the j-th column of $M_g$can be seen as

$M_g(:,j)=B(f,g)\cdot\left(B(1,f)^{-1}e_j\right)$,

that is, it consists in solving a triangular Hankel linear system and computing a matrix-vector product. For row computation, recall that the Bezoutian is a symmetric matrix; we have analogously:

$M_g(j,:)=e_j^T\cdot B(f,g)\cdot B(1,f)^{-1}=\left(B(1,f)^{-1}B(f,g)e_j\right)^T$,

so that the computation of a row of M_g amounts to performing a matrix-vector product and solving a Hankel triangular system.

A similar approach holds for computation of displacement generators of the Jacobian matrix $J(g,v)$ associated with the functional $\mathcal{F}(g,v)$.

6.4 Description of the algorithm

Input: coefficients of polynomials $f(x)$ and $g(x)$.
Output: a perturbed polynomial $\tilde{g}(x)$ such that f and $\tilde{g}$ have a nontrivial common factor.

1. Estimate the approximate rank k of M_g by computing a fast pivoted LU decomposition of the associated Cauchy-like matrix.

2. Again by using fast LU, compute a vector $v=[v_0,v_1,\dots,v_{k-1},1,0,\dots,0]^T$ in the approximate null space of M_g.

3. Apply structured Newton with initial guess (g,v) and compute polynomials $\tilde{g}$ and $\tilde{v}$ such that f and $\tilde{g}$ have a common factor of degree $\deg f-k$ and $\tilde{v}$ is the monic cofactor for f.

The algebraic complexity of step 1 is $\mathcal{O}(\deg(f)^2)$ and this is also the complexity of step 2 and of each iteration of step 3. This asymptotic bound seems to be realistic even if the degrees used for our experimentations are not large enough to truly confirm it in practice. The number of Newton iterations needed, in our tests, does not seem to grow with the degree as far as our implementation allows us to go. So, in our tests, the complexity of the rank computation dominates the arithmetical cost of the algorithm.

6.5 Numerical experiments and computational issues

We have written a preliminary implementation of the proposed method in Matlab (available at the URL `http://www.unilim.fr/pages_perso/paola.boito/MMgcd.m`).

The results of a few numerical experiments are shown below. The polynomials f and g are monic and have random coefficients uniformly distributed over $[-1,1]$. They have an exact GCD of prescribed degree. A perturbation is then added to g. The perturbation vector has random entries uniformly distributed over $[-\eta,\eta]$ and its norm is of the order of magnitude of η. We show:

- the residual $\mathcal{F}(\tilde{g},\tilde{v})$,
- the 2-norm distance between the exact and the computed cofactor v,
- the 2-norm distance between the exact and the computed perturbed polynomial g (which is expected to be roughly of the same order of magnitude as η).

In the following table we have taken η =1e-5.

$n,m,\deg\gcd(f,g)$	$\mathcal{F}(\tilde{g},\tilde{v})$	$\|v-\tilde{v}\|_2$	$\|g-\tilde{g}\|_2$
8, 7, 3	1.02e-15	1.19e-15	1.40e-5
15, 14, 5	1.51e-15	2.26e-15	1.35e-4
22, 22, 7	2.07e-13	1.40e-13	2.20e-4
36, 36, 11	1.19e-12	5.07e-14	0.0012

Here are results for η =1e-8:

$n,m,\deg\gcd(f,g)$	$\mathcal{F}(\tilde{g},\tilde{v})$	$\|v-\tilde{v}\|_2$	$\|g-\tilde{g}\|_2$
8, 7, 3	5.49e-15	1.63e-15	5.85e-8
28, 27, 13	7.90e-14	8.98e-14	6.50e-7
38, 37, 13	4.88e-12	4.26e-12	2.30e-5
58, 57, 23	2.03e-12	4.40e-12	2.54e-4

There are several issues in our approach that deserve further investigations. Let us mention in particular:

- The choice of a threshold (or a more refined technique) for estimating approximate rank.
- Normalization of polynomials: here we mostly work with monic polynomials, but other normalizations may be considered.
- The structured implementation of the optimization step (minimizing $\mathcal{F}(g,v)$). We have used for now a heuristic structured version of the Gauss-Newton algorithm. Observe that each step of classical Gauss-Newton applied to our problem has the form $z^{(j+1)}=z^{(j)}-y^{(j)}$, where $z^{(j)}$ is the vector containing the coefficients of the j-th iterate polynomials $g^{(j)}$ and $v^{(j)}$, and $y^{(j)}$ is the least-norm solution to the underdetermined system $J(g^{(j)},v^{(j)})y^{(j)}=M_{g^{(j)}}v^{(j)}$. Computing this least-norm solution in a structured and fast way is a difficult point that will require more work. Our implementation gives a solution which is not, in general, the

least-norm one, even though it is typically quite close. Further work will also include a study of other possible optimization methods that lend themselves well to a structured approach.

7. CONCLUSIONS

We have proposed and implemented a fast structured matrix-based approach to a variant of the AGCD problem, namely, the problem of computing an approximate greatest common divisor of two univariate polynomials, one of which is known to be exact. To our knowledge, this variant has been so far neglected in the existing literature. It may be also interesting when one polynomial is known with high accuracy and the other is not.

Our approach is based on the structure of the multiplication matrix and on the subsequent reformulation of the problem as the minimization of a suitably defined functional. Our choice of the multiplication matrix M_g over other resultant matrices (e.g., Sylvester, Bézout...) is motivated by

- the smaller size of M_g, with respect e.g. to the Sylvester matrix,
- the strong link between the null space of M_g and the gcd, and in particular the fact that the null space of M_g immediately yields a gcd cofactor,
- the displacement structure of M_g,
- the possibility of computing selected rows and columns of M_g in a stable and cheap way, thanks to Barnett's formula.

This is, however, a preliminary study. Further work will include generalizations of the proposed problem and a more thorough analysis of the optimization part of the algorithm. Furthermore, this approach can be generalized in several interesting ways:

- using better bases than the monomial one,
- it can be extended to some multivariate setting to compute the co-factor of a polynomial g in $\mathbb{C}[x_1, \dots, x_n]/(f_1, \dots, f_n)$ when $f_1, \dots, f_n$ define a complete intersection since Barnett's formula still holds,
- to compute the AGCD of f with $g_1, \dots, g_k$ where f is known with accuracy but $g_1, \dots, g_k$ are inaccurate, one can take g as a linear combination of $g_1, \dots, g_k$ with our method and succeed with a high probability.

8. REFERENCES

[1] A. Aricò, G. Rodriguez, *A fast solver for linear systems with displacement structure*. Numer. Algorithms, 2010. DOI: 10.1007/s11075-010-9421-x.

[2] S. Barnett, *Polynomials and linear control systems*, Monographs and Textbooks in Pure and Applied Mathematics, 77, Marcel Dekker, Inc., New York, 1983. xi+452 pp. ISBN: 0-8247-1898-4.

[3] B. Beckermann, G. Labahn, *When are two numerical polynomials relatively prime?*, Journal of Symbolic Computation, 28 (1998), no. 6, 691–714.

[4] B. Beckermann, G. Labahn, *Fast and numerically stable Euclidean-like algorithm for detecting relatively prime numerical polynomials*, Journal of Symbolic Computation, 28 (1998), no. 6, 691–714.

[5] D. Bini, P. Boito, *A fast algorithm for approximate polynomial gcd based on structured matrix computations*, in Numerical Methods for Structured Matrices and Applications: Georg Heinig memorial volume, Operator Theory: Advances and Applications, vol. 199, Birkhäuser (2010), 155-173.

[6] J.-P. Cardinal, *On two iterative methods for approximating roots of a polynomial*, In J. Renegar, M. Shub and S. Smale editors, Proc. SIAM-AMS Summer Seminar on Math. of Numerical Analysis, Vol. 32 of Lecture Notes in Applied Math., AMS Press (1996), 165-188.

[7] R. Corless, P. Gianni, B. Trager, S. Watt, *The Singular Value Decomposition for Approximate Polynomial Systems*, Proceeedings of ISSAC 1995 (Montreal, Canada), 195-207, ACM Press (1995).

[8] R. Corless, S. Watt, L. Zhi, QR *factoring to compute the GCD of univariate approximate polynomials*, IEEE Trans. Signal Process., 52 (2004), no. 12, 3394–3402.

[9] A. Edelman, H. Murakami, *Polynomial roots from companion matrix eigenvalues*, Math. Comp. 64 (1995), no. 210, 763-776.

[10] S. Fortune, *An iterated eigenvalue algorithm for approximating roots of univariate polynomials*, Journal of Symbolic Computation, 33 (2002), no. 5, 627-646.

[11] I. Gohberg, T. Kailath, and V. Olshevsky, *Fast Gaussian elimination with partial pivoting for matrices with displacement structure,* Math. Comp. 64 (212), 1557-1576 (1995).

[12] M. Gu, *Stable and Efficient Algorithms for Structured Systems of Linear Equations*, SIAM J. Matrix Anal. Appl. 19, 279-306 (1998).

[13] G. Heinig, *Inversion of generalized Cauchy matrices and other classes of structured matrices*, Linear Algebra in Signal Processing, IMA volumes in Mathematics and its Applications 69, 95-114 (1994).

[14] U. Helmke, P. Fuhrmann, *Bezoutians*, Linear Algebra Appl. 124, 1039-1097 (1989).

[15] T. Kailath, A. Sayed, *Displacement Structure: Theory and Applications*, SIAM Review 37(3), 297-386 (1995).

[16] E. Kaltofen, Z. Yang, L. Zhi, *Structured Low Rank Approximation of a Sylvester Matrix*, in Dongming Wang and Lihong Zhi (eds.), Proc. International Workshop on Symbolic-Numeric Computation (2005).

[17] N. Karmarkar, Y. Lakshman, *Approximate polynomial greatest common divisors and nearest singular polynomials*, Proc. Int. Symp. Symbolic Algebraic Comput., Zurich, Switzerland, 1996, pp. 35-42.

[18] N. Karmarkar, Y. Lakshman, *On approximate GCDs of univariate polynomials*, Journal of Symbolic Computation, 26 (1996), no. 6, 653–666.

[19] B. Mourrain, O. Ruatta, *Relations between roots and coefficients, interpolation and application to system solving*, Journal of Symbolic Computation, 33 (2002), no. 5, 679-699.

[20] V. Pan, *On computations with dense structured matrices*, Math. of Comput. 55(191), 179-190 (1990).

[21] M. Stewart, *Stable Pivoting for the Fast Factorization of Cauchy-Like Matrices*, preprint (1997).

[22] Z. Zeng, *The approximate GCD of inexact polynomials Part I: a univariate algorithm*, http://www.neiu.edu/~zzeng/uvgcd.pdf.

Using weighted norms to find nearest polynomials satisfying linear constraints

Nargol Rezvani
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada.
nrezvani@cs.toronto.edu

Robert M. Corless
Department of Applied Mathematics
University of Western Ontario
London, Ontario, Canada.
rcorless@uwo.ca

ABSTRACT

This paper extends earlier results on finding nearest polynomials, expressed in various polynomial bases, satisfying linear constraints. Results are extended to different bases, including Hermite interpolational bases (not to be confused with the Hermite orthogonal polynomials). Results are also extended to the case of weighted norms, which turns out to be slightly nontrivial, and interesting in practice.

Categories and Subject Descriptors

1.4 [**Symbolic Manipulation**]: Applications

General Terms

Algorithms

Keywords

Nearest polynomial; Lagrange basis; Hermite interpolational basis

1. INTRODUCTION

Given a pair of polynomials f and g, the problem of finding the smallest perturbations Δf and Δg such that $f+\Delta f$ and $g+\Delta g$ have a nontrivial GCD has been well-studied since [2]. In particular, the paper [9] provides a complete and efficient *structured total least squares* approach to this problem, when the 2, 1, or ∞-norm is used and the polynomials (which may be multivariate for their algorithm) are expressed in the usual monomial basis.

As in [10], here we look at a related but somewhat simpler problem, which was discussed in [14]. Our approach here differs from other works in that it specifically allows weighted p-norms for all values of p, not just 1, 2, or ∞, at the cost of essentially restricting to the degree-1 GCD case, that is $x-r$ where we also allow $r=\infty$ meaning *degree reduction*, and it allows essentially arbitrary polynomial bases to be used. Earlier papers discussing special cases of our result include [2], [7], and [8].

The use of degree reduction in the Bernstein-Bézier basis in the Computer-Aided Geometric Design literature is widespread: see for example the recent work [16], but the idea is well-understood and in many textbooks. Our approach differs in that the metric we advocate is a metric on the vector of polynomial coefficients, whereas the standard work uses a metric based on an integral function norm. At this juncture it is not clear that our approach offers any real advantage, although we believe that it might.

In [10] the problem was stated in four forms, which we reproduce below. We suppose that we are given $f \in \mathbb{P}$ where

$$\mathbb{P} \cong \mathbb{C}^{n+1}[x] = \operatorname{span}\{\phi_0(x), \phi_1(x), \ldots, \phi_n(x)\}$$

where the $\phi_k(x)$ form a basis for the set of polynomials with complex coefficients of degree less than $n+1$. We also suppose that we are given a metric $d : \mathbb{P}\times\mathbb{P} \longrightarrow \mathbb{R}^+ \cup \{0\}$ measuring the distance between polynomials, and given a complex number r (which, here unlike in [10], may be infinity) that is the desired zero.

- Problem A. Find $\tilde{f} \in \mathbb{P}$ with $\tilde{f}(r) = 0$ and $d(f, \tilde{f})$ minimal.
- Problem B. Find $\tilde{f}$ as above but also insist that $\deg(f) = \deg(\tilde{f})$.
- Problem C. Given a *monic* f (in a basis for which this makes sense), find *monic* $\tilde{f}$ as above.
- Problem D. Given a polynomial with some *intrinsic* (i.e. fixed) coefficients, and some *empiric* (i.e. contaminated by data error) coefficients, find the nearest $\tilde{f}$ with the same intrinsic coefficients and $\tilde{f}(r) = 0$. For example, *sparse* polynomials fit into this category: the intrinsic coefficients are the zero coefficients.

The solution to Problem A was given in a lecture by E. Kaltofen in August 1999, based on work by M. Hitz. It was further studied in [14] in the case $\phi_k(x) = x^k$, the monomial basis, and $d(f_1, f_2) = \|f_1 - f_2\|_q$, the q-norm of the vector of coefficients, $\mathbf{v}$, of the difference between f_1 and f_2:

The formulas of [14] were only implemented in the monomial basis. In [10], following the same approach as [14], we extended the problem to different bases such as Lagrange, Bernstein and Chebyshev. In our 2007 SNC extended abstract [3], we briefly discussed the weighted norm and applied it to this problem. This present paper extends those results to the Hermite interpolational basis.

Table 1: Examples of polynomial bases allowed

Power basis	$\phi_k(x) = x^k$
Chebyshev	$\phi_k(x) = T_k(x)$
Pochhammer	$\phi_k(x) = (x+a)^{\overline{n}}$
Newton	$\phi_k(x) = (x-\tau_1)(x-\tau_2)\cdots(x-\tau_{k-1})$
Bernstein-Bézier	$\phi_k(x) = \binom{n}{k}(x-a)^k(b-x)^{n-k}/(b-a)^n$
Lagrange	$\phi_k(x) = \beta(x)\gamma_k/(x-\tau_k)$
Hermite	$\phi_{i,j}(x) = \beta(x)\sum_{j=k}^{s_i-1}\gamma_{i,j}(x-\tau_i)^{k-j-1}$

As in that previous paper, we here define a weighted p-norm with the weight vector w, that is continuous in p. We make use of its weighted dual norm, the q-norm with the dual weight vector w^*, where $1/p + 1/q = 1$. There is a uniform relation between the entries of w and w^* for different values of p. By using the weighted version, we can insist on sparsity or on monicity. We may also smooth the polynomial by lowering the degree of it iteratively. By exploiting the differentiation matrix (see e.g. [15, Ch. 6]), we may also find the nearest polynomial with a given *derivative* zero.

2. PROBLEM SET-UP

We suppose that we are given $f \in \mathbb{P}$ where

$$\mathbb{P} \cong \mathbb{C}^{n+1}[x] = \operatorname{span}\{\phi_0(x), \phi_1(x), \ldots, \phi_n(x)\}$$

where the $\phi_k(x)$ form a basis for the set of polynomials of degree less than $n+1$ with complex coefficients. In this paper, we allow the bases listed in table 1. We also suppose that we are given a metric $d : \mathbb{P} \times \mathbb{P} \longrightarrow \mathbb{R}^+ \cup \{0\}$ measuring the distance between polynomials, and given a complex number r that is the desired zero (it could be ∞). So the our goal is to find $\tilde{f} \in \mathbb{P}$ with $\tilde{f}(r) = 0$ and $d(f, \tilde{f})$ minimal. This problem can be stated as an unconstrained problem. We can also insist on the monicity (in bases where this makes sense) or sparsity of $\tilde{f}$.

3. WEIGHTED DUAL NORM

The dual norm is a consequences of Hölder's inequality. It can be proved that a weighted norm satisfies a form of the inequality. Therefore, we may define the dual of a weighted norm and all the previous work goes through mutatis mutandis.

Consider the weighted p-norm with the following definition.

Definition 1. Let $\mathbf{w} = [w_0, w_1, \cdots, w_n]$ be the weight vector. Then,

$$\|\mathbf{u}\|_{p,w} = \begin{cases} (\sum_{i=0}^{n} |w_i u_i|^p)^{1/p} & \text{if } p \neq \infty \\ \max_{0\le i\le n} w_i|u_i| & \text{if } p = \infty \end{cases} \tag{1}$$

The weight vector w has for the moment strictly positive components: $w_i > 0$. Considering the above definition, we can extend our algorithm for finding the nearest polynomial with a give zero (NPGZ). An easy way to do this is as follows.

Step 1. Change the weighted problem to an unweighted problem by scaling.

Step 2. Solve the problem by the un-weighted theorem.

Step 3. Convert the achieved results back to the weighted version.

Specifically, given a vector $\mathbf{u}$ with $\|\mathbf{u}\|_{p,w} = 1$, define

$$\hat{\mathbf{u}} = (w_0u_0, \cdots, w_nu_n).$$

So for $1 \le p < \infty$ we will have,

$$\|\hat{\mathbf{u}}\|_p = \left(\sum_{i=0}^{n}(|w_iu_i|)^p\right)^{1/p} = \|\mathbf{u}\|_{p,w} = 1\,,$$

and for $p = \infty$

$$\|\hat{\mathbf{u}}\|_p = \|\hat{\mathbf{u}}\|_\infty \;=\; \max_{0\le i\le n}|w_iu_i| = \|\mathbf{u}\|_{p,w} = 1.$$

Define $\hat{\mathbf{v}}$ as the witness vector with $\|\hat{\mathbf{v}}\|_p^* = 1$ and $\hat{\mathbf{v}} \cdot \hat{\mathbf{u}} = 1$. Then if we define $\mathbf{v}$ by

$$\mathbf{v} = (w_0\hat{v}_0, \cdots, w_n\hat{v}_n)$$

we will have

$$\begin{aligned} \hat{\mathbf{u}} \cdot \hat{\mathbf{v}} &= \sum_{i=0}^{n}\hat{u}_i\hat{v}_i = \sum_{i=0}^{n}(w_iu_i)(w_i^{-1}v_i)) \\ &= \sum_{i=0}^{n}u_iv_i = \mathbf{u}\cdot\mathbf{v}\,, \end{aligned}$$

Note $w_i > 0$ so these all make sense. By the definition of the dual norm we have

$$\|\hat{\mathbf{v}}\|_p^* \ge |\hat{\mathbf{u}} \cdot \hat{\mathbf{v}}| = |\mathbf{u}\cdot\mathbf{v}|$$

for $1 \le p < \infty$, with equality if and only if

$$\hat{v}_k = \begin{cases} \gamma|\hat{u}_k|^{p-2}\bar{\hat{u}}_k & \text{if } \hat{u}_k \neq 0 \\ 0 & \text{if } \hat{u}_k = 0 \end{cases}\,, \tag{2}$$

which is equivalent to

$$v_k = \begin{cases} \gamma w_k^p|u_k|^{p-2}\bar{u}_k & \text{if } u_k \neq 0 \\ 0 & \text{if } u_k = 0 \end{cases}\,. \tag{3}$$

For $p = \infty$

$$\hat{v}_k = \begin{cases} 0 & \text{if } k \neq k_0 \\ \gamma\bar{\hat{u}}_{k_0} & \text{if } k = k_0 \end{cases} \tag{4}$$

which is equivalent to

$$v_k = \begin{cases} 0 & \text{if } k \neq k_0 \\ \gamma w_{k_0}^2\bar{u}_{k_0} & \text{if } k = k_0 \end{cases} \tag{5}$$

So assuming $w_i > 0$, define the dual weighted norm as below.

$$\|\mathbf{v}\|_{p,w}^* = \|\mathbf{v}\|_{q,w^*}$$

where $w^* = (w_0^{-1}, \cdots, w_n^{-1})$ is the weight vector in the dual norm.

So for the case $1 < p < \infty$ we will have

$$\begin{aligned}
\|\mathbf{v}\|^*_{p,w} &= \|\mathbf{v}\|_{q,w^*} = (\sum_{i=0}^{n}(|w_i^{-1}v_k|)^q)^{1/q} \\
&= (\sum_{i=0}^{n}(|w_i^{-1}\gamma w_i^p|u_i|^{p-2}\bar{u}_i|)^q)^{1/q} \\
&= (\sum_{i=0}^{n}(w_i^{p-1}|u_i|^{p-1})^q)^{1/q} \\
&= (\sum_{i=0}^{n}(w_i^{p-1}|u_i|^{p-1})^{p/(p-1)})^{1/q} \\
&= (\sum_{i=0}^{n}|w_iu_i|^p)^{1/q} = (\|\mathbf{u}\|^p_{p,w})^{1/q} = 1.
\end{aligned}$$

for the case $p = \infty$ $(q = 1)$ we will have

$$\begin{aligned}
\|\mathbf{v}\|^*_{\infty,w} &= \|\mathbf{v}\|_{1,w^*} = \sum_{k=0}^{n}|w_k^{-1}v_k| \\
&= w_{k_0}^{-1}|v_{k_0}| = |w_{k_0}^{-1}\gamma w_{k_0}^2\bar{u}_{k_0}| \\
&= |w_{k_0}\bar{u}_{k_0}| = \max_{0\le i\le n}|w_iu_i| \\
&= \|\mathbf{u}\|_{p,w} = 1.
\end{aligned}$$

and finally for the case $p = 1$ $(q = \infty)$ we get

$$\begin{aligned}
\|\mathbf{v}\|^*_{1,w} &= \|\mathbf{v}\|_{\infty,w^*} \\
&= \max_{0\le k\le n} w_k^{-1}|v_k| \\
&= \max_{0\le k\le n} |w_k^{-1}w_k|u_k|^{-1}\bar{u}_k| \\
&= \max_{0\le k\le n} 1 \\
&= 1.
\end{aligned}$$

Now consider the following limits that show what happens when the weight vector has components with a value of zero. Three different cases are considered.

- $(1 < p < \infty) \longleftrightarrow (1 < q < \infty)$:

$$\begin{aligned}
\lim_{w_j\longrightarrow 0}\|\mathbf{v}\|_{q,w^*} &= \lim_{w_j\longrightarrow 0}(\sum_{i=0}^{n}|w_i^{-1}v_i|^q)^{1/q} \\
&= \lim_{w_j\longrightarrow 0}(\sum_{i=0}^{n}w_i^{-q}|v_i|^q)^{1/q} \\
&= \lim_{w_j\longrightarrow 0}(\sum_{i=0,i\neq j}^{n}w_i^{-q}|v_i|^q)^{1/q} + (w_j^{-q}|v_j|^q)^{1/q}
\end{aligned}$$

The second part dominates the limit and therefore the above limit will be

$$\lim_{w_j\longrightarrow 0}(w_j^{-q}|v_j|^q)^{1/q} = \lim_{w_j\longrightarrow 0} w_j^{-1}|v_j|$$

- $p = 1 \longleftrightarrow q = \infty$:

$$\begin{aligned}
\lim_{w_j\longrightarrow 0}\|\mathbf{v}\|_{\infty,w^*} &= \lim_{w_j\longrightarrow 0}\max_{0\le i\le n} w_i^{-1}|v_i| \\
&= \lim_{w_j\longrightarrow 0} w_j^{-1}|v_j|
\end{aligned}$$

- $p = \infty \longleftrightarrow q = 1$:

$$\begin{aligned}
\lim_{w_j\longrightarrow 0}\|\mathbf{v}\|_{1,w^*} &= \lim_{w_j\longrightarrow 0}(\sum_{i=0}^{n}w_i^{-1}|v_i|) \\
&= \lim_{w_j\longrightarrow 0}\sum_{i=0,i\neq j}^{n}w_i^{-1}|v_i| + w_j^{-1}|v_j| \\
&= \lim_{w_j\longrightarrow 0} w_j^{-1}|v_j|
\end{aligned}$$

Since $\|\mathbf{v}\|^*_{p,w^*}$ is bounded, in all three cases we must require $v_j = 0$ and therefore the definition of the weighted dual norm is now complete.

Definition 2.

$$\|\mathbf{v}\|^*_{p,w} = \|\mathbf{v}\|_{q,w^*}$$

where $w^* = (w_0^{-1}, \cdots, w_n^{-1})$ and if any $w_i = 0$, then unless $v_i = 0$ as well, $\|\mathbf{v}\|^*_{p,w} = \infty$.

The NPGZ algorithm finds the nearest polynomial to a given zero in q-norm, while it minimizes the perturbations which are built by the unit vector $\mathbf{u}$ in p-norm, where $1/p + 1/q = 1$. Therefore, the weight vector used in the routine is w, the weight vector in p-norm. When $w_i = 0$ we have $w_i^* = \infty$. It is easier for the user to specify a zero than infinity. Therefore we ask that the user specifies the dual weights. In order to add the weight vectors to the algorithm of NPGZ, we should therefore modify equations (21) and (22). Here is the modified algorithm.

4. THE WEIGHTED VERSION OF NPGZ

Input: Polynomial $f = \mathbf{a}\cdot\mathbf{\Phi}(x) \in \mathbb{P}^n$, spanned by a basis Φ, where $\mathbf{a}$ is the vector of coefficients, a given zero r (the case where $r = \infty$ is handled in the next section) and the q-norm in which we wish to minimize the perturbations. The vector w is the dual weight vector (in p-norm).
Output: The perturbed polynomial $\tilde{f}$ such that $\|f-\tilde{f}\|_{q,w^*}$ is minimal.

1. Compute $p = q/(q-1)$.
2. Let $\mathbf{u} = \mu\tilde{\mathbf{\Phi}}(r)$ where $\mu = \frac{1}{\|\mathbf{\Phi}\|_{p,w}}$ and $\tilde{\Phi}_i = w_i\Phi_i$ (Φ is the basis in which f is defined).
3. Compute the minimal perturbations using equations (3) and (5) and definition (2).
   For $p \neq \infty$
   $$\Delta a_k = \begin{cases} \mu|f(r)|\gamma w_k^p|u_k|^{p-2}\bar{u}_k & \text{if } w_ku_k \neq 0 \\ 0 & \text{if } w_ku_k = 0 \end{cases}, \quad (6)$$
   and for $p = \infty$
   $$\Delta a_k = \begin{cases} 0 & \text{if } k \neq k_0 \\ \mu|f(r)|\gamma w_{k0}^2\bar{u}_{k_0} & \text{if } k = k_0 \end{cases}, \quad (7)$$
   where $\gamma = -\text{signum}(f(r)) = -\frac{f(r)}{|f(r)|}$ and k_0 is any, say the least, index with $|u_{k_0}| = 1$; there must be at least one such, because $\|\mathbf{u}\|_\infty = 1$. Then
   $$\|\mathbf{v}\|^* = \|\mathbf{v}\|_q = 1$$
   and $\mathbf{v}\cdot\mathbf{u} = \gamma$ so $|\mathbf{v}\cdot\mathbf{u}| = 1$.
4. For symbolic x, return $\tilde{f} = (\mathbf{a}+\Delta\mathbf{a})\cdot\mathbf{\Phi}(x)$.

5. LOWER DEGREE

In some cases we are interested in finding the nearest polynomial with a lower degree, i.e. with a root at ∞. However, we need to discuss this issue in the separate cases when the polynomial basis is degree-graded, e.g. the monomial basis, and the polynomial basis is not degree-graded, e.g. Bernstein-Bézier, Lagrange, or Hermite interpolational bases.

5.1 Degree-graded Bases

In this simple case, all we need to do is force the leading coefficient to be zero. This approach is used without comment in many places (for example, in Chebyshev economization [12]).

LEMMA 1. *If $f(x) = \sum_{k=0}^{n} c_k\phi_k(x)$, $\deg \phi_k = k$, then the nearest polynomial of lower degree to f in any p-norm is $\tilde{f}(x) = \sum_{k=0}^{n-1} c_k\phi_k(x)$.*

Remark- In this Lemma, we want to force the leading coefficient to be zero. In fact we are using $p = \infty$, which is the second case of step 3 in NPGZ algorithm.

5.2 Bernstein-Bézier Basis

As is well-known given

$$p(x) = \sum_{k=0}^{n} C_k^{(n)} B_k^{(n)}(x)\ ,$$

if $p(x)$ truly is of lower degree, one can also express $p(x)$ in terms of Bernstein polynomials of lower degree,

$$p(x) = \sum_{k=0}^{n-1} C_k^{(n-1)} B_k^{(n-1)}(x)\ .$$

This is called *degree reduction* [5]. The process uses the recurrence in Lemma 2 below. This only works if $p(x)$ really has degree $n-1$ or less.

Given a polynomial $p(x)$ that has been computed by some process involving *errors*, $p(x)$ may not be of degree $n-1$, but only close to such a polynomial. The purpose of our paper is to introduce a procedure that can be used to guarantee lower degree, before the formal process of degree reduction is carried out.

The next lemma slightly generalizes those in section 3.2 of [5].

LEMMA 2.

$$C_k^{n+1} = \begin{cases} C_0^n & \text{if } k=0 \\ \frac{1}{b-a}\left[\frac{k}{n+1}C_{k-1}^n + (1-\frac{k}{n+1})C_k^n\right] & \text{if } 1 \le k \le n \\ C_n^n & \text{if } k = n+1 \end{cases}\ .$$

Proof: exercise. **Remark.** This recurrence relation can be solved for the vector C_k^n given the vector C_k^{n+1}. This is what is meant by 'degree reduction'.

In Bernstein basis, the vector of leading coefficients will be

$$[(-1)^0\binom{n}{0}, \cdots, (-1)^k\binom{n}{k}, \cdots, (-1)^n\binom{n}{n}]\ .$$

After normalization, this vector is the limit of $\mathbf{u}$ from equation (8) as $r \to \infty$. As before, we enforce lower degree by the linear constraint

$$\sum_{k=0}^{n} (-1)^k \binom{n}{k} C_k = 0$$

.

5.3 Lagrange Basis

We define the unit vector u to be

$$\mathbf{u} = \frac{\mathbf{\Phi}(r)}{\|\mathbf{\Phi}(r)\|_p}\ , \tag{8}$$

where $p = q/(q-1)$ and Φ is the polynomial basis. The vector of leading coefficients of the Lagrange polynomials is proportional to the vector of the barycentric weights, $[\beta_0, \beta_1, \ldots, \beta_n]^T$, the limit of $\mathbf{u}$ from equation (8) as $r \to \infty$. We thus enforce lower degree by the linear constraint $\sum_{j=0}^{n} \beta_j f_j = 0$. This is mathematically equivalent to setting the leading coefficient of f in the monomial basis

$$\sum_{j=0}^{n} \beta_j f_j \ = \ \frac{f^{(n)}(0)}{n!}$$

to zero.

5.4 Hermite Basis

Following the methods in [1], [4], and [6] we use the *barycentric forms* of the Hermite interpolational basis: If the Hermite data $f^{(j)}(\tau_i)/j!$ is known for $1 \le i \le n$ and $0 \le j \le s_i - 1$, then the interpolational polynomial $f(t)$ may be represented (in a numerically stable fashion, if the confluencies s_i are not too large) as

$$f(t) = \beta(t) \sum_{i=1}^{n} \sum_{j=0}^{s_i-1} \sum_{k=0}^{j} \gamma_{ij} f^{(k)}(\tau_i)(t-\tau_i)^{k-j-1}/k!\ , \tag{9}$$

where $\beta(t)$ is given by (10) below,

$$\beta(z) = \prod_{i=1}^{n} (z-\tau_i)^{s_i}, \tag{10}$$

or $f(t)$ can be written in the "second barycentric form"

$$f(t) = \frac{\sum_{i=1}^{n}\sum_{j=0}^{s_i-1}\sum_{k=0}^{j} \gamma_{ij} f^{(k)}(\tau_i)(t-\tau_i)^{k-j-1}/k!}{\sum_{i=1}^{n}\sum_{j=0}^{s_i-1} \gamma_{ij}(t-\tau_i)^{-j-1}}\ . \tag{11}$$

Evaluating $f(t)$ or its derivatives exactly (to floating-point precision) at the nodes $t = \tau_i$ requires special treatment, but this turns out not to be difficult. The coefficients γ_{ij} appearing in those formulae were computed via divided differences in [13], but this is numerically unstable for some orderings of the nodes, and the local series method of [4] is recommended for stability.

6. EXAMPLES

6.1 Monomial Basis

Let $f = 4x^3 - 1, r = 1/2, q = 2$ and $w = [0, 1, 0, 2]$. Therefore we have $p = 2$. Now we let

$$\mathbf{u} = \mu[1, r, r^2, r^3] = \mu[1, 1/2, 1/4, 1/8]$$

where $\mu = \frac{1}{\|\mathbf{\Phi}(r)\|_{p,w}}$ and $\mathbf{\Phi}(r) = [1, r, r^2, r^3]$. So $\|\mathbf{u}\|_{p,w} = 1$. Considering the previous discussions we get the perturbation vector as below.

$$\Delta a_i = \mu|f(r)|v_i = \mu|f(r)|(\gamma w_i^p |u_i|^{(p-2)}\bar{u_i}) = [0, 4/5, 0, 4/5].$$

As seen where $w_i = 0$ the perturbation is zero as well and therefore the coefficient remains the same.

6.1.1 *How can we use the weight vector to make the perturbed polynomial monic?*

According to this method, for getting a monic result, we need to take two steps: Start with a monic polynomial (starting with a non-monic polynomial, we need to divide the polynomial by the leading coefficient in order to make it monic) and then make the last component of the weight vector, which corresponds to the leading coefficient, zero. This way, in fact we do not allow the routine to change the leading coefficient by making its corresponding perturbation zero.

6.1.2 *How can we use the weight vector to make the perturbed polynomial sparse?*

Using the same idea, for insisting on sparsity, we just need to define a suitable weight vector, instead of modifying the routine. Here is the recipe of such a vector.

$$w_k = \begin{cases} 1 & \text{if } f_k \neq 0 \\ 0 & \text{if } f_k = 0 \end{cases} \tag{12}$$

The meaning of $w_k = 0$ is that, the routine is not permitted to change f_k. And $w_k = 1$ means it is permitted.

Obviously we can change the nonzero entries of w to any wanted nonnegative number, according to what we wish to see. This way we can unify our routines for different mentioned problems and look at the weight vectors as a sign of the presence of some specific coefficients in the perturbed polynomial.

6.1.3 *Playing with Weights*

Consider the same sparse polynomial $f(x) = 1+5x-2x^{30}$ at the given zero $r = 1 - \frac{1}{2}i$. Let w be the weight vector such that

$$w_k = \begin{cases} 1 & \text{if } k = 0, 10, 30 \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

By letting $w_1 = 0$, we mean that we want $f_1 = 5$ to remain untouched. Now the routine is allowed to work on the values of $\{\tilde{f}_0, \tilde{f}_{10}, \tilde{f}_{10}\}$ for finding the perturbed polynomial.

Our routine NPGZ when $q = 1$ gives a polynomial very close to

$$\tilde{f} = 1 + 5x + (-0.13 - i0.18)x^{30}$$

for $q = 2$ a polynomial very close to

$$\tilde{f} = (1.00-i0.06)+5x+(-0.19-i0.01)x^{10}+(-0.15-i0.18)x^{30}$$

and for $q = \infty$ a polynomial very close to

$$\tilde{f} = (1.21-i1.62)+5x+(-1.63-i0.08)x^{10}+(-0.36-i0.16)x^{30}$$

In the polynomials found, as expected $\tilde{f}_1 = 5$. For each perturbed polynomial, we report $\|\Delta f\|_{k,w^*}$ for $k = 1$, $k = 2$, and $k = \infty$, in Table 6.1.3. The diagonal entries should be the smallest in each column.

If we change our weight vector to

$$w = [w_0, 0, \cdots, 0, w_{10}, 0, \cdots, 0, w_{30}] = [2, 0, \cdots, 0, 1, 0, \cdots, 0, 4] \tag{14}$$

then NPGZ gives the following results.
For $q = 1$ a polynomial very close to

$$\tilde{f} = 1 + 5x + (-0.13 - i0.18)x^{30}$$

for $q = 2$ a polynomial very close to

$$\tilde{f} = (1 - i0.01) + 5x - (0.01 + i0.0006)x^{10} - (0.13 + i0.18)x^{30}$$

Table 2: Distances between $f = -2x^{30} + 5x + 1$ and $\tilde{f}$ using weighted 1,2 and ∞ norms with the weight vector (13).

q	$\|\Delta f\|_{1,w^*}$	$\|\Delta f\|_{2,w^*}$	$\|\Delta f\|_{\infty,w^*}$
1	1.8758	1.8758	1.8758
2	2.1162	1.8640	1.8522
∞	4.9254	2.8437	1.6418

Table 3: Distances between $f = -2x^{30} + 5x + 1$ and $\tilde{f}$ using weighted norms with the weight vector (14).

q	$\|\Delta f\|_{1,w}$	$\|\Delta f\|_{2,w}$	$\|\Delta f\|_{\infty,w}$
1	0.4689	0.4689	0.4689
2	0.4893	0.4687	0.4684
∞	1.3470	0.7777	0.4490

and for $q = \infty$ a polynomial very close to

$$\tilde{f} = (1.11 - i0.89) + 5x - (0.44 + i0.02)x^{10} - (0.21 + i0.17)x^{30}$$

For each perturbed polynomial, we report $\|\Delta f\|_{k,w}$ for $k = 1$, $k = 2$, and $k = \infty$, in Table 6.1.3. The diagonal entries should be the smallest in each column.

6.2 Lagrange Basis

Suppose that we are given the values

$$\mathbf{f} = [-26/27, 8/9, -22/27, 28/27]$$

of a polynomial on the nodes $[-1, -1/3, 1/3, 1]$. Suppose also that we apply Newton's iteration (just once, to exaggerate the effect)

$$x_{n+1} = x_n + \frac{\sum_{k=0}^{3}(\beta_j f_j)/(x_n - z_j)}{\sum_{k=0}^{3}(\beta_j f_j)/(x_n - z_j)^2} \tag{15}$$

to the barycentric rational function $R(z) = f(z)/\ell(z)$, starting from an initial guess $x_0 = 0$, and find the root $x^* \approx 0.01234567903$. We *deflate* the polynomial by dividing each f_j by $(x^* - z_j)$, giving a new vector of values,

$$\mathbf{g} = [-.951220, 2.57143, 2.53846, -1.05] \,.$$

Obviously this is supposed to be of lower degree, and so we may throw away one data point. But as is well known for the monomial basis, deflation may introduce errors, and we might be better off by first applying NPGZ for finding a polynomial of lower degree. When we do this, we find

$$\hat{g} = [-.951204, 2.57141, 2.53848, 0.050015] \,,$$

which differs from the first by about $1.5 \cdot 10^{-5}$, and now we may throw away any data point without fear of loss of information because up to one ulp these are the values of a degree 2 polynomial on this set of nodes.

6.3 Bernstein Basis

Suppose the coefficients of a degree 5 Bernstein expansion on $[0, 1]$ are

$$[0., 50.097, -25.006, -25.200, 50.214, 0.] \,.$$

We suspect this is close to a degree 4 polynomial. If we simply apply the degree-reducing procedure, we get the vector $[0., 62.621, -83.423, 62.135, 0.]$. If we first apply NPLD, with weights given by the absolute values of the coefficients, then after degree-reduction we get $[0., 62.621, -83.423, 62.767, 0.]$. These results do not differ much, but they do differ, and indeed the original unfiltered polynomial has a term $2.5 \cdot x^3$ (after conversion to the monomial basis, which we avoid) which may not appear negligible unless carefully considered.

6.4 Hermite Basis

Suppose that $\tau = [-1, r, 1]$, and we know the values of the functions at these nodes and the derivative value at the second node. We take these values to be $y = [1, [1, 0], 1]$. Here, $\beta(z) = (z-1)(z-r)^2(z-1)$. Then $\beta(z)^{-1}$ has the partial fraction expansion

$$\frac{1}{(z-1)(z-r)^2(z-1)} = -1/2\,\frac{1}{(1+r)^2\,(z+1)}$$
$$-2\,\frac{r}{(1+r)^2\,(r-1)^2\,(z-r)} + 1/2\,\frac{1}{(r-1)^2\,(z-1)}$$
$$+\frac{1}{(1+r)\,(r-1)\,(z-r)^2}\,.$$

Now we can write

$$P(z) = \beta(z)\sum_{i=1}^{3}\sum_{j=0}^{s_i-1}\sum_{k=0}^{j}\gamma_{ij}\rho_{ik}(z-\tau_i)^{k-j-1}\,, \qquad (16)$$

where ρ_{ik} are the data values, i is the node index, and k represents the derivative level, i.e. $k = 0$ shows the function value and $k = 1$ represents the first derivative. If we expand $P(z)$, we find that the basis function are

$$\Phi_0(z) = \gamma_{10}(z-r)^2(z-1) \qquad (17)$$
$$\Phi_1(z) = \gamma_{20}(z-r)(z+1)(z-1) + \gamma_{21}(z+1)(z-1) \qquad (18)$$
$$\Phi_2(z) = \gamma_{21}(z-r)(z+1)(z-1) \qquad (19)$$
$$\Phi_3(z) = \gamma_{30}(z+1)(z-r)^2\,. \qquad (20)$$

If we normalize these basis functions (divide by z^3) and take the limit when $z \to \infty$, we get the barycentric weights.

In this example the coefficient of $\Phi_1(z)$ is assumed to be very uncertain. Therefore, we want to put less emphasis on this coefficient. If we choose the weight associated with this coefficient be very small, or equivalently, we choose a very large dual weight, then we can apply our NPGZ routine to get a zero at ∞ .

7. OBSERVATIONS ABOUT NUMERICAL STABILITY

The barycentric weights may vary widely in magnitude, in which case it is known that interpolation is very sensitive to data errors. Likewise, the elements of the vector of leading coefficients $(-1)^k\binom{n}{k}$ for the Bernstein case also vary widely in magnitude, and the reader ought to be concerned for the numerical stability of this process.

The one-row matrix formed by $B = [(-1)^k\binom{n}{k}]$ has only one singular value σ_1, and $\sigma_1 = O(2^n)$. Thus, errors in coefficients may be amplified by as much as 2^n on multiplication by this vector. One expects, in fact, that $Bv \approx 2^n\varepsilon$. But we are actually using the reverse of this process: we are looking for a vector of deltas that satisfy $B\Delta(f) = 0$, and hence we might expect to damp errors.

What really helps, though, is the fact that we have an analytic solution to this minimization problem. The computation of the components of $\mathbf{v}$ involves only powers, absolute values, multiplication, and division, and tests for zero. The size of the resulting $\Delta(f)$ is proportional to the residual $f(r)$ (for a finite root), and this may be subject to large relative errors if the residual is small, but we are going to add this change to f, and hence the effect of the large errors is muted.

8. CONCLUSIONS

We have presented an algorithm here that finds the nearest (in a weighted, parameterized vector coefficient norm) polynomial that satisfies a linear constraint. To do so, we have defined a dual norm and used the converse of the Hölder inequality. The algorithm has linear cost in the length of the input polynomial coefficient (assuming, for example, that the basis elements (and thus for example the barycentric weights) are known). One might think of using this iteratively as a first approximation for finding the nearest polynomial that satisfies several linear constraints, but if there are too many then a direct linear algebraic approach as in [9] would be preferable. One of the main contributions of this article is the correct definition of a weighted dual norm.

9. ACKNOWLEDGMENTS

Special thanks to NSERC and the University of Toronto for their support. We also thank Ken Jackson and Dhavide Aruliah for their help.

10. REFERENCES

[1] J.-P. Berrut and L. N. Trefethen. Barycentric Lagrange interpolation. *SIAM Review*, 46(3):501–517, 2004.

[2] R. M. Corless, P. M. Gianni, B. M. Trager, and S. M. Watt. The Singular Value Decomposition for polynomial systems. In A. Levelt, editor, *International Symposium on Symbolic and Algebraic Computation*, pages 195–207, Montréal, Canada, 1995. ACM.

[3] R. M. Corless and N. Rezvani. The nearest polynomial of lower degree. In *Proceedings of the 2007 international workshop on Symbolic-numeric computation*, SNC '07, pages 199–200, New York, NY, USA, 2007. ACM.

[4] R. M. Corless, A. Shakoori, D. Aruliah, and L. Gonzalez-Vega. Barycentric Hermite interpolants for event location in initial-value problems. *Journal of Numerical Analysis, Industrial, and Applied Mathematics*, 3(1–2):1–18, 2008.

[5] R. T. Farouki and V. T. Rajan. Algorithms for polynomials in Bernstein form. *Comput. Aided Geom. Des.*, 5(1):1–26, 1988.

[6] N. J. Higham. The numerical stability of barycentric Lagrange interpolation. *IMA Journal of Numerical Analysis*, 24:547–556, 2004.

[7] M. A. Hitz and E. Kaltofen. Efficient algorithms for computing the nearest polynomial with constrained roots. In O. Gloor, editor, *International Symposium on Symbolic and Algebraic Computation*, pages 236–243, Rostock, Germany, 1998. ACM.

[8] M. A. Hitz, E. Kaltofen, and Y. N. Lakshman. Efficient algorithms for computing the nearest polynomial with a real root and related problems. In

Proceedings of the 1999 international symposium on Symbolic and algebraic computation, ISSAC '99, pages 205–212, New York, NY, USA, 1999. ACM.

[9] E. Kaltofen, Z. Yang, and L. Zhi. Approximate greatest common divisors of several polynomials with linearly constrained coefficients and singular polynomials. In *Proceedings of the 2006 international symposium on Symbolic and algebraic computation*, ISSAC '06, pages 169–176, New York, NY, USA, 2006. ACM.

[10] N. Rezvani and R. M. Corless. The nearest polynomial with a given zero, revisited. SIGSAM BULLETIN, *Communications on Computer Algebra*, 134(3):71–76, September 2005.

[11] N. Rezvani Dehaghani. Approximate polynomials in different bases. Master's thesis, University of Western Ontario, December 2005.

[12] T. Rivlin. *Chebyshev polynomials: from approximation theory to number theory.* Wiley, 1990.

[13] C. Schneider and W. Werner. Hermite interpolation: the barycentric approach. *Computing*, 46:35–51, 1991.

[14] H. J. Stetter. The nearest polynomial with a given zero, and similar problems. *SIGSAM Bull.*, 33:2–4, December 1999.

[15] L. N. Trefethen. *Spectral Methods in* MATLAB. SIAM, 2000.

[16] P. Woźny and S. Lewanowicz. Multi-degree reduction of Bézier curves with constraints, using dual Bernstein basis polynomials. *Computer Aided Geometric Design*, 26(5):566 – 579, 2009.

APPENDIX

A. THE NPGZ ALGORITHM

This algorithm was discussed in [11] and [10] in detail.
Input: Polynomial $f = \mathbf{a} \cdot \mathbf{\Phi}(x) \in \mathbb{P}^n$, spanned by a basis Φ where $\mathbf{a}$ is the vector of coefficients, a given zero r and the q-norm in which we wish to minimize the perturbations.
Output: The perturbed polynomial $\tilde{f}$ such that $\|f - \tilde{f}\|_q$ is minimal.

1. Compute $p = q/(q-1)$.
2. Let $\mathbf{u} = \mu\mathbf{\Phi}(r)$ where $\mu = \frac{1}{\|\mathbf{\Phi}(r)\|_p}$.
3. Compute the minimal perturbations using Proposition 1. For $p \neq \infty$
$$\Delta a_k = \begin{cases} \mu|f(r)|\gamma|u_k|^{p-2}\bar{u}_k & \text{if } u_k \neq 0 \\ 0 & \text{if } u_k = 0 \end{cases}, \tag{21}$$
and for $p = \infty$
$$\Delta a_k = \begin{cases} 0 & \text{if } k \neq k_0 \\ \mu|f(r)|\gamma\bar{u}_{k_0} & \text{if } k = k_0 \end{cases}, \tag{22}$$
where $\gamma = -\text{signum}(f(r)) = -\frac{f(r)}{|f(r)|}$ and k_0 is any, say the least, index with $|u_{k_0}| = 1$; there must be at least one such, because $\|\mathbf{u}\|_\infty = 1$. [The ambiguity if there are more than one such k_0 was addressed in [11, 10].] Then
$$\|\mathbf{v}\|^* = \|\mathbf{v}\|_q = 1$$
and $\mathbf{v} \cdot \mathbf{u} = \gamma$ so $|\mathbf{v} \cdot \mathbf{u}| = 1$.
4. For symbolic x, return $\tilde{f} = (\mathbf{a} + \Delta\mathbf{a}) \cdot \mathbf{\Phi}(x)$.

PROPOSITION 1. *If $1 \leq p < \infty$ and $\mathbf{u}$ is such that $\|\mathbf{u}\|_p = 1$, define $\mathbf{v}$ so that*
$$v_k = \begin{cases} \gamma|u_k|^{p-2}\bar{u}_k & \text{if } u_k \neq 0 \\ 0 & \text{if } u_k = 0 \end{cases} \tag{23}$$
where γ is an arbitrary constant with $|\gamma| = 1$ and $\bar{u}_k$ is the conjugate of u_k.

If $p = \infty$, take instead
$$v_k = \begin{cases} 0 & \text{if } k \neq k_0 \\ \gamma\bar{u}_{k_0} & \text{if } k = k_0 \end{cases} \tag{24}$$
where k_0 is any, say the least, index with $|u_{k_0}| = 1$; there must be at least one such, because $\|\mathbf{u}\|_\infty = 1$. [The ambiguity if there are more than one such k_0 was not discussed in [14].] Then
$$\|\mathbf{v}\|^* = \|\mathbf{v}\|_q = 1$$
and $\mathbf{v} \cdot \mathbf{u} = \gamma$ so $|\mathbf{v} \cdot \mathbf{u}| = 1$.

Arrangement Computation for Planar Algebraic Curves

Eric Berberich
Max-Planck-Institut für Informatik
Campus E1 4
66123 Saarbrücken, Germany
eric@mpi-inf.mpg.de

Pavel Emeliyanenko
Max-Planck-Institut für Informatik
Campus E1 4
66123 Saarbrücken, Germany
asm@mpi-inf.mpg.de

Alexander Kobel
Max-Planck-Institut für Informatik
Campus E1 4
66123 Saarbrücken, Germany
akobel@mpi-inf.mpg.de

Michael Sagraloff
Max-Planck-Institut für Informatik
Campus E1 4
66123 Saarbrücken, Germany
msagralo@mpi-inf.mpg.de

ABSTRACT

We present a new *certified* and *complete* algorithm to compute arrangements of real planar algebraic curves. Our algorithm provides a geometric-topological analysis of the decomposition of the plane induced by a finite number of algebraic curves in terms of a cylindrical algebraic decomposition of the plane. Compared to previous approaches, we improve in two main aspects: Firstly, we significantly limit the types of involved exact operations, that is, our algorithms only use resultant and gcd computations as purely symbolic operations. Secondly, we introduce a new hybrid method in the lifting step of our algorithm which combines the use of a certified numerical complex root solver and information derived from the resultant computation. Additionally, we never consider any coordinate transformation and the output is also given with respect to the initial coordinate system.

We implemented our algorithm as a prototypical package of the C++-library CGAL. Our implementation exploits graphics hardware to expedite the resultant and gcd computation. We also compared our implementation with the current reference implementation, that is, CGAL's curve analysis and arrangement for algebraic curves. For various series of challenging instances, our experiments show that the new implementation outperforms the existing one.

Categories and Subject Descriptors

I.1.2 [**Symbolic and Algebraic Manipulation**]: Algorithms; G.4 [**Mathematical Software**]: Reliability and Robustness; D.2.13 [**Software Engineering**]: Reusable Software

General Terms

Algorithms, Experimentation, Performance, Reliability

Keywords

Algebraic curves, topology, arrangements, certified algorithms

1. INTRODUCTION

Computing the topology of a planar algebraic curve

$$C = V(f) = \{(x, y) \in \mathbb{R}^2 : f(x, y) = 0\} \qquad (1.1)$$

can be considered as one of the fundamental problems in real algebraic geometry with numerous applications in computational geometry, computer graphics and computer aided geometric design. Typically, the topology of C is given in terms of a planar graph $\mathcal{G}_C$ embedded in $\mathbb{R}^2$ that is isotopic to C. For a geometric-topological analysis, we further require the vertices of $\mathcal{G}_C$ to be located on C. In this paper, we study the general problem of computing an arrangement of a given set of algebraic curves, that is, the decomposition of the plane into cells of dimensions 0, 1 and 2 induced by the given curves. The proposed algorithm is *certified* and *complete*, and the overall arrangement computation is exclusively carried out in the initial coordinate system. Efficiency of our approach is shown by implementing our algorithm and comparing it to the current reference implementation.

There exist a number of certified and complete approaches to determine the topology of an algebraic curve; we refer the reader to [12, 17, 22, 24, 29] for recent work and further references. At present, only the method from [17] has been extended to arrangement computations of arbitrary algebraic curves [16]. Common to all existing approaches is that, in a first step, they use eliminations techniques (e.g., resultants) to project the x-critical points (i.e., points $p \in C$ with $f_y(p) = 0$) of the curve into one dimension. In a second step, the fiber at each of these projected points is computed. In general, this lifting step has turned out to be the most time-consuming part because it amounts to determining the real roots of a non-square univariate polynomial $f(\alpha, y) \in \mathbb{R}[y]$ with algebraic coefficients. The high computational cost for computing the roots of $f(\alpha, y)$ is mainly due to more comprehensive algebraic machinery such as subresultants (in [16, 17, 22]), Gröbner basis or rational univariate representation (in [12]) in order to obtain additional information on the number of distinct real (or complex) roots of $f(\alpha, y)$ or the multiplicity of the multiple roots of $f(\alpha, y)$. In addition, all except the method from [12] consider a shearing of the curve which guarantees that the sheared curve has no two x-critical points sharing the same x-coordinate which in turn simplifies the lifting step but for the price of giving up sparseness of the initial input; an approach, which usually results in larger bitsizes of the coefficients and considerably increased running times. In a final connection step, arcs incident to the same x-critical points are identified.

The high-level description of the algorithm presented in this paper is almost identical to that of the existing methods, and, similar to [16, 17], we reduce the arrangement computation to the geometric-topological analysis of a single curve and of a pair of curves. However, we improve the analyses in the following two ways: Firstly, we considerably reduce the amount of purely symbolic computations, that is, we only use resultant and gcd computation. The main reason for this approach is that we can outsource both computations to graphics hardware [21, 19, 20], removing a bottleneck of previous methods which was due to the high amount of symbolic operations. Secondly, for curve analysis, we use a result from Teissier [23, 36] to obtain additional information for the number of distinct complex roots of $f(\alpha, y)$ along a critical fiber (actually, an upper bound which matches the exact number in most cases; see Section 2.3 for details). We combine this information with a new certified complex root solver [25] to isolate the roots of $f(\alpha, y)$. The latter symbolic-numeric step applies as an efficient filter denoted FastLift which fails only in very special instances. Each failure can be recognized and in order to achieve completeness of our overall method (i.e., for cases where FastLift fails), we modify the method from [4] for solving bivariate polynomial systems in order to isolate the roots of $f(\alpha, y)$.

We implemented our algorithm as a development branch of Cgal's[1] bivariate algebraic kernel (Ak_2 for short) which is based on the algorithms from [16, 17] for topology and arrangement computation. Intensive benchmarks [17, 29] have shown that Ak_2 can be considered as the current reference implementation. For fair comparison, we run an Ak_2 with GPU-enabled resultants and gcds against our implementation on numerous challenging benchmark instances. Our experiments show that the new curve analysis outperforms Ak_2 for all instances. More precisely, our method is, on average, twice as fast for easy instances such as non-singular curves in generic position. For hard instances, we typically improve by large factors between 5 and 120 which is mainly due to the new symbolic-numeric filter FastLift, the exclusive use of resultant and gcd computations as the only symbolic operations and the absence of shearing. Computing arrangements mainly benefit from the improved curve-analyses and from avoiding subresultants and shears for harder instances. In summary, the presented approach demonstrates the strength of symbolic-numeric techniques. It further proves that, in order to achieve practical efficiency, it is of great importance to reduce the amount of symbolic operations and to consider approximate operations whenever this is possible.

[1]Computational Geometry Algorithms Library, `www.cgal.org`; see also `http://exacus.mpi-inf.mpg.de/cgi-bin/xalci.cgi` for an online demo on arrangement computation.

Finally, we are confident that our new approach will have positive impacts in the following respects: There exist several non-certified (or non-complete) approaches either based on subdivision [2, 11, 28, 30, 34, 35] or homotopy methods [27]. They show excellent behavior for most inputs. However, in order to guarantee exactness for all possible inputs (e.g., singular curves), additional certification steps (e.g., worst case separation bounds for subdivision methods) have to be considered, an approach which has not shown to be effective in practice so far. An advantage of the latter methods, compared to elimination approaches, is that they are local and do not need (global) algebraic operations. In our method, we considerably reduced the amount of such algebraic operations and outsourced the remaining computations to graphics card. Hence, it seems reasonable that combining our algorithm with a subdivision or a homotopy approach eventually leads to a certified and *complete* method which shows excellent "local" behavior as well. We further see numerous applications of our method, in particular, when computing arrangements of surfaces. The actual implementation [7] for surface triangulation is crucially based on planar arrangement computations of singular curves. Thus, we are confident that the algorithm's efficiency can be considerably improved by using the new algorithm for planar arrangement computation.

2. CURVE ANALYSIS

2.1 The Algorithm

The input of our algorithm is a planar algebraic curve C as defined in (1.1), where f is a *square-free*, bivariate polynomial $f \in \mathbb{Z}[x, y]$ with integer coefficients. If f is considered as polynomial in y with coefficients $f_i(x) \in \mathbb{Z}[x]$, its coefficients typically share a trivial content $h := \gcd(f_0, f_1, \ldots)$. A non-trivial content $h \notin \mathbb{Z}$ defines vertical lines at the real roots of h. Our algorithm handles this situation by dividing out h first and finally merging the vertical lines defined by $h = 0$ and the analysis of the curve $C' := V(f/h)$ at the end of the algorithm; see [24] for details. Hence, throughout the following considerations, we can assume that h is trivial, thus C contains no vertical line.

The algorithm returns a planar graph $\mathcal{G}_C$ that is isotopic[2] to C, where all vertices V of $\mathcal{G}_C$ are located on C. The proposed algorithm follows a classical cylindrical algebraic decomposition approach. We start with a high-level description of the three-step algorithm.

In the first step, the ***projection phase***, we project all *x-critical points* $(\alpha, \beta) \in C$ (i.e., $f(\alpha, \beta) = f_y(\alpha, \beta) = 0$) onto the x-axis by means of a resultant computation and root isolation for the elimination polynomial. The set of x-critical points comprises exactly the points where C has a vertical tangent or is singular. It is well known (e.g., see [24, Theorem 2.2.10] for a short proof) that, for any two consecutive x-critical values α and α', C is *delineable* over $I = (\alpha, \alpha')$, that is, $C|_{I\times\mathbb{R}}$ decomposes into a certain number m_I of disjoint function graphs $C_{I,1}, \ldots, C_{I,m_I}$. In the ***lifting phase***, we first isolate the roots of the (square-free) *intermediate polynomial* $f(q_I, y) \in \mathbb{Q}[y]$, where q_I constitutes an arbitrary chosen but fixed rational value in I. This computation yields the number m_I (= number of real roots of $f(q_I, y)$) of arcs

[2]$\mathcal{G}_C$ is isotopic to C if there exists a continuous mapping $\phi : [0, 1] \times C \mapsto \mathbb{R}^2$ with $\phi(0, C) = C$, $\phi(1, C) = \mathcal{G}_C$ and $\phi(t_0, .) : C \mapsto \phi(t_0, C)$ a homeomorphism for each $t_0 \in [0, 1]$.

above I and corresponding representatives $(q_I, y_{I,i}) \in C_{I,i}$ on each arc. We further compute all points on C that are located above an x-critical value α, that is, we determine the real roots $y_{\alpha,1}, \ldots, y_{\alpha,m_\alpha}$ of each (non square-free) *fiber polynomial* $f(\alpha, y) \in \mathbb{R}[y]$. From the latter two computations, we obtain the vertex set V of $\mathcal{G}_C$ as the union of all points $(q_I, y_{I,i})$ and $(\alpha, y_{\alpha,i})$. In the final ***connection phase***, which concludes the topology analysis, we determine which of the above vertices are connected via an arc of C. For each connected pair $(v_1, v_2) \in V$, we insert a line segment connecting v_1 and v_2. It is then straightforward to prove that $\mathcal{G}_C$ is isotopic to C; see [24, Theorem 6.4.4] for a proof. We remark that we never consider any kind of coordinate transformation, even in case where C contains two or more x-critical points sharing the same x-coordinate.

We next describe the three phases in detail:

2.1.1 Projection Phase

In the projection step, we follow well-known techniques from elimination theory, that is, we compute the resultant $R(x) := \operatorname{res}(f, f_y; y) \in \mathbb{Z}[x]$ and a square-free factorization of R. More precisely, we determine square-free and pairwise coprime factors $r_i \in \mathbb{Z}[x]$, $i = 1, \ldots, \deg(R)$, such that $R(x) = \prod_{i=1}^{\deg(R)} (r_i(x))^i$. We remark that, for some $i \in \{1, \ldots, \deg(R)\}$, $r_i(x) \equiv 1$. Yun's algorithm [38, Alg. 14.21] constructs such a square-free factorization by essentially computing greatest common divisors of R and its higher derivatives in an iterative way. Next, we isolate the real roots $\alpha_{i,j}$, $j = 1, \ldots, \ell_i$, of the polynomials r_i which in turn are i-fold roots of R. More precisely, we compute disjoint intervals $I(\alpha_{i,j}) \subset \mathbb{R}$ with rational endpoints such that $I(\alpha_{i,j})$ contains $\alpha_{i,j}$ but no other root of r_i, and the union of all $I(\alpha_{i,j})$, $j = 1, \ldots, \ell_i$, contains all real roots of r_i. For isolating the real roots, we consider the Descartes method [15, 31] as a well-suited algorithm. By further refining the isolating intervals, we can achieve that all intervals $I(\alpha_{i,j})$ are pairwise disjoint and, thus, also constitute isolating intervals for the real roots of R. Then, for each pair α and α' of consecutive roots of R defining an open interval $I = (\alpha, \alpha')$, we choose a separating rational value q_I in between the corresponding isolating intervals.

2.1.2 Lifting Phase

Isolating the roots of the intermediate polynomials $f(q_I, y)$ is rather straightforward: since $f(q_I, y)$ is a square-free polynomial with rational coefficients, the Descartes method directly applies. Determining the roots of $f(\alpha, y) \in \mathbb{R}[y]$ at an x-critical value α is more complicated because $f(\alpha, y)$ has multiple roots and, in general, irrational coefficients. We propose to run the following approach: We first consider a method denoted FastLift which works as a filter for the fiber computation. We will see in the experiments enlisted in Section 4 that FastLift applies to all fibers for the majority of all input curves and only fails for a small number of fibers for some very special instances. In case of success, the fiber at α is returned. If FastLift fails, we use a second method denoted Lift which serves as a "backup" for FastLift. In comparison to FastLift, Lift is a complete method which applies to any input curve and any corresponding x-critical value. The price is, however, that Lift is less efficient. Nevertheless, our experiments show that even the exclusive use of Lift significantly improves upon existing approaches.

Lift — *a complete method for fiber computation*: Lift is based on our recent studies on solving a bivariate polynomial system. In [4], we introduced a highly efficient method, denoted Bisolve, to isolate the real solutions of a system of two bivariate polynomials $f, g \in \mathbb{Q}[x, y]$. Its output consists of a set of disjoint boxes $B_1, \ldots, B_m \subset \mathbb{R}^2$ such that each box B_i contains exactly one real solution $\xi := (x_0, y_0)$ of $f(x, y) = g(x, y) = 0$, and the union of all B_i covers all solutions. Furthermore, for each solution ξ, Bisolve provides square-free polynomials $p, q \in \mathbb{Z}[x]$ with $p(x_0) = q(y_0) = 0$ and corresponding isolating (and refineable) intervals $I(x_0)$ and $I(y_0)$ for x_0 and y_0, respectively. Comparing ξ with another point $\xi_1 = (x_1, y_1) \in \mathbb{R}^2$ given by a similar representation is rather straightforward. Namely, let $\tilde{p}, \tilde{q} \in \mathbb{Z}[x]$ be corresponding defining square-free polynomials and $I(x_1)$ and $I(y_1)$ isolating intervals for x_1 and y_1, respectively, then we can compare the x- and y-coordinates of ξ and $\tilde{\xi}$ via gcd-computation of the defining univariate polynomials and sign evaluation at the endpoints of the isolating intervals (see [3, Algorithm 10.44] for more details).

In order to compute the fiber at an x-critical value α of C, we proceed as follows: We first use Bisolve to determine all solutions $p_i = (\alpha, \beta_i)$, $i = 1, \ldots, l$, of the system $f = f_y = 0$ with x-coordinate α. Then, for each p_i, we compute

$$k_i := \min\{k : f_{y^k}(\alpha, \beta_i) = \frac{\partial^k f}{\partial y^k}(\alpha, \beta_i) \neq 0\} \geq 2.$$

The latter computation is done by iteratively calling Bisolve for $f_y = f_{y^2} = 0$, $f_{y^2} = f_{y^3} = 0$, etc., and sorting the solutions along the vertical line $x = \alpha$. We eventually obtain disjoint intervals $I_1, \ldots, I_l$ and corresponding multiplicities $k_1, \ldots, k_l$ such that β_j is a k_j-fold root of $f(\alpha, y)$ which is contained in I_j. The intervals I_j already separate the roots β_j from any other multiple root of $f(\alpha, y)$, however, I_j might still contain ordinary roots of $f(\alpha, y)$. Hence, we further refine each I_j until we can guarantee via interval arithmetic that $\frac{\partial^{k_j} f}{\partial y^{k_j}}(\alpha, y)$ does not vanish on I_j. If this condition is fulfilled, then I_j cannot contain any root of $f(\alpha, y)$ except β_j due to the mean value theorem, thus, I_j is isolating.

After refining all intervals I_j, it remains to isolate the ordinary roots of $f(\alpha, y)$. For this purpose, we use the so-called *Bitstream Descartes* isolator [18] (Bdc for short) which can be considered as a variant of the Descartes method working on polynomials with interval coefficients. This method can be used to get arbitrary good approximations of the real roots of a polynomial with "bitstream" coefficients, that is, coefficients that can be approximated to arbitrary precision. Bdc starts from an interval guaranteed to contain all real roots of a polynomial and proceeds with interval subdivisions giving rise to a *subdivision tree.* Accordingly, the approximation precision for the coefficients is increased in each step of the algorithm. Each leaf of the tree is associated with an interval I and stores a lower bound $l(I)$ and an upper bound $u(I)$ on the number of real roots within this interval based on Descartes' Rule of Signs. Hence, $u(I) = 0$ implies that I contains no root and thus can be discarded. If $l(I) = u(I) = 1$, then I is an *isolating interval* for a simple root. Intervals with $u(I) > 1$ are further subdivided. We remark that, after a number of iterations, Bdc isolates all simple roots of a bitstream polynomial, and intervals not containing any root are eventually discarded. For a multiple

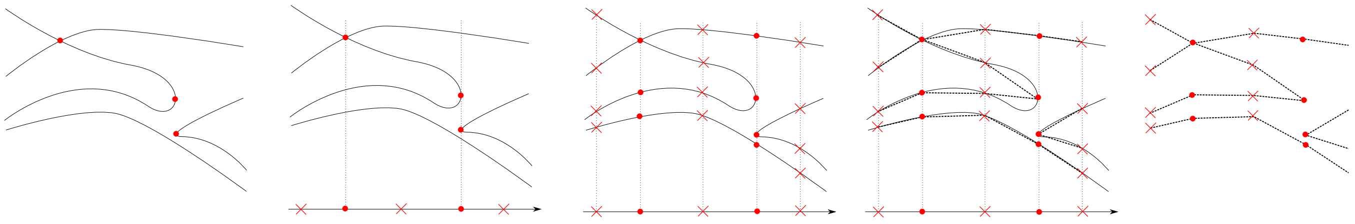


Figure 2.1: The figure on the left shows a curve C with two x-extremal points and one singular point (red dots). In the *projection phase*, these points are projected onto the x-axis and rational points separating the x-critical values are inserted (red crosses). In the *lifting phase*, the fibers at the critical values (red dots) and at the rational points in between (red crosses) are computed. In the *connection phase*, each pair of lifted points connected by an arc of C is determined and a corresponding line segment is inserted. The right figure shows the final graph that is isotopic to C.

root ξ, BDC constructs intervals I which approximate ξ to an arbitrary good precision but never certifies such an interval I to be isolating.

In our situation, we have already isolated the multiple roots of $f(\alpha, y)$ by the intervals I_j. It remains to isolate the simple roots of $f(\alpha, y)$. Therefor, we consider the following modification of BDC : We discard an interval I if one of following three cases applies: i) $u(I) = 0$, or ii) I is completely contained in one of the intervals I_j, or iii) I contains an interval I_j and $u(I) \leq k_j$. Namely, in each of these situations, I cannot contain an ordinary root of $f(\alpha, y)$. An interval I is stored as isolating for an ordinary root of $f(\alpha, y)$ if $l(I) = u(I) = 1$ and I intersects no interval I_j. All intervals which do not fulfill one of the above conditions are further subdivided. In a last step, we sort the intervals I_j (isolating the multiple roots) and the isolating intervals for the ordinary roots along the vertical line.

We remark that BISOLVE applied in LIFT reuses the resultant obtained in the projection phase of the algorithm. Furthermore, it is a local approach in the sense that its cost is proportional to the number of x-critical fibers considered. This will turn out to be beneficial in the overall approach where most fibers can successfully be treated by FASTLIFT:

FASTLIFT — *a fast method for fiber computation*: FASTLIFT is a hybrid method to isolate all complex roots and, thus, also the real roots of $f(\alpha, y)$, where α is an x-critical value of C. It combines a numerical solver to compute arbitrary good approximations (i.e., complex discs in $\mathbb{C}$) of the roots, an exact certification step to certify the existence of roots within the computed discs and the following result due to Teissier [23, 36]:

LEMMA 1 (TEISSIER). *For an x-critical point $p = (\alpha, \beta)$ of $C = V(f)$, it holds that*

$$\operatorname{mult}(f(\alpha, y), \beta) = \operatorname{Int}(f, f_y, p) - \operatorname{Int}(f_x, f_y, p) + 1, \quad (2.1)$$

where $\operatorname{mult}(f(\alpha, y), \beta)$ denotes the multiplicity of β as root of $f(\alpha, y) \in \mathbb{R}[y]$, $\operatorname{Int}(f, f_y, p)$ the intersection multiplicity[3] *of the curves implicitly defined by $f = 0$ and $f_y = 0$ at p, and $\operatorname{Int}(f_x, f_y, p)$ the intersection multiplicity of $f_x = 0$ and $f_y = 0$ at p.*

Remark. In the case, where f_x and f_y share a common non-trivial factor $h = \gcd(f_x, f_y) \in \mathbb{Z}[x, y]$, h does not vanish on any x-critical point p of C. Namely, $h(p) = 0$ would imply that $\operatorname{Int}(f_x, f_y, p) = \infty$ and, thus, $\operatorname{Int}(f, f_y, p) = \infty$ as well, a contradiction to our assumption on f to be square-free. Hence, we have $\operatorname{Int}(f_x, f_y, p) = \operatorname{Int}(f_x^*, f_y^*, p)$ with $f_x^* := f_x/h$ and $f_y^* := f_y/h$ and, thus, the following formula (which is equivalent to (2.1) for trivial h) applies:

$$\operatorname{mult}(f(\alpha, y), \beta) = \operatorname{Int}(f, f_y, p) - \operatorname{Int}(f_x^*, f_y^*, p) + 1, \quad (2.2)$$

We come to the description of FASTLIFT. In the first step, we determine an upper bound m_α^* for the actual number m_α of distinct complex roots of $f(\alpha, y)$ which matches m_α in most situations. We distinguish the cases $\deg f(\alpha, y) \neq \deg_y f$ and $\deg f(\alpha, y) = \deg_y f$. In the first case, C has a vertical asymptote at α. Then, we define $m_\alpha^* := \deg f(\alpha, y)$ which is obviously an upper bound for m_α. If C is in generic position, $f(\alpha, y)$ has only ordinary roots and, thus, $m_\alpha^* = m_\alpha$.

We now consider the case $\deg f(\alpha, y) = \deg_y f$. Then, due to the formula (2.2), we have

$$\begin{aligned}
m_\alpha &= \#\{\text{distinct complex roots of } f(\alpha, y)\} \\
&= \deg_y f - \deg \gcd(f(\alpha, y), f_y(\alpha, y)) \\
&= \deg_y f - \sum_{\beta \in \mathbb{C}: f(\alpha,\beta)=0} (\operatorname{mult}(f(\alpha, y), \beta) - 1) \\
&= \deg_y f - \sum_{\substack{\beta \in \mathbb{C}: \\ (\alpha,\beta) \text{ is } x\text{-critical}}} \left(\operatorname{Int}(f, f_y, (\alpha, \beta)) - \operatorname{Int}(f_x^*, f_y^*, (\alpha, \beta))\right) \\
&= \deg_y f - \operatorname{mult}(R, \alpha) + \sum_{\substack{\beta \in \mathbb{C}: \\ (\alpha,\beta) \text{ is } x\text{-critical}}} \operatorname{Int}(f_x^*, f_y^*, (\alpha, \beta)) \quad (2.3) \\
&\leq \deg_y f - \operatorname{mult}(R, \alpha) + \sum_{\beta \in \mathbb{C}} \operatorname{Int}(f_x^*, f_y^*, (\alpha, \beta)) \quad (2.4) \\
&= \deg_y f - \operatorname{mult}(R, \alpha) + \operatorname{mult}(Q, \alpha) =: m_\alpha^* \quad (2.5)
\end{aligned}$$

where $R(x) = \operatorname{res}(f, f_y; y)$ and $Q(x) := \operatorname{res}(f_x^*, f_y^*; y)$. The equality (2.3) is due to the fact that f has no vertical asymptote at α and, thus, the multiplicity $\operatorname{mult}(R, \alpha)$ equals the sum $\sum_{\beta \in \mathbb{C}} \operatorname{Int}((f, f_y, (\alpha, \beta))$ of the intersection multiplicities of f and f_y in the fiber at α. (2.5) follows by an analogous argument for the intersection multiplicities of f_x^* and f_y^* along the vertical line at α. From the square-free factorization of R, we already know $\operatorname{mult}(R, \alpha)$, and $\operatorname{mult}(Q, \alpha)$ can be determined by computing Q, its square-free factorization and checking whether α is a root of one of the factors. If the curve C is in generic position,[4] the inequality (2.4) becomes

[3]The intersection multiplicity of two curves $f = 0$ and $g = 0$ at a point p is defined as the dimension of the localization of $\mathbb{C}[x, y]/(f, g)$ at p, considered as a $\mathbb{C}$-vector space.

[4]The reader may notice that generic position is used in a different context here. It is required that all intersection points of f_x^* and f_y^* above α are located on the curve C. We remark that this requirement is usually fulfilled even if there are two x-critical values sharing the same x-coordinate.

an equality because then f_x^* and f_y^* do not intersect in any point above α which is not located on C. Thus, in this case, we have $m_\alpha = m_\alpha^*$.

In the second step of FastLift, we aim to isolate all (complex) roots of $f(\alpha, y)$. The above computation of an upper bound m_α^* motivates the following ansatz: In order to determine the roots of $f(\alpha, y)$, we combine a numerical complex solver for $f(\alpha, y)$ and an exact certification step. More precisely, we use the numerical solver to determine disjoint discs $D_1, \ldots, D_m$ in the complex space and an exact certification step to certify the existence of a certain number $m_i \geq 1$ of roots (counted with multiplicity) of $f(\alpha, y)$ within each D_i; see Section 2.2 for details. Increasing the working precision and the number of iterations for the numerical solver eventually leads to arbitrary well refined discs D_i – but without a guarantee that these discs are actually isolating! Since m_α^* constitutes an upper bound on the number of distinct complex roots of $f(\alpha, y)$, we must have $m \leq m_\alpha \leq m_\alpha^*$ at any time. Hence, if the number of discs m equals the upper bound m_α^*, we know for sure that all complex roots of $f(\alpha, y)$ are isolated. Then, the isolating discs $D_1, \ldots, D_m$ are further refined until, for all $i = 1, \ldots, m$,

$$D_i \cap \mathbb{R} = \emptyset \text{ or } \bar{D}_i \cap D_j = \emptyset \text{ for all } j \neq i, \tag{2.6}$$

where $\bar{D}_i := \{\bar{z} : z \in D_i\}$ denotes the complex conjugate of D_i. The latter condition guarantees that each disc D_i which intersects the real axis actually isolates a real root of $f(\alpha, y)$. In addition, for each real root isolated by some D_i, we further obtain its multiplicity m_i as a root of $f(\alpha, y)$.

If $m \neq m_\alpha^*$, either one of the roots of $f(\alpha, y)$ is still not isolated or it holds that $m_\alpha < m_\alpha^*$. In the case that we do not succeed, that is, we still have $m < m_\alpha^*$ after several iterations with increasing precision,[5] we stop FastLift and proceed with Lift. In Section 2.3, we discuss the few failure cases.

We remark that the use of Teissier's formula is not entirely new when computing the topology of algebraic curves. In [12, 29], the formula (2.1) was used in its simplified form to compute $\operatorname{mult}(\beta, f(\alpha, y))$ for a non-singular point $p = (\alpha, \beta)$ (i.e., $\operatorname{Int}(f_x, f_y, p) = 0$). In contrast, we use the formula in its general form and sum up the information along the entire fiber which eventually leads to the upper bound m_α^* on the number of distinct complex roots of $f(\alpha, y)$.

2.1.3 Connection Phase

Let us consider a fixed x-critical value α, the corresponding isolating interval $I(\alpha) = (a, b)$ computed in the projection phase and the points $p_i := (\alpha, y_{\alpha,i}) \in C$, $i = 1, \ldots, m_\alpha$, located on C above α. Furthermore, let $I = (\alpha, \alpha')$ be the interval connecting α with the nearest x-critical value to the right of α (or $+\infty$ if none exists) and A_j, $j = 1, \ldots, m_I$, the j-th arc of C above I with respect to vertical ordering. A_j is represented by a point $a_j := (q_I, y_{I,j}) \in C$, where $y_{I,j}$ denotes the j-th real root of $f(q_I, y)$ and q_I an arbitrary but fixed rational value in I. To its left, A_j is either connected to $(\alpha, \pm\infty)$ (in case of a vertical asymptote) or to one of the points p_i. In order to determine the point to which an arc A_j is connected, we consider the following two distinct cases:

- The *generic case*, in which there exists exactly one real x-critical point p_{i_0} above α and $\deg f(\alpha, y) = \deg_y f$. The latter condition implies that C has no vertical asymptote at α. Then, the points $p_1, \ldots, p_{i_0-1}$ must be connected with $A_1, \ldots, A_{i_0-1}$ in buttom-up fashion, respectively, since, for each of these points, there exists a single arc of C passing this point. The same argument shows that $p_{i_0+1}, \ldots, p_{m_\alpha}$ must be connected to $A_{m_I - m_\alpha + i_0 + 1}, \ldots, A_{m_I}$ in top-down fashion, respectively. Finally, the remaining arcs in between must all be connected to the x-critical point p_{i_0}.
- The *non-generic case*: We choose arbitrary rational values $t_1, \ldots, t_{m_\alpha+1}$ with $t_1 < y_{\alpha,1} < t_2 < \ldots < y_{\alpha,m_\alpha} < t_{m_\alpha+1}$. Then, the points $\tilde{p}_i := (\alpha, t_i)$ separate the p_i's from each other. Computing such $\tilde{p}_i$ is easy since we have isolating intervals with rational endpoints for each of the roots $y_{\alpha,i}$ of $f(\alpha, y)$. In a second step, we use interval arithmetic to obtain intervals $\mathfrak{B}f(I(\alpha) \times t_i)$ with $f(I(\alpha) \times t_i) \subset \mathfrak{B}f(I(\alpha) \times t_i)$. As long as there exists an i with $0 \in \mathfrak{B}f(I(\alpha) \times t_i)$, we refine $I(\alpha)$. Since none of the $\tilde{p}_i$ is located on C, we eventually obtain a sufficiently refined interval $I(\alpha)$ with $0 \notin \mathfrak{B}f(I(\alpha) \times t_i)$ for all i. It follows that none of the arcs A_j intersects any line segment $I(\alpha) \times t_i$. Hence, above $I(\alpha)$, each A_j stays within the the rectangle bounded by the two segments $I(\alpha) \times t_{i_0}$ and $I(\alpha) \times t_{i_0+1}$ and is thus connected to p_{i_0}. In order to determine i_0, we compute the j-th real root γ_j of $f(b, y) \in \mathbb{Q}[y]$ and the largest i_0 such that $\gamma_j > t_{i_0}$. In the special case where $\gamma_j < t_i$ or $\gamma_j > t_i$ for all i, it follows that A_j is connected to $(\alpha, -\infty)$ or $(\alpha, +\infty)$, respectively.

For the arcs located to the left of α, we proceed in exactly the same manner. This concludes the connection phase and, thus, the description of our algorithm.

2.2 Numerical Solver with Certificate

In FastLift, we deploy a certified numerical solver for a fiber polynomial to find regions certified to contain its complex roots. Bini and Fiorentino presented a highly efficient solution to this problem in their MPSolve package [8]. We adapt their approach in a way suited to handle the case where the coefficients are not known a priori, but rather in an intermediate representation which can be refined to any arbitrary finite precision. The description given in this section is high-level. For the details of an efficient implementation, we refer the reader to [25]. In the following considerations, let $g(z) := f(\alpha, z) = \sum_{i=0}^{n} g_i z^i \in \mathbb{R}[z]$ be the fiber polynomial at an x-critical value α and $V(g) = \{\zeta_i\}$, $i = 1, \ldots, n$, its complex roots. Thus, $g(z) = g_n \prod_{i=1}^{n} (z - \zeta_i)$.

The main algorithm used in the numerical solver is the Aberth-Ehrlich iteration for simultaneous root finding. Starting from arbitrary distinct root guesses $(z_i)_{i=1,\ldots,n}$, it is given by the component-wise iteration rule $z_i' = z_i$ if $g(z_i) = 0$, and

$$z_i' = z_i - \frac{g(z_i)/g'(z_i)}{1 - g(z_i)/g'(z_i) \cdot \sum_{j \neq i} \frac{1}{z_i - z_j}}$$

otherwise. As soon as the approximation vector $(z_i)_i$ lies in a sufficiently small neighborhood of some permutation of the actual roots $(\zeta_i)_i$ of g, this iteration converges with cubic order [37]. In practice, Aberth's method shows excellent performance even if started from an arbitrary configuration far away from the solutions.

[5]The threshold for the number of iterations should be chosen based on the degree of f and its coefficient's bitlengths. For the instances considered in our experiments, we stop when reaching 2048 bits of precision.

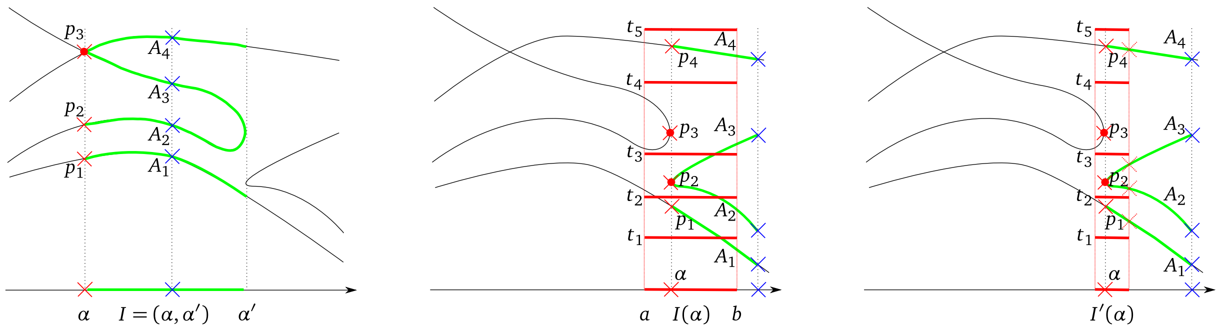


Figure 2.2: The left figure shows the generic case, where exactly one x-critical point (p_3) above α exists. The bottom-up method connects A_1 to p_1 and A_2 to p_2; the remaining arcs have to pass p_3. In the second figure, the fiber at α contains two critical points p_2 and p_3. The red horizontal line segments pass through arbitrary chosen points (α, t_i) separating p_{i-1} and p_i. The initial isolating interval $I(\alpha) = (a, b)$ for α is not sufficient to determine the connections for all arcs since A_1, A_2, A_3 intersect the segments $I \times \{t_i\}$. On the right, the refined isolating interval $I'(\alpha)$ induces boxes $I'(\alpha) \times (t_i, t_{i+1})$ small enough such that no arc crosses the horizontal boundaries. By examination of the y-coordinates of the intersections between the arcs and the fiber over the right-hand boundary of $I'(\alpha)$ (red crosses), we can match arcs and critical points.

A straightforward implementation of [8] expects the coefficients g_i of g to be known up to some relative precision p, that is, the input is a polynomial $\tilde{g} = \sum \tilde{g}_i x^i$ whose floating point coefficients satisfy $|\tilde{g}_i - g_i| \leq 2^{-p} |g_i|$. In particular, this requirement implies that we have to decide in advance whether a coefficient vanishes. In general, though, the critical x-coordinate α of the fiber polynomial, and thus the coefficients of g, are not rational. Thus, it translates to expensive symbolic gcd computations of R and the coefficients of f as a univariate polynomial in $\mathbb{Z}[y][x]$.

Instead, we work on a *Bitstream interval representation* [18, 25] $[g]^\mu$ of g. Its coefficients are interval approximations of the coefficients of g, where we require the width $|g_i^+ - g_i^-|$ of each coefficient $[g]_i^\mu = [g_i^-, g_i^+]$ to be $\leq \mu$ for a certain *absolute* precision $\mu = 2^{-p}$. In this sense $[g]^\mu$ represents the set $\{\tilde{g} : \tilde{g}_i \in [g]_i^\mu\}$ of polynomials in a *μ-polynomial neighborhood* of g; in particular, g itself is contained in $[g]^\mu$. Naturally, for the interval boundaries, we consider dyadic floating point numbers (*bigfloats*). Note that we can easily compute arbitrarily good Bitstream representations of $f(\alpha, z)$ by approximating α to an arbitrary small error, for example using the quadratic interval refinement technique [1].

Starting with some precision (say, $\mu = 2^{-53}$) and a vector of initial approximations, we perform Aberth's iteration on some representative $\tilde{g} \in [g]^\mu$. The natural choice is the *median polynomial* with $\tilde{g}_i = (g_i^- + g_i^+)/2$, but we take the liberty to select other candidates in case of numerical singularities in Aberth's rule (most notably, if $\tilde{g}'(z_i) = 0$ in some iteration).

After a finite number of iterations (depending on the degree of g), we interrupt the iteration and check whether the current approximation state already captures the structure of $V(g)$. We use the following result by Neumaier and Rump [32], founded in the conceptually similar Weierstraß-Durand-Kerner simultaneous root iteration:

LEMMA 2 (NEUMAIER). *Let $g(z) = g_n \prod_{i=1}^n (z - \zeta_i) \in \mathbb{C}[z]$, $g_n \neq 0$. Let $z_i \in \mathbb{C}$ for $i = 1, \ldots, n$ be pairwise distinct root approximations. Then, all roots of g belong to the union $\mathcal{D}$ of the discs*

$$D_i := D(z_i - r_i, |r_i|),$$

$$\textit{where } r_i := \frac{n}{2} \cdot \frac{\omega_i}{g_n} \textit{ and } \omega_i := \frac{g(z_i)}{\prod_{j \neq i}(z_i - z_j)}.$$

Moreover, every connected component C of $\mathcal{D}$ consisting of m discs contains exactly *m zeros of g, counted with multiplicity.*

The above lemma applied to $[g]^\mu$ using conservative interval arithmetic yields a superset $\mathcal{C} = \{C_1, \ldots, C_m\}$ of regions and corresponding multiplicities $\lambda_1, \ldots, \lambda_m$ such that, for each $C_k \in \mathcal{C}$, all polynomials $\tilde{g} \in [g]^\mu$ (and, in particular, g) have exactly λ_k roots in C_k counted with multiplicities. Furthermore, once the quality of the approximations $(z_i)_i$ and $[g]^\mu$ is sufficiently high, $\mathcal{C}$ converges to $V(g)$.

In FASTLIFT, where we aim to isolate the roots of $g := f(\alpha, y)$, we check whether $m = m_\alpha^*$. If the latter equality holds, Teissier's lemma guarantees that the regions $C_k \in \mathcal{C}$ are isolating for the roots of g, and we stop. Otherwise, we repeat Aberth's iteration after checking whether $0 \in [g]^\mu(z_i)$. Informally, if this holds the quality of the root guess is not distinguishable from any (possibly better) guess within the current interval approximation of g, and we double the precision ($\mu' = \mu^2$) for the next stage.

Aberth's iteration lacks a proof for convergence in the general case and, thus, cannot be considered complete. However, we feel this is a merely theoretical issue: to the best of our knowledge, only artificially constructed, highly degenerate configurations of initial approximations render the algorithm to fail. Regardless of this assumption, the regions $C_k \in \mathcal{C}$ are certified to comprise the roots of g at any stage of the algorithm by Neumaier's lemma and the rigorous use of interval arithmetic. Furthermore, since we use the numerical method as a filter only, the overall algorithm is complete.

2.3 Discussion

We conclude our description of the curve analysis algorithm with the following discussion on our method FASTLIFT in the lifting step. FASTLIFT is a certified method, that is, in case of success, it returns the mathematical correct result. However, the reader may notice that FASTLIFT does not apply in all cases, a reason why we additionally consider the complete but less efficient backup method LIFT for some of the x-critical fibers. The failure of FASTLIFT is either due to a *very special geometric situation* along a certain fiber or due to the *behavior of the numerical solver*. Special geometric situations are:

(Geo1) C has a vertical tangent at $x = \alpha$ and $f(\alpha, y)$ is not square-free, or

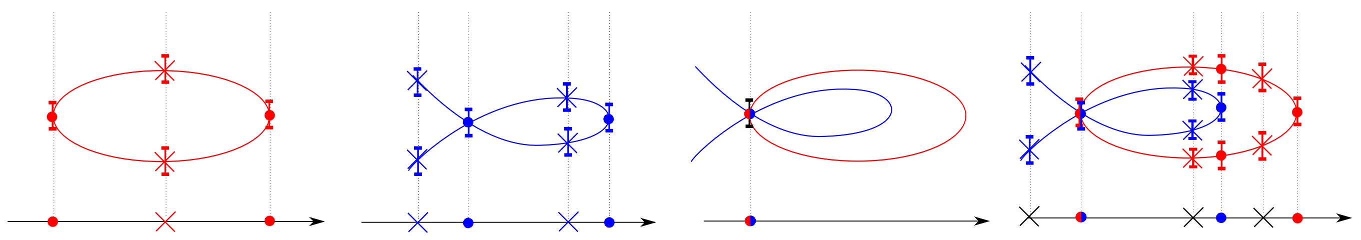


Figure 2.3: The two figures on the left show the topology analyses for the curves $C = V(f)$ and $D = V(g)$. The figure in the middle shows the intersection of the two curves. For the curve pair analysis, critical event lines (at dots) are sorted and non-critical event lines (at crosses) in between are inserted. Finally, for each event line $x = \alpha$, the roots of $f(\alpha, y)$ and $g(\alpha, y)$ are sorted. The latter task is done by further refining corresponding isolating intervals (blue or red intervals) and using the combinatorial information from the curve analyses and the computation of the intersection points.

(Geo2) there exists an intersection point of $f_x^* = \frac{f_x}{\gcd(f_x, f_y)}$ and $f_y^* = \frac{f_y}{\gcd(f_x, f_y)}$ above an x-critical value α of f which is not located on C.

In case that none of the above special geometric situations is given (or removed by applying a shear; see below), the success of FastLift is guaranteed if the following conditions on the numerical solver are fulfilled:

(Num1) The numerical solver is run for sufficiently many iterations, and

(Num2) the approximations returned by the numerical solver converge against the roots of $f(\alpha, y)$ when increasing precision and number of iterations.

As already mentioned in Section 2.2, we consider (Num2) as an exclusively theoretical problem. We further remark that, alternatively, we can use an exact and complete complex bitstream solver [33] to compute arbitrary good approximations of the fiber polynomial $f(\alpha, y)$, an approach which can be considered as an extension of Bdc to complex roots. However, for efficiency reasons, we decided to integrate a numerical method into our implementation instead. (Num1) is a practical problem and, in our implementation, we just empirically set the maximal number of iterations and the maximal precision as parameters depending on the degree and the bitsize of the polynomials. We noticed that the success of FastLift actually depends on these parameter values and consider it an interesting research question how they should be related to the given input to achieve optimal running times.

It is worth mentioning that a complete and exact topology computation can be fully based on FastLift if we allow shearing (i.e., a coordinate transformation $x \mapsto x + sy$ for a $s \in \mathbb{Q}$). Namely, for all but a finite number N of shearing factors,[6] (Geo1) and (Geo2) do not apply to the sheared curve. Hence, when considering $N+1$ distinct shearing factors one by one in circular order and increasing the number of iterations and the precision in the numerical solver for each factor, FastLift eventually succeeds for all fibers.

In practice, as observed in our experiments presented in Section 4, the failure conditions for FastLift are almost negligible, as the method only fails for a few critical fibers on very special instances. For the remaining fibers and for all fibers of the majority of instances, FastLift is successful and extremely fast.

[6]In [24, Corollary 3.2.10], it is shown that there exist at most $n^4 + n$ shearing factors such that the sheared curve has covertical x-critical points. In analogous manner, one can compute an upper bound for the number of shearing factors, where the sheared curve is not in (Geo1) or (Geo2).

3. ARRANGEMENT COMPUTATION

Cgal's recent implementation for computing arrangements of planar algebraic curves reduces all required geometric constructions (as intersections) and predicates (as comparisons of points and x-monotone curves) to the geometric-topological analysis of a single curve [17] and pairs of curves [16]; see also [5] and Cgal's documentation [39].

Beyond the improved curve analysis proposed in Section 2, we aim to avoid subresultant sequences in general when analysing pairs of curves (see illustration in Figure 2.3), which is straightforward given the analyses of each single curve and the common intersection points of the two curves computed by Bisolve:

Let $C = V(f)$ and $D = V(g)$ be two planar algebraic curves implicitly defined by the square-free polynomials f, $g \in \mathbb{Z}[x, y]$. The curve analysis for C provides a set of *critical event lines* $x = \alpha$, where $f(\alpha, y)$ is non square-free. Each α is represented as the root of a square-free polynomial r_i, with r_i a factor of $R_C := (f, f_y, y)$, together with an isolating interval $I(\alpha)$. In addition, we have isolating intervals for the roots of $f(\alpha, y)$. A corresponding result also holds for the curve D with $R_D := \text{res}(g, g_y; y)$. For the common intersection points of C and D, a similar representation is known. That is, we have critical event lines $x = \alpha'$, where α' is a root of a square-free factor of $R_{CD} := \text{res}(f, g; y)$ and, thus, $f(\alpha, y)$ and $g(\alpha, y)$ share at least one common root (or the their leading coefficients both vanish for $x = \alpha$). In addition, isolating intervals for each of these roots are computed. The curve-pair analysis now essentially follows from merging this information. More precisely, we first compute merged *critical event lines* (via sorting the roots of R_C, R_D and R_{CD}) and, then, insert merged *non-critical event lines* at rational values q_I in between. The intersections of C and D with a non-critical event line at $x = q_I$ are easily computed by isolating the roots of $f(q_I, y)$ and $g(q_I, y)$ and further refining the isolating intervals until all isolating intervals are pairwise disjoint. For a critical event line $x = \alpha$, we refine the already computed isolating intervals for $f(\alpha, y)$ and $g(\alpha, y)$ until the number of pairs of overlapping intervals matches the number m of intersection points of C and D above α. This number is obtained from the output of Bisolve applied to f and g, restricted to $x = \alpha$. The information on how to connect the lifted points is provided by the curve analyses for C and D.

We remark that, in the previous approach by Eigenwillig and Kerber [16], m is determined via efficient filter methods, too, while in general, a subresultant computation is needed if the filters fail. This is, for instance, the case when two covertical intersections of C and D occur.

4. IMPLEMENTATION & EXPERIMENTS

Setup.

We have implemented our algorithms as a branch of the bivariate algebraic kernel released with version 3.7 in October 2010, and replaced the curve and curve-pair analyses therein with our new methods based on FASTLIFT, LIFT and BISOLVE. As throughout CGAL, we follow the *generic programming paradigm*, which allows us to choose among various number types and methods to isolated the real roots of integral univariate polynomials. For our setup, we rely on the number types provided by GMP 5.0.1[7] and the highly efficient univariate solver based on the Descartes method contained in RS by Fabrice Rouillier [31],[8] which is also the basis for ISOLATE in Maple 13.

All experiments have been conducted on 2.8 GHz 8-Core Intel Xeon W3530 with 8 MB of L2 cache on a Linux platform. For the GPU-part of the algorithm, we have used the GeForce GTX580 graphics card (Fermi Core). All test data sets are available for download.[9]

Symbolic Speedups.

Our algorithm exclusively relies on two symbolic operations, that is, resultant and gcd computation. We outsource *both* computations to the graphics hardware to reduce the overhead of symbolic arithmetic which typically constitutes the main bottleneck in previous approaches. Besides a quick introduction given next, we refer the interested reader to [21, 19, 20] for more details.

Shortly, both approaches are based on the "divide-conquer-combine" principle used in the modular algorithms by Brown [10] and Collins [14]. This principle allows us to distribute the computation over a large number of processor cores of the graphics card. At highest level, the modular approach can be formulated as follows: 1. Apply modular and/or evaluation homomorphisms to reduce the problem to a large set of subproblems over a simple domain. 2. Solve the subproblems individually in a finite field. 3. Recover the result with polynomial interpolation (in case evaluation homomorphism has been applied) and Chinese remaindering.

Altogether, the GPU realization is quite straightforward and does not deserve much attention within the context of this work. It is only worth noting that our implementation of univariate gcds on the graphics card is comparable in speed with the one from NTL[10] 5.5 running on the host machine. Our intuition for this is because, in contrast to bivariate resultants, computing a gcd of moderate degree univariate polynomials does not provide a sufficient amount of parallelism, and NTL's implementation is nearly optimal. Moreover, the time for initial modular reduction of polynomials, still performed on the CPU, can become noticeably large, thereby neglecting the efficiency of the GPU algorithm. Yet, we find it very promising to perform the modular reduction directly on the GPU which should further speed-up our algorithm.

Contestants.

We compare the new implementation with CGAL's bivariate algebraic kernel (see [5] and [6]) that has shown excellent performance in exhaustive experiments over existing approaches, namely CAD2D[11] and ISOTOP [22] which is based on RS. Both contestants were, except for few example instances, less efficient than CGAL's implementation, so that we omit further tests with them. Two further reasons can be given: Firstly, we enhanced CGAL's kernel with GPU-supported resultants and gcds, which makes it more competitive to existing software, but also to our new software, in case of non-singular curves, though slowdowns are still expected for singular curves or curves in non-generic position due to the need of subresultants sequences performed on the CPU. Secondly, the contestants based on RS require as subtask RS to solve the bivariate polynomial system $f = f_y = 0$ in the curve-analysis. In [4], we learned that the GPU-supported BISOLVE was at least competitive to the current version of RS and even showed in most cases an excellent speed gain over RS. However, RS is currently getting a very promising polish based on Rational Univariate Representations and modular arithmetic [9]. We are looking forward to compare our algorithms with a preliminary version of the new RS quite soon. Moreover, we are confident that the authors will support the computation of arrangements of algebraic curves in a similar setup.

4.1 Curve Analysis

We first present the experiments comparing the analyses of single algebraic curves for different families of curves: (R) random curves of various degree and bit-lengths of their coefficients, (I) curves interpolated through points on a grid, (S) curves in the two-dimensional parameter space of a sphere, (T) curves that were constructed by multiplying a curve $f(x, y)$ with $f(x, y + 1)$, such that each fiber has more than one critical point. (P) projections of intersections of algebraic surfaces in 3D and, finally, (X) "special" curves of degrees up to 42 with many singularities or high-curvature points which we already considered in [4]. The non-random and non-special curves are taken from [24, 4.3]. For the curve topology analysis, we consider four different setups: (a) BISOLVE is, strictly speaking, not comparable with the actual curve analysis as it only computes the solutions of the system $f = f_y = 0$. Still, it is interesting to see that our new hybrid method even outperforms this algorithm that only solves a subproblem of the curve-analysis. (b) LIFTANA exclusively uses LIFT for the fiber computations. (c) FASTANA combines FASTLIFT and LIFT in the fiber computations. More precisely, it uses FASTLIFT first, and if it fails for a certain fiber, LIFT is considered for this fiber instead, (d) AK_2 is the bivariate algebraic kernel shipped with CGAL 3.7, but with GPU-supported resultants and gcds. FASTANA is our default setting, and its running time also includes the timing for the fiber computations where FASTLIFT fails and LIFT is applied instead.

Table 1 lists the running times for single-curve analyses. We only give the results for representative examples. It is easy to see that FASTLIFT is generally superior to the existing

[7]GMP: `http://gmplib.org`

[8]RS: `http://www.loria.fr/equipes/vegas/rs`

[9]`http://www.mpi-inf.mpg.de/departments/d1/projects/Geometry/DataSetsSNC-2011.zip`

[10]NTL, `http://www.shoup.net/ntl`

[11]`http://www.usna.edu/Users/cs/qepcad/B/QEPCAD.html`

Table 1: Running times (in sec) for analyses of algebraic curves of various families; **timeout:** algorithm timed out (> 400 sec)

(R) sets of five random curves				
degree, bits	Bisolve	LiftAna	FastAna	Ak_2
dense curves				
9, 10	0.83	0.73	**0.24**	0.66
9, 2048	4.17	9.00	**2.24**	3.43
15, 10	2.82	3.47	**1.01**	2.21
15, 2048	20.48	50.06	**13.31**	16.82
sparse curves				
9, 10	0.25	0.33	**0.11**	1.00
9, 2048	0.25	0.39	**0.15**	0.98
15, 10	1.50	3.31	**0.57**	3.19
15, 2048	15.66	32.15	**8.08**	24.32
(I) curve interpolated through points on a grid				
degree	Bisolve	LiftAna	FastAna	Ak_2
9	4.51	7.25	**2.50**	4.98
12	25.25	45.58	**14.66**	27.07
(S) parametrized curve on a sphere with 16bit-coefficients				
degree	Bisolve	LiftAna	FastAna	Ak_2
6	6.52	8.17	**2.49**	14.65
9	51.92	78.09	**24.14**	45.66
(T) curves wih a vertically translated copy				
degree	Bisolve	LiftAna	FastAna	Ak_2
6	5.27	3.15	**0.83**	12.89
9	14.78	14.91	**2.97**	159.22
(P) projected intersection curve of surfaces with 8bit-coefficient				
degree(s)	Bisolve	LiftAna	FastAna	Ak_2
$2 \cdot 2$	0.26	0.23	**0.09**	0.22
$3 \cdot 3$	2.87	1.75	**0.52**	1.97
$4 \cdot 4$	10.01	11.93	**2.24**	38.91
$5 \cdot 5$	83.00	95.37	**13.76**	timeout
(X) special curves				
name	Bisolve	LiftAna	FastAna	Ak_2
curve_issac	2.89	2.30	**0.44**	2.96
SA_2_4_eps	0.35	2.18	**0.64**	73.00
grid_deg_10	1.32	2.52	**0.79**	1.54
L6_circles	4.52	92.24	**2.10**	224.19
huge_cusp	8.50	18.40	**5.50**	16.10
swinnerton	19.61	16.81	**7.12**	304.09
degree_7_surf	14.57	48.94	**7.39**	timeout
spider	57.09	195.21	**26.26**	timeout
FTT_5_4_4	11.70	44.27	**32.23**	timeout
challenge_12b	16.48	52.71	**44.30**	timeout
mignotte_xy	260.61	270.70	**136.54**	timeout

kernel, even though Cgal's implementation profits from GPU-accelerated symbolic arithmetic. Moreover, while the speed-up for curves in generic position is already considerable (about half of the time), it becomes even more impressive for projected intersection curves of surfaces and "special" curves with singularities. The reason for this tremendous speed-up is that, for singular curves, Ak_2's performance drops significantly with the degree of the curve when the time to compute subresultants on the CPU becomes dominating. In addition, for curves in non-generic position, the efficiency of Ak_2 is affected because a coordinate transformation has to be considered in these cases.

Recall that the symbolic-numerical filter in FastLift fails for very few instances, in which case Lift is locally used instead. The switch to the backup method is observable in timings; see for instance, *challenge_12b*. As a result, the difference of the running times between Lift and FastLift are considerably less than for instances where the filter method succeeds for all fibers. In these cases, the numerical solver cannot isolate the roots within a given number of iterations, or we indeed have $m < m_\alpha^*$; see Section 2.3. Nevertheless, the running times are still very promising and yet perform better than Ak_2 for non-generic input, even though Lift's implementation is not yet mature enough, and we anticipate a further performance improvement.

Table 2: Running times (in sec) for computing arrangements of algebraic curves; **timeout:** algorithm timed out (> 4000 sec)

(P) increasing number of projected surface intersections		
#resultants	FastKernel	Ak_2
2	**0.21**	0.49
3	**0.48**	0.93
4	**1.03**	1.64
5	**2.44**	3.92
6	**5.14**	7.84
7	**13.65**	21.70
8	**22.69**	35.77
9	**41.53**	67.00
10	**58.37**	91.84
(X) combinations of special curves		
#curves	FastKernel	Ak_2
2	**9.2**	81.93
3	**25.18**	148.46
4	**248.87**	730.57
5	**323.42**	836.43
6	**689.39**	3030.27
7	**757.94**	3313.27
8	**1129.98**	timeout
9	**1166.17**	timeout
10	**1201.34**	timeout
11	**2696.15**	timeout

Similar as Ak_2 has improved on previous approaches when it was presented in 2008, our new methods improve on Ak_2 now. That is, for random, interpolated and parametrized curves, the speed gain is noticeable, while for translated curves and projected intersections, we improve the more the higher the degrees. On special curves of large degree(!), we improve by a factors up to 100 and more.

4.2 Arrangements

For arrangements of algebraic curves, we compare two implementations: (A) Ak_2 is Cgal's bivariate algebraic kernel shipped with Cgal 3.7 but with GPU-supported resultants and gcds. (B) FastKernel is the same, but relies on FastLift-filtered analyses of single algebraic curves. For the curve pair analysis, FastKernel exploits Ak_2's functionality whenever subresultant computations are not needed (i.e., a unique transversal intersection of two curves along a critical event line). For more difficult situations (i.e., two covertical intersections or a tangential intersection), the curve pair analysis uses Bisolve as explained in Section 3. Our testbed consists of sets of curves from different families: (F) random rational functions of various degree (C) random circles (E) random ellipses (R) random curves of various degree and coefficient bit-length (P) sets of projected intersection curves of algebraic surfaces , and, finally, (X) combinations of "special" curves.

We skip the tables for rational functions, circles, ellipses and random curves because the performance of both contestants are more or less equal: The *linearly* many curve-analyses are simple and, for the *quadratic* number of curve-pair analyses, there are typically no multiple intersections along a fiber, that is, BISOLVE is not triggered. Thus, the execution paths of both implementations are almost identical, but only as we enhanced AK_2 with GPU-enabled resultants and gcds. In addition, we also do not expect the need of a shear for such curves, thus, the behavior is anticipated. The picture changes for projected intersection curves of surfaces and combinations of special curves whose running times are reported in Table 2. The AK_2 requires for both sets expensive subresultants to analyze single curves and to compute covertical intersections, while FASTKERNEL's performance is crucially less affected in such situations.

Acknowledgments

Without Michael Kerber's careful implementation of the bivariate kernel in CGAL, this work would not have been realizable in a reasonable time. We would like to use the opportunity to thank Michael for his excellent work.

5. REFERENCES

[1] J. Abbott. Quadratic interval refinement for real roots, 2006. `www.dima.unige.it/~abbott/publications/RefineInterval.pdf`.

[2] L. Alberti, B. Mourrain, and J. Wintz. Topology and Arrangement Computation of Semi-Algebraic Planar Curves. *CAGD*, 25(8):631–651, 2008.

[3] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry*, volume 10 of *Algorithms and Computation in Mathematics*. Springer, 2006.

[4] E. Berberich, P. Emeliyanenko, and M. Sagraloff. An elimination method for solving bivariate polynomial systems: Eliminating the usual drawbacks. In *ALENEX '11*, pages 35–47, San Francisco, USA, 2011. SIAM.

[5] E. Berberich, M. Hemmer, and M. Kerber. A generic algebraic kernel for non-linear geometric applications. In *SoCG '11*, Paris, France, June 2011. To appear.

[6] E. Berberich, M. Hemmer, S. Lazard, L. Peñaranda, and M. Teillaud. Algebraic kernel. In *CGAL User and Reference Manual 3.7*. CGAL Editorial Board, 2010.

[7] E. Berberich, M. Kerber, and M. Sagraloff. An efficient algorithm for the stratification and triangulation of algebraic surfaces. *CGTA*, 43:257–278, 2010.

[8] D. A. Bini and G. Fiorentino. Design, analysis, and implementation of a multiprecision polynomial rootfinder. *Numerical Algorithms*, 23:127–173, 2000.

[9] Y. Bouzidi, S. Lazard, M. Pouget, and F. Rouillier. New bivariate system solver and topology of algebraic curves. In *EuroCG '11*, Nancy, France, March 2011.

[10] W. S. Brown. On Euclid's algorithm and the computation of polynomial greatest common divisors. In *SYMSAC '71*, pages 195–211, New York, NY, USA, 1971. ACM.

[11] M. Burr, S. W. Choi, B. Galehouse, and C. K. Yap. Complete subdivision algorithms, II: Isotopic Meshing of Singular Algebraic Curves. In *ISSAC '08*, pages 87–94. ACM, 2008.

[12] J. Cheng, S. Lazard, L. Peñaranda, M. Pouget, F. Rouillier, and E. Tsigaridas. On the topology of real algebraic plane curves. *MCS (special issue on Comp. Geom. and CAGD)*, 4(1):113–137, 2010.

[13] J.-S. Cheng, X.-S. Gao, and J. Li. Root isolation for bivariate polynomial systems with local generic position method. In *ISSAC '09*, pages 103–110, New York, NY, USA, 2009. ACM.

[14] G. E. Collins. The calculation of multivariate polynomial resultants. In *SYMSAC '71*, pages 212–222. ACM, 1971.

[15] G. E. Collins and A. G. Akritas. Polynomial real root isolation using Descarte's rule of signs. In *SYMSAC '76*, pages 272–275, New York, NY, USA, 1976. ACM.

[16] A. Eigenwillig and M. Kerber. Exact and Efficient 2D-Arrangements of Arbitrary Algebraic Curves. In *SoDA '08*, pages 122–131. ACM & SIAM, 2008.

[17] A. Eigenwillig, M. Kerber, and N. Wolpert. Fast and Exact Geometric Analysis of Real Algebraic Plane Curves. In *ISSAC '07*, pages 151–158. ACM, 2007.

[18] A. Eigenwillig, L. Kettner, W. Krandick, K. Mehlhorn, S. Schmitt, and N. Wolpert. A Descartes algorithm for polynomials with bit-stream coefficients. In *CASC '05*, volume 3718 of *LNCS*, pages 138–149, 2005.

[19] P. Emeliyanenko. A complete modular resultant algorithm targeted for realization on graphics hardware. In *PASCO*, pages 35–43, New York, USA, 2010. ACM.

[20] P. Emeliyanenko. Modular Resultant Algorithm for Graphics Processors. In *ICA3PP '10*, pages 427–440, Berlin, Heidelberg, 2010. Springer-Verlag.

[21] P. Emeliyanenko. High-performance polynomial GCD computations on graphics processors. In *HPCS '11*, Istanbul, Turkey, July 2011. To appear.

[22] L. Gonzalez-Vega and I. Necula. Efficient Topology Determination of Implicitly Defined Algebraic Plane Curves. *CAGD '02*, 19(9):719–743, 2002.

[23] J. Gwozdziewicz, A. Ploski, and J. G. Zdziewicz. Formulae for the singularities at infinity of plane algebraic curves, 2000.

[24] M. Kerber. *Geometric Algorithms for Algebraic Curves and Surfaces.* PhD thesis, Saarland University, Saarbrücken, Germany, 2009.

[25] A. Kobel. Certified Numerical Root Finding. Master's thesis, Universität des Saarlandes, Saarbrücken, Germany, 2011.

[26] O. Labs. A list of challenges for real algebraic plane curve visualization software. In *Nonlinear Computational Geometry*, volume 151 of *The IMA Volumes*, pages 137–164. Springer New York, 2010.

[27] Y. Lu, D. Bates, A. Sommese, and C. Wampler. Finding all real points of a complex curve. *Contemporary Mathematics*, (448):183–206, 2007.

[28] B. Mourrain and J.-P. Pavone. Subdivision methods for solving polynomial equations. Technical report, INRIA, Sophia Antipolis, France, 2005.

[29] L. Peñaranda. *Non-linear computational geometry for planar algebraic curves.* PhD thesis, Uni. Nancy, 2010.

[30] S. Plantinga and G. Vegter. Isotopic approximation of implicit curves and surfaces. In *Symp. on Geometry Processing*, pages 251–260, 2004.

[31] F. Rouillier and P. Zimmermann. Efficient isolation of polynomial's real roots. *J. Comput. Appl. Math.*, 162(1):33–50, 2004.

[32] S. M. Rump. Ten methods to bound multiple roots of polynomials. *J. Comp. Appl. Math.*, 156:403–432, 2003.

[33] M. Sagraloff. A general approach to isolating roots of a bitstream polynomial. *Mathematics in Computer Science*, 2011. Accepted for publication, see also `http://www.mpi-inf.mpg.de/~msagralo/bit11.pdf`.

[34] J. M. Snyder. Interval analysis for computer graphics. In *SIGGRAPH*, pages 121–130, 1992.

[35] J. M. Snyder and J. T. Kajiya. Generative modeling: a symbolic system for geometric modeling. In *SIGGRAPH*, pages 369–378, 1992.

[36] B. Teissier. Cycles évanescents, sections planes et conditions de whitney. (french). *Singularités à Cargèse*, (78):285–362, 1973.

[37] P. Tilli. Convergence conditions of some methods for the simultaneous computation of polynomial zeros. *Calcolo*, 35:3–15, 1998.

[38] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 2003.

[39] R. Wein, E. Fogel, B. Zukerman, and D. Halperin. 2D arrangements. In *CGAL User and Reference Manual 3.7*. CGAL Editorial Board, 2010.

APPENDIX

A. FURTHER EXPERIMENTS OF CURVE ANALYSES

Table 3: Running times (in sec) for analyses of random algebraic curves; **timeout:** algorithm timed out (> 400 sec)

(R) sets of five random curves				
degree, bits	BISOLVE	LIFTANA	FASTANA	AK_2
dense curves				
6, 10	0.38	0.39	**0.15**	0.36
6, 128	0.32	0.49	**0.18**	0.32
6, 512	0.55	0.89	**0.35**	0.55
6, 2048	1.85	3.50	**0.99**	1.71
9, 10	0.83	0.73	**0.24**	0.66
9, 128	0.57	0.90	**0.31**	0.55
9, 512	1.04	1.91	**0.57**	0.96
9, 2048	4.17	9.00	**2.24**	3.43
12, 10	2.18	2.21	**0.69**	1.77
12, 128	1.64	2.87	**0.86**	1.46
12, 512	2.84	6.09	**1.56**	2.55
12, 2048	12.07	30.26	**7.06**	9.96
15, 10	2.82	3.47	**1.01**	2.21
15, 128	2.36	4.55	**1.29**	2.06
15, 512	4.40	9.84	**2.63**	3.77
15, 2048	20.48	50.06	**13.31**	16.82
sparse curves				
6, 10	0.17	0.21	**0.12**	0.22
6, 128	0.14	0.19	**0.09**	0.22
6, 512	0.21	0.36	**0.12**	0.39
6, 2048	0.73	1.13	**0.39**	1.06
9, 10	0.25	0.33	**0.11**	1.00
9, 128	0.25	0.39	**0.15**	0.98
9, 512	0.43	0.65	**0.23**	1.43
9, 2048	1.53	2.51	**0.85**	4.31
12, 10	0.41	0.83	**0.19**	1.67
12, 128	0.40	1.05	**0.25**	1.60
12, 512	0.78	2.16	**0.50**	2.53
12, 2048	3.82	10.54	**2.82**	9.94
15, 10	1.50	3.31	**0.57**	3.19
15, 128	1.48	3.98	**0.72**	2.99
15, 512	2.96	7.10	**1.45**	5.58
15, 2048	15.66	32.15	**8.08**	24.32

Table 4: Running times (in sec) for analyses of algebraic curves of various families; **timeout:** algorithm timed out (> 400 sec)

(I) curve interpolated through points on a grid				
degree	Bisolve	LiftAna	FastAna	Ak_2
5	0.41	0.50	**0.21**	0.49
6	0.72	1.13	**0.41**	0.84
7	1.42	2.20	**0.78**	1.62
8	2.59	3.78	**1.32**	2.86
9	4.51	7.25	**2.50**	4.98
10	6.62	11.52	**3.62**	7.65
11	12.38	23.27	**7.25**	13.93
12	25.25	45.58	**14.66**	27.07
13	48.97	93.97	**27.90**	46.21
14	101.96	193.61	**59.14**	90.27
15	211.95	timeout	**114.68**	166.68
16	timeout	timeout	**236.39**	314.61
(S) parametrized curve on a sphere with 16bit-coefficients				
degree	Bisolve	LiftAna	FastAna	Ak_2
6	6.52	8.17	**2.49**	14.65
7	22.58	27.39	**8.47**	16.83
8	37.94	53.3	**16.25**	32.09
9	51.92	78.09	**24.14**	45.66
10	72.21	110.81	**32.38**	64.31
(T) curves with a vertically translated copy				
degree	Bisolve	LiftAna	FastAna	Ak_2
5	3.95	1.93	**0.6**	5.61
6	5.27	3.15	**0.83**	12.89
7	7.31	6.14	**1.28**	33.97
8	9.66	7.99	**1.75**	73.59
9	14.78	14.91	**2.97**	159.22
10	17.03	15.78	**3.36**	timeout
(P) projected intersection curve of surfaces with 8bit-coefficient				
degree(s)	Bisolve	LiftAna	FastAna	Ak_2
$2 \cdot 2$	0.26	0.23	**0.09**	0.22
$2 \cdot 3$	0.39	0.40	**0.12**	0.32
$2 \cdot 4$	1.03	0.77	**0.25**	0.73
$2 \cdot 5$	1.97	1.53	**0.44**	1.93
$3 \cdot 3$	2.87	1.75	**0.52**	1.97
$3 \cdot 4$	3.63	3.02	**0.67**	5.21
$3 \cdot 5$	8.71	9.22	**1.90**	25.87
$4 \cdot 4$	10.01	11.93	**2.24**	38.91
$4 \cdot 5$	21.45	28.22	**4.46**	231.64
$5 \cdot 5$	83.00	95.37	**13.76**	timout
(X) special curves (see Table 5, Appendix B for desciptions)				
name	Bisolve	LiftAna	FastAna	Ak_2
curve_issac	2.89	2.30	**0.44**	2.96
L4_circles	1.56	8.80	**0.52**	9.92
SA_2_4_eps	0.35	2.18	**0.64**	73.00
grid_deg_10	1.32	2.52	**0.79**	1.54
ten_circles	6.77	3.79	**0.82**	24.07
dfold_10_6	4.45	4.62	**1.58**	30.80
L6_circles	4.52	92.24	**2.10**	224.19
cov_sol_20	8.68	12.16	**3.38**	65.05
curve24	14.07	15.61	**4.07**	50.73
huge_cusp	8.50	18.40	**5.50**	16.10
swinnerton	19.61	16.81	**7.12**	304.09
degree_7_surf	14.57	48.94	**7.39**	timeout
challenge_12a	7.47	15.84	**14.41**	timeout
spider	57.09	195.21	**26.26**	timeout
FTT_5_4_4	11.70	44.27	**32.23**	timeout
challenge_12b	16.48	52.71	**44.30**	timeout
mignotte_xy	260.61	270.70	**136.54**	timeout

B. DESCRIPTION OF SPECIAL CURVES

Table 5: Description of the special curves used in the experiments of Section 4. Sources of curves are given where known.

Instance	y-degree	Description
curve_issac	15	isolated points, high-curvature points [13]
L4_circles	16	4 circles w.r.t. L4-norm; clustered solutions
SA_2_4_eps	16	singular points with high tangencies, displaced [26]
grid_deg_10	10	large coefficients; curve in generic position
ten_circles	20	set of 10 random circles multiplied together; rational solutions
dfold_10_6	30	many half-branches [26]
L6_circles	32	4 circles w.r.t. L6-norm; clustered solutions
cov_sol_20	20	covertical solutions
curve24	24	curvature of degree 8 curve; many singularities
huge_cusp	8	large coefficients; high-curvature points
swinnerton	25	covertical solutions in x and y
degree_7_surf	42	silhouette of an algebraic surface; covertical solutions in x and y
challenge_12a	30	many candidate solutions to be checked [26]
spider	12	degenerate curve; many clustered solutions
FTT_5_4_4	40	many non-rational singularities [26]
challenge_12b	40	many candidates to check [26]
mignotte_xy	42	a product of x/y-Mignotte polynomials, displaced; many clustered solutions

A subresultant based subspace method for the computation of polynomial GCDs (extended abstract)[*]

Bingyu Li
School of Mathematics and Statistics
Northeast Normal University
Changchun, China
mathliby@gmail.com

Xiaoli Wu
School of Science
Hangzhou Dianzi University
Hangzhou, China
xlwu@amss.ac.cn

ABSTRACT

In [3], a generalized Sylvester matrix based subspace method for computing the greatest common divisor (GCD) of a set of univariate polynomials was derived. In this paper, we consider the problem of computing the GCD of two polynomials and establish a new Sylvester subresultant matrix based subspace method. Compared with the method proposed in [3], our method has a lower computational cost.

Categories and Subject Descriptors

I.1.2 [**Computing Methodologies**]: Symbolic and Algebraic Manipulation-Algorithms

General Terms

Algorithms, Experimentation

Keywords

GCD, subspace method, subresultant matrix

1. SYLVESTER SUBRESULTANT MATRIX-BASED SUBSPACE METHOD

Given two univariate polynomials:

$$Y_i(x) = y_i(0) + y_i(1)x + \ldots + y_i(d_i)x^{d_i},\ i = 1, 2. \tag{1}$$

Assume that the GCD of $Y_1(x)$ and $Y_2(x)$ is $C(x)$ with degree N, i.e.,

$$Y_i(x) = C(x)H_i(x),\ i = 1, 2, \tag{2}$$

where $H_i(x)$ is with degree $L_i = d_i - N, i = 1, 2$, and $H_1(x)$ and $H_2(x)$ are coprime. Denote

$$C(x) = c(0) + c(1)x + \ldots + c(N)x^N,$$
$$H_i(x) = h_i(0) + h_i(1)x + \ldots + h_i(L_i)x^{L_i},\ i = 1, 2.$$

[*]Xiaoli Wu is supported by the Hangzhou Dianzi University Startup Foundation under Grant KYS075610053.

Given a vector $\mathbf{u} = [u(k), \ldots, u(1), u(0)]^T \in \mathbb{R}^{k+1}$ and a positive integer l, we define

$$\mathcal{C}(\mathbf{u}, k, l) = \begin{bmatrix} u(k) & \cdots & u(0) & & \\ & \ddots & & \ddots & \\ & & u(k) & \cdots & u(0) \end{bmatrix}_{l\times(l+k)}.$$

Then we can rewrite (2) in the matrix form:

$$S_N = CH^T, \tag{3}$$

where

$$\begin{aligned} S_N &= \left[\mathcal{C}(\mathbf{y}_1, d_1, L_2+1)^T\ \mathcal{C}(\mathbf{y}_2, d_2, L_1+1)^T\right], \\ C &= \mathcal{C}(\mathbf{c}, N, L_1+L_2+1)^T, \\ H^T &= \left[\mathcal{C}(\mathbf{h}_1, L_1, L_2+1)^T\ \ \mathcal{C}(\mathbf{h}_2, L_2, L_1+1)^T\right], \\ \mathbf{y}_i &= [y_i(d_i), \ldots,\ y_i(1),\ y_i(0)]^T,\ i = 1, 2, \\ \mathbf{c} &= [c(N), \ldots,\ c(1),\ c(0)]^T, \\ \mathbf{h}_i &= [h_i(L_i), \ldots,\ h_i(1),\ h_i(0)]^T,\ i = 1, 2. \end{aligned}$$

S_N is called an N-th Sylvester matrix, or a subresultant matrix, it relates GCDs deeply. See, for example [1, 2].

THEOREM 1. *With the GCD assumption (2), H^T has full row rank and S_N has column rank of deficiency one.*

Next we describe our subspace method:
Case I: $d_1 + d_2 \geq 3N$
Let $r = L_1 + L_2 + 1$, $S_N = U_{r+1}\Sigma V^T$ be the thin SVD, where Σ is a diagonal matrix consisting of singular values of S_N, $U_{r+1} \in \mathbb{R}^{(d_1+d_2-N+1)\times(r+1)}$, $V \in \mathbb{R}^{(r+1)\times(r+1)}$, and $U_{r+1}^T U_{r+1} = I_{r+1}$, $V^T V = I_{r+1}$. Partition U_{r+1} as $U_{r+1} = [U_r\, \mathbf{u}_{r+1}]$, where $\mathbf{u}_{r+1}$ is the last column of U_{r+1}. Then $\mathcal{R}(U_r) = \mathcal{R}(C)$, where $\mathcal{R}(\cdot)$ denotes the range space of a matrix. It follows that

$$\mathbf{u}_{r+1}^T C = 0. \tag{4}$$

Equivalently, it holds that

$$Q_{r+1}\mathbf{c} = 0, \tag{5}$$

where

$$\begin{aligned} Q_{r+1} &= \begin{bmatrix} \mathbf{u}_{r+1}[1] & \mathbf{u}_{r+1}[2] & \cdots & \mathbf{u}_{r+1}[N+1] \\ \mathbf{u}_{r+1}[2] & \mathbf{u}_{r+1}[3] & \cdots & \mathbf{u}_{r+1}[N+2] \\ \vdots & \vdots & \cdots & \vdots \\ \mathbf{u}_{r+1}[r] & \mathbf{u}_{r+1}[r+1] & \cdots & \mathbf{u}_{r+1}[d_1+L_2+1] \end{bmatrix}, \\ &\in \mathbb{R}^{(d_1+d_2-2N+1)\times(N+1)} \\ \mathbf{c} &= [c(N), \ldots, c(1), c(0)]^T. \end{aligned}$$

With the assumption $d_1 + d_2 \geq 3N$, the row dimension of Q_{r+1} is equal to or greater than the column dimension of Q_{r+1}. In this case, we extract $\mathbf{c}$ from the unique (up to a constant scalar) minimal right singular vector of Q_{r+1}.

Case II: $d_1 + d_2 < 3N$

Now we let $S_N = [U_{r+1}, U_0] \begin{bmatrix} \Sigma \\ 0 \end{bmatrix} V^T$ be the full SVD, where U_{r+1}, Σ and V are defined as before, $[U_{r+1}, U_0]$ is an orthogonal matrix of order $d_1 + d_2 - N + 1$. As before, we partition U_{r+1} as $U_{r+1} = [U_r\, \mathbf{u}_{r+1}]$. Then both (4) and (5) still hold. However, with the assumption $d_1+d_2 < 3N$, Q_{r+1} has more columns than rows and the dimension of its null space is larger than one in general. In this case, we single out the desired null vector $\mathbf{c}$ from the null space as follows.

Partition U_0 as $U_0 = [\mathbf{u}_{r+2}, \ldots, \mathbf{u}_{d_1+d_2-N+1}]$. Note that $d_1 + d_2 - N + 1 = r + N$ and that $\mathbf{u}_{r+i}^T C = 0$, $i = 2, \ldots, N$. Starting with $i = 2$, we use $\mathbf{u}_{r+i}$ to construct Q_{r+i} such that it has the same structure as that of Q_{r+1}, and stack the matrices $Q_{r+1}, \ldots, Q_{r+i}$ to get $Q = \left[Q_{r+1}^T, \ldots, Q_{r+i}^T\right]^T \in \mathbb{R}^{i(d_1+d_2-2N+1)\times(N+1)}$, until

$$i(d_1 + d_2 - 2N + 1) \geq N + 1. \tag{6}$$

Assume that when $i = i_0$, (6) is achieved. Then we calculate the second smallest singular value of Q. If this singular value is larger than a given tolerance, then it confirms that the nullspace of Q is of dimension one, we then extract $\mathbf{c}$ from the minimal right singular vector of Q; Otherwise, we continue to construct Q_{r+i_0+1} and stack it to get a new $Q := \begin{bmatrix} Q \\ Q_{r+i_0+1} \end{bmatrix}$, until we find a Q which has a nullspace of dimension one.

Remark. We see that, in [3], $\mathbf{c}$ is essentially extracted from the minimal right singular vector of the matrix

$$\left[\mathbf{U}_{2d+2-N}^T, \ldots, \mathbf{U}_{2d+1}^T\right]^T \in \mathbb{R}^{N(2d-N+1)\times(N+1)}. \tag{7}$$

See the definitions of $\mathbf{U}_{2d+2-N}, \ldots, \mathbf{U}_{2d+1}$ in **Appendix**.

In contrast, for our method, in **Case I**, Q_{r+1} is

$$(d_1 + d_2 - 2N + 1) \times (N + 1);$$

in **Case II**, Q is formed by stacking $Q_{r+1}, \ldots, Q_{r+i}$ and is

$$i(d_1 + d_2 - 2N + 1) \times (N + 1), \text{ where } i \leq N.$$

In the numerical tests, i equals i_0, the smallest integer satisfying (6). Thus, compared with the method in [3], the matrix eigenvalue problems to be resolved in our method have smaller sizes.

2. NUMERICAL EXPERIMENTS

We test the performance of our derived method in Maple 11 and set Digits= 16. For **Cases I** and **II**, we conduct 100 tests, respectively. At each test, the tested polynomials $Y_i(x), i = 1, 2$ are constructed by (2), where $C(x)$ and $H_i(x), i = 1, 2$ are randomly generated polynomials with given degrees $N, L_i, i = 1, 2$, respectively. Specifically, the GCD vector $\mathbf{c}$ and cofactors vectors $\mathbf{h}_i, i = 1, 2$ are randomly generated from the standard Gaussian distribution $\mathcal{N}(0, 1)$. In the two cases, we apply our method to compute the GCD of $Y_i(x), i = 1, 2$. Then we test the error $\frac{\|\hat{\mathbf{c}}-\mathbf{c}\|}{\|\mathbf{c}\|}$, where $\hat{\mathbf{c}}$ denotes the computed coefficient vector of the GCD. In the table, the reported error is the average error of 100 results in each case. In **Case II**, at each test we stop stacking $\{Q_{r+i}\}$ at $i = i_0$ and form Q. The average of the second smallest singular values of Q is about 0.085.

For each set of polynomials $Y_i, i = 1, 2$, we also apply the method in [3] to calculate the GCD vector $\hat{\hat{\mathbf{c}}}$ and test the error $\frac{\|\hat{\hat{\mathbf{c}}}-\mathbf{c}\|}{\|\mathbf{c}\|}$. In most of tests, $\frac{\|\hat{\hat{\mathbf{c}}}-\mathbf{c}\|}{\|\mathbf{c}\|}$ and $\frac{\|\hat{\mathbf{c}}-\mathbf{c}\|}{\|\mathbf{c}\|}$ have similar magnitudes; however, there are two tests where $\hat{\hat{\mathbf{c}}}$ have large errors 1.3 10^{-3} and 4.8 10^{-6}, respectively. We mention that, the matrices (7) in **Cases I, II** are of 510×11 and 820×21, respectively, and they have much larger sizes than those of Q_{r+1} and Q, as observed in the table.

Case I.			**Case II**.		
N, L_1, L_2	$\frac{\|\hat{\mathbf{c}}-\mathbf{c}\|}{\|\mathbf{c}\|}$	size of Q_{r+1}	N, L_1, L_2	$\frac{\|\hat{\mathbf{c}}-\mathbf{c}\|}{\|\mathbf{c}\|}$	size of Q
$10, 15, 20$	$2.7\ 10^{-11}$	36×11	$20, 5, 10$	$3.3\ 10^{-13}$	32×21

3. ACKNOWLEDGEMENTS

We thank the anonymous referees for their helpful suggestions and comments. Comparisons with the method in [3] on the computation of approximate GCDs of perturbed polynomials with not small perturbations and bad conditioning have been done. We are unable to understand the numerical phenomenons well, and so we do not summarize the results here. Further theoretical considerations and more thorough comparisons with the method in [3] and other existing methods in the literature will be addressed in our full version paper.

4. REFERENCES

[1] I. Z. Emiris, A. Galligo, and H. Lombardi. Certified approximate univariate GCDs. *Journal of Pure and Applied Algebra*, 117-118:229–251, 1997.

[2] E. Kaltofen, Z. Yang, and L. Zhi. Structured low rank approximation of a sylvester matrix. In D. Wang and L. Zhi, editors, *Symbolic-Numeric Computation*. Birkhäuser Verlag, Basel, Switzerland, 2007, 69–83.

[3] W. Qiu, Y. Hua, and K. A. Meraim. A subspace method for the computation of the GCD of polynomials. *Automatica*, 33:741–743, 1997.

APPENDIX

We summarize the method in [3] as follows. Denote $d = \max(d_1, d_2)$ and define

$$\bar{\mathbf{y}}_i = [y_i(0), y_i(1), \ldots, y_i(d_i), 0, \ldots, 0]^T \in \mathbb{R}^{d+1},\ i = 1, 2.$$

Step 1. Form the matrix $\mathbf{Y} = \begin{bmatrix} \mathcal{C}(\bar{\mathbf{y}}_1, d, d+1) \\ \mathcal{C}(\bar{\mathbf{y}}_2, d, d+1) \end{bmatrix}$.

Step 2. Calculate the SVD: $\mathbf{Y}^T = [\mathbf{U}_t\, \mathbf{U}_0] \begin{bmatrix} \mathbf{\Sigma}_t & \\ & 0 \end{bmatrix} [\mathbf{V}_t\, \mathbf{V}_0]^T$, where $t = \operatorname{rank}(\mathbf{Y}) = 2d + 1 - N$, $\mathbf{\Sigma}_t$ contains t non-zero singular values, $\mathbf{U}_t$ and $\mathbf{U}_0 = [\mathbf{u}_{t+1}, \ldots, \mathbf{u}_{2d+1}]$ contain the left principal and left null singular vectors, respectively.

Step 3. Form $\mathbf{R} = \sum_{i=t+1}^{2d+1} \mathbf{U}_i^T \mathbf{U}_i$, where

$$\mathbf{U}_i = \begin{bmatrix} \mathbf{u}_i[1] & \mathbf{u}_i[2] & \cdots & \mathbf{u}_i[N+1] \\ \mathbf{u}_i[2] & \mathbf{u}_i[3] & \cdots & \mathbf{u}_i[N+2] \\ \cdots & \cdots & \cdots & \cdots \\ \mathbf{u}_i[t] & \mathbf{u}_i[t+1] & \cdots & \mathbf{u}_i[2d+1] \end{bmatrix}.$$

Step 4. Extract $\mathbf{c}$ by the eigenvector of $\mathbf{R}$ corresponding to the smallest eigenvalue.

An Improved Method for Evaluating Max Noether Conditions: Case of Breadth One

[Extended Abstract]

Nan Li
Key Laboratory of Mathematics Mechanization, AMSS, Beijing 100190, China
INRIA, Paris-Rocquencourt, 78513 Le Chesnay, France
linan08@amss.ac.cn

For an isolated breadth-one singular solution $\hat{\mathbf{x}}_e$ of a polynomial system $F = \{f_1, \ldots, f_n\}$, $f_i \in \mathbb{C}[x_1, \ldots, x_n]$ (breadth one means the Jacobian $J_F(\hat{\mathbf{x}}_e)$ is of corank one), in [6, 7] we present a symbolic-numeric method to refine an approximate solution $\hat{\mathbf{x}}$ with quadratic convergence if $\hat{\mathbf{x}}$ is close to $\hat{\mathbf{x}}_e$. A preliminary implementation performs well in case the approximate Max Noether conditions computed are sparse, but suffers from the evaluation of them when they are not. In this paper we describe how to avoid the linear transformation and evaluate Max Noether conditions efficiently by solving a sequence of least squares problems.

Categories and Subject Descriptors

I.2.1 [**Computing Methodologies**]: Symbolic and Algebraic Manipulation—*Algorithms*; G.1.5 [**Mathematics of Computing**]: Numerical Analysis—*Roots of Nonlinear Equations*

General Terms

Algorithms, Theory, Experimentation

Keywords

root refinement, isolated singular solution, polynomial system, evaluation, Max Noether condition

1. COMPUTING MAX NOETHER CONDITIONS WITHOUT THE LINEAR TRANSFORMATION

In [6], we compute a non-trivial null vector

$$\mathbf{a}_1 = [a_{11}, \ldots, a_{1,n}]$$

of $J_F(\hat{\mathbf{x}}_e)$, then form a regular matrix R which has $\mathbf{a}_1$ as its first column and perform the linear transformation $H(\mathbf{z}) = F(R\mathbf{z})$. The purpose is to ensure that the new system $H(\mathbf{z})$ satisfies

$$\frac{\partial H(\hat{\mathbf{z}}_e)}{\partial z_1} = \mathbf{0},$$

where $\hat{\mathbf{z}}_e = R^{-1}\hat{\mathbf{x}}_e$. Therefore, one can construct the k-th order Max Noether condition incrementally for k from 2 to $\mu - 1$ by formulas in [6, Theorem 3.1], where μ is the multiplicity of $\hat{\mathbf{z}}_e$.

In order to simplify discussions, without loss of generality, we assume

$$|a_{1,1}| \geq |a_{1,j}|, \text{ for } 1 \leq j \leq n. \tag{1}$$

Otherwise, one can perform changes of variables to guarantee that (1) is satisfied. After normalizing $\mathbf{a}_1$ by $a_{1,1}$, we modify Theorem 3.1 in [6] as follows.

Theorem 1. *Suppose* $L_1 = \frac{\partial}{\partial x_1} + a_{1,2}\frac{\partial}{\partial x_2} + \cdots + a_{1,n}\frac{\partial}{\partial x_n}$, *the k-th Max Noether condition of F at $\hat{\mathbf{x}}_e$ can be constructed as*

$$L_k = P_k + a_{k,2}\frac{\partial}{\partial x_2} + \cdots + a_{k,n}\frac{\partial}{\partial x_n}, \tag{2}$$

where

$$P_k = \frac{1}{k}\left[\frac{\partial}{\partial x_1}L_{k-1} + \sum_{j=2}^{n}\frac{\partial}{\partial x_j}\left(a_{1,j}L_{k-1} + \cdots + (k-1)a_{k-1,j}L_1\right)\right]. \tag{3}$$

The parameters $a_{k,j}$, $j = 2, \ldots, n$ are determined by solving

$$\left[\frac{\partial F(\hat{\mathbf{x}}_e)}{\partial x_2}, \ldots, \frac{\partial F(\hat{\mathbf{x}}_e)}{\partial x_n}\right] \cdot \begin{bmatrix} a_{k,2} \\ \vdots \\ a_{k,n} \end{bmatrix} = -\begin{bmatrix} P_k(f_1)_{\mathbf{x}=\hat{\mathbf{x}}_e} \\ \vdots \\ P_k(f_n)_{\mathbf{x}=\hat{\mathbf{x}}_e} \end{bmatrix}. \tag{4}$$

When the linear system (4) has no solutions, we stop the computation and set $\mu = k$ as the multiplicity of $\hat{\mathbf{x}}_e$.

Theorem 1 can be used to compute the approximate Max Noether conditions of F at an approximate solution $\hat{\mathbf{x}}$. However, a threshold has to be selected properly to determine that the linear system (4) has no solution numerically.

2. EVALUATION OF MAX NOETHER CONDITIONS

According Theorem 1, in order to compute the k-th Max Noether condition, we need to evaluate $P_k(F)$ at $\hat{\mathbf{x}}_e$. To do this, in [6], we first compute the differential operator P_k, then obtain $P_k(F)$ and evaluate it at $\hat{\mathbf{x}}_e$. Even if F is sparse, P_k could still be very dense. Hence, evaluation of the vector on the right side of (4) could be very expensive sometimes.

Example 1. *Consider a system* $F = \{f_1, \dots, f_s\}$

$$\begin{aligned} f_i &= x_i^3 + x_i^2 - x_{i+1}, \text{ if } i < s, \\ f_s &= x_s^2 \end{aligned}$$

with zero $(0, \dots, 0)$ *of the multiplicity* 2^s.

As shown in [6], for $s = 6$, about 17MB of memory is used to store the Max Noether conditions and it takes about 3 hours to compute all of them. Moreover, for $s = 7$, we are not able to obtain all Max Noether conditions in 2 days. It is not a surprise that the computation is dominated by the evaluation of $P_k(F)$ at $\hat{\mathbf{x}}_e$ in (4). This motivates us to design a new method to remedy this drawback.

In fact, we can take advantage of (3) to construct $P_k(F)$ and $L_k(F)$ by

$$P_k(F) = \sum_{j=1}^{k-1} \frac{j}{k} \cdot J_{L_{k-j}(F)} \cdot \mathbf{a}_j \text{ and } L_k(F) = P_k(F) + J_F \cdot \mathbf{a}_k,$$

where $J_{L_j(F)}$ is the Jacobian of the polynomial system $L_j(F)$ and $\mathbf{a}_j = (0, a_{j,2}, \dots, a_{j,n})^T$ for $j = 2, \dots, k-1$. Hence, we can evaluate $P_k(F)$ at $\hat{\mathbf{x}}_e$ without computing and storing the dense Max Noether conditions L_j.

We have implemented this method in Maple. In the following table, we show the time needed for computing all $a_{i,j}$ in Example 1.

s	6	7	8	9	10
multiplicity	64	128	256	512	1024
time(sec.)	0.593	1.377	3.445	10.913	44.659

It should be noticed that the algorithm still suffer from the denseness of the Max Noether conditions if we want to store them in memory. Fortunately, for refining approximate isolated singular solutions, we don't need to store them [7]. Moreover, only the evaluations of $L_{\mu-1}(F)$ and $P_\mu(F)$ at $\hat{\mathbf{x}}$ are needed to construct the inverse system for refining $\hat{\mathbf{x}}$. In order to avoid constructing $L_j(F)$ and $P_\mu(F)$ repeatedly, we propose below a more efficient method for computing the two evaluations in every iteration.

We regard $\mathbf{a}_k$ as unknowns and construct the μ polynomial systems

$$\begin{aligned} F_1(\mathbf{x}, \mathbf{a}_1) &= J_{F_0} \cdot \mathbf{a}_1, \\ F_2(\mathbf{x}, \mathbf{a}_1, \mathbf{a}_2) &= \frac{1}{2} J_{F_1}^{\mathbf{x}} \cdot \mathbf{a}_1 + J_{F_0} \cdot \mathbf{a}_2, \\ &\cdots \\ F_{\mu-1}(\mathbf{x}, \mathbf{a}_1, \dots, \mathbf{a}_{\mu-1}) &= \sum_{j=1}^{\mu-1} \frac{j}{\mu-1} \cdot J_{F_{\mu-1-j}}^{\mathbf{x}} \cdot \mathbf{a}_j, \\ F_\mu(\mathbf{x}, \mathbf{a}_1, \dots, \mathbf{a}_{\mu-1}) &= \sum_{j=1}^{\mu-1} \frac{j}{\mu} \cdot J_{F_{\mu-j}}^{\mathbf{x}} \cdot \mathbf{a}_j, \end{aligned}$$

where $F_0 = F$, $J_{F_k}^{\mathbf{x}}$ is the first n columns of the Jacobian J_{F_k} with respect to $\mathbf{x}$, $a_{1,1} = 1$ and $a_{k,1} = 0$ for $k = 1, \dots, \mu - 1$. For a given $\hat{\mathbf{x}}$, we solve the least square problem $F_1(\hat{\mathbf{x}}, \mathbf{a}_1) = 0$ to get $\hat{\mathbf{a}}_1$, then solve $F_2(\hat{\mathbf{x}}, \hat{\mathbf{a}}_1, \mathbf{a}_2) = 0$ to get $\hat{\mathbf{a}}_2, \cdots$, until we obtain $\hat{\mathbf{a}}_{\mu-1}$. It's clear that

$$\begin{aligned} L_{\mu-1}(F)_{\mathbf{x}=\hat{\mathbf{x}}} &= F_{\mu-1}(\hat{\mathbf{x}}, \hat{\mathbf{a}}_1, \dots, \hat{\mathbf{a}}_{\mu-1}), \\ P_\mu(F)_{\mathbf{x}=\hat{\mathbf{x}}} &= F_\mu(\hat{\mathbf{x}}, \hat{\mathbf{a}}_1, \dots, \hat{\mathbf{a}}_{\mu-1}). \end{aligned}$$

For an updated $\hat{\mathbf{x}}$, we only need to solve the updated $\mu - 1$ least square problems to obtain updated $\hat{\mathbf{a}}_1, \dots, \hat{\mathbf{a}}_{\mu-1}$, then compute $P_\mu(F)_{\mathbf{x}=\hat{\mathbf{x}}}$ and $L_{\mu-1}(F)_{\mathbf{x}=\hat{\mathbf{x}}}$ by above formulas.

NOTE A threshold has to be selected properly to obtain the correct multiplicity μ in the first iteration.

3. DISCUSSION

The breadth one case root refinement has been studied in [1, 2, 3, 4, 5]. In [7] we have proved the quadratic convergency of our algorithm when $\hat{\mathbf{x}}$ is close to $\hat{\mathbf{x}}_e$, but we also notice that when $\hat{\mathbf{x}}_e$ is not well separated from other solutions of F, it is difficult to ensure that the refined solution will converge to $\hat{\mathbf{x}}_e$. We plan to compute the certified bound for the initial approximation to guarantee our algorithm converges. In [8, 9] they studied the verified error bounds for multiple roots of polynomial systems. We also plan to compute the verified error bounds for breadth-one multiple roots with arbitrary multiplicity.

4. ACKNOWLEDGEMENT

We greatly thank Bruno Salvy for fruitful discussions during his visit to our lab in November, 2010.

5. REFERENCES

[1] B. Dayton, T. Li, and Z. Zeng. Multiple zeros of nonlinear systems, 2009. preprint, `http://orion.neiu.edu/~zzeng/Papers/MultipleZeros.pdf`.

[2] M. Giusti, G. Lecerf, B. Salvy, and J.-C. Yakoubsohn. On location and approximation of clusters of zeros: case of embedding dimension one. *Found. Comput. Math.*, 7(1):1–58, 2007.

[3] A. Griewank. On solving nonlinear equations with simple singularities or nearly singular solutions. *SIAM Review*, 27(4):537–563, 1985.

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

[5] A. Leykin, J. Verschelde, and A. Zhao. Evaluation of jacobian matrices for newton's method with deflation to approximate isolated singular solutions of polynomial systems. In D. Wang and L. Zhi, editors, *Symbolic-Numeric Computation*, Trends in Mathematics, pages 269–278. Birkhäuser Basel, 2007.

[6] N. Li and L. Zhi. Compute the multiplicity structure of an isolated singular solution: case of breadth one. *MM Research Preprints*, 28:93–103, 2009. Accepted for publication in Journal of Symbolic Computation. `http://www.mmrc.iss.ac.cn/~lzhi/Publications/jsc_LiZhi2010.pdf`.

[7] N. Li and L. Zhi. Computing isolated singular solutions of polynomial systems: case of breadth one. *ArXiv Mathematics e-prints*, Mar. 2011. `http://www.arxiv.com/abs/1008.0061v2`.

[8] A. Mantzaflaris and B. Mourrain. Deflation and certified isolation of singular zeros of polynomial systems. *ArXiv Mathematics e-prints*, Jan. 2011. `http://arxiv.org/abs/1101.3140`.

[9] S. Rump and S. Graillat. Verified error bounds for multiple roots of systems of nonlinear equations. *Numerical Algorithms*, 54(3):359–377, 2009.

Generating Invariants of Hybrid Systems via Sums-Of-Squares of Polynomials with Rational Coefficients *

Min Wu and Zhengfeng Yang
Shanghai Key Laboratory of Trustworthy Computing
East China Normal University, Shanghai 200062, China
{mwu,zfyang}@sei.ecnu.edu.cn

ABSTRACT

In this paper we discuss how to generate inequality invariants for continuous dynamical systems involved in hybrid systems. A hybrid symbolic-numeric algorithm is presented to compute inequality invariants of the given systems, by transforming this problem into a parameterized polynomial optimization problem. A numerical inequality invariant of the given system can be obtained by applying polynomial Sum-of-Squares (SOS) relaxation via Semidefinite Programming (SDP). And a method based on Gauss-Newton refinement is deployed to obtain candidates of polynomials with rational coefficients, and finally we certify that this polynomial *exactly* satisfies the conditions of invariants, by use of SOS representation of polynomials with rational coefficients.

Several examples are given to show that our algorithm can successfully yield inequality invariants with rational coefficients.

Categories and Subject Descriptors: I.2.1 [Computing Methodologies]: Symbolic and Algebraic Manipulation —Algorithms; G.1.2 [Mathematics of Computing]: Numerical Analysis—Approximation

General Terms: algorithms, verification

Keywords: semidefinite programming, sum-of-squares relaxation, program verification, differential invariant

1. INTRODUCTION

Complex physical systems are systems in which the techniques of sensing, control, communication and coordination are involved and interacted with each other. Among complex physical systems, many of them are safety critical systems, such as airplanes, railway, and automotive applications. Due to the complexity, ensuring correct functioning of these systems, e.g., spatial separation, especially collision avoidance, of aircrafts during the entire flights, is among the most challenging and most important problems in computer science, mathematics and engineering.

As a common mathematical model for complex physical systems, hybrid systems [13, 3] are dynamical systems that are governed by interacting discrete and continuous dynamics [1, 13, 10]. Continuous dynamics is specified by differential equations, which is possibly subject to domain restrictions or algebraic relations resulting from physical circumstances or the interaction of continuous dynamics with discrete control. For discrete transitions, the hybrid system changes state instantaneously and possibly discontinuously, for example, the instantaneous change of control variables like the acceleration (e.g., the changing of a by setting $a := -b$ with braking force $b > 0$).

The analysis of hybrid systems is an important problem that has been studied extensively both by the control theory, and the formal verification community for over a decade. Among the most important analysis questions for hybrid systems are those of *safety*, i.e., deciding whether a given property ψ holds in all the reachable states, and the dual problem of *reachability*, i.e., deciding if a state satisfying the given property ψ is reachable. Both these problems are closely related to the problem of generating invariants of hybrid systems.

An invariant [31] of a hybrid system is a property that holds in all the reachable states of the system, or, in other words, an over-approximation of all the reachable states of the system. Invariants are useful facts about the dynamics of a given system and are widely used in numerous approaches to verify and understand systems. The invariants of hybrid systems are used to establish temporal properties of systems such as safety, stability, termination, progress and so on [19, 7, 28, 23].

The problem of generating invariants of an arbitrary form is known to be computationally hard, intractable even for the simplest classes. The usual technique for generating invariants is to generate an *inductive invariant*, i.e., an assertion that holds at the initial states of the system, and is preserved by all discrete and continuous state changes. There has been a considerable volume of work towards invariant generation for hybrid systems using techniques from convex optimization, commutative algebraic and semi-algebraic geometry [36, 4, 32, 21, 22, 33, 23, 12, 33, 26, 24, 20, 27, 31]. Many of these techniques synthesize invariants of prespecified form by computing constraints on the unknown co-

[*]Min Wu is supported by the Chinese National Natural Science Foundation under Grants 10801052 and 91018012.
Zhengfeng Yang is supported by the Chinese National Natural Science Foundation under Grants 10901055 and 91018012, the China National 973 Project(No.2011CB302802), and Shanghai Natural Science Foundation under Grant 09ZR1408800.

efficients of a single or fixed number of polynomial inequalities with a bounded degree, that guarantee that any solution will also be inductive. Carbonell and Tiwari [4] presented a powerful computational method for automatically generating polynomial invariants, which does not assume bounded degree, whereas their method is only applicable to linear hybrid systems. In [2], Asarin et al. employed a numerical approach based on conservative approximation to deal with nonlinear systems. However, as pointed out in [25], numerical errors resulted from numerical computation tended to cause unsoundness of the verification of hybrid systems. Recently, Platzer et al. proposed a powerful theorem-proving framework [21, 22, 23, 26, 24, 27] for verifying properties of hybrid systems through differential logic that extends dynamic logics using differential operators and quantifier elimination [6]. However, it is well known that the bottleneck of quantifier elimination algorithms lies in their high complexity, which is doubly exponential.

In this work, we study how to generate inequality invariants for nonlinear continuous systems given by polynomial vector fields. In virtue of the efficiency of numerical computation and the error-free property of symbolic computation, we present a symbolic-numeric hybrid method, based on Sum-of-Squares (SOS) relaxation via semidefinite programming (SDP) and exact SOS representation recovery, to generate inequality invariants of continuous systems. More specifically, computing inequality invariants of a continuous system is transformed into solving a semi-algebraic system with parameters, while the latter system is constructed by predetermining the template of the polynomial invariant with the given degree. It is noticed that SDP is applicable to polynomial optimization problem by use of Sum-of-Squares relaxation. Since Matlab packages, such as *SOSTOOLS* [29] and *Yalmip* [18], that deal with SDP problem, are running in the fixed precision, these algorithms yield only numerical solutions. However, numerical invariants may not be the correct invariants of the given continuous system. In [14, 11], Kaltofen et al. provided a symbolic-numeric hybrid method, combined with the SDP solvers and rational vector recovery techniques, to verify whether a multivariate polynomial is positive semidefinite by demonstrating the corresponding exact SOS representation. In this paper, we also deploy this symbolic-numeric method to compute inequality-form invariants of the given continuous systems. The idea is as follows: (a) From the conditions of invariants, we construct an SDP system to solve the associated semi-algebraic system with parameters; (b) an exact invariant is obtained by recovering the exact SOS representation from the approximate SOS representation yielded from the SDP solvers. During the process of recovery step, we need apply Gauss-Newton iteration to refine the approximate SOS representation. More details can be found in Section 3.

In comparison with other symbolic approaches of invariant generation based on qualifier elimination theory, our approach is more efficient and practical, because the SDP systems constructed from the continuous system can be solved in polynomial time. Furthermore, we apply Gauss-Newton refinement technique and rational vector recovery to obtain an exact invariant of the given system. Having the exact representation of polynomials as sums of squares with rational coefficients, the invariant obtained by our approach can be easily verified to satisfy its constraints *exactly*. Therefore, our approach is able to overcome the weakness of numerical approaches on approximate invariant computation, as claimed in [25], .

The rest of the paper is organized as follows. In Section 2, we introduce the notions of hybrid systems and invariants. Section 3 is devoted to illustrating a symbolic-numeric approach for generating invariants of continuous systems. In Section 4, we show two examples of inequality invariant generation. Section 5 concludes the paper.

2. INVARIANTS

To model hybrid systems, we use hybrid automata [13, 33].

Definition 1 (Hybrid Systems). *A* hybrid system $\mathbf{H}$: $\langle V, L, \mathcal{T}, \Theta, \mathcal{D}, \psi, \ell_0\rangle$ *consists of the following components:*

- V*, a set of real-valued system* variables*. A* state *is an interpretation of* V*, assigning to each* $x \in V$ *a real value. An* assertion *is a first-order formula over* V*. A state* s *satisfies an assertion* φ*, written as* $s \models \varphi$*, if* φ *holds on* s*. We will also write* $\varphi_1 \models \varphi_2$ *for two assertions* φ_1, φ_2 *to denote that* φ_2 *is true at least in all the states in which* φ_1 *is true;*
- L*, a finite set of locations;*
- $\mathcal{T}$*, a set of (discrete) transitions. Each transition* $\tau : \langle \ell_1, \ell_2, \rho_\tau\rangle \in \mathcal{T}$ *consists of a prelocation* $\ell_1 \in L$*, a postlocation* $\ell_2 \in L$*, and an assertion* ρ_τ *over* $V \cup V'$ *representing the next-state relation, where* V' *denotes the values of* V *in the next state;*
- Θ*, an assertion specifying the* initial *condition;*
- $\mathcal{D}$*, a map that maps each location* $\ell \in L$ *to a* differential rule *(also known as a* vector field *or a* flow field*)* $\mathcal{D}(\ell)$*, of the form* $\dot{x}_i = \frac{dx_i}{dt} = f_i(V)$ *for each* $x_i \in V$*, with* t *the time variable. The differential rule at a location specifies how the system variables evolve in that location;*
- ψ*, a map that maps each location* $\ell \in L$ *to a* location condition (location invariant) $\psi(\ell)$*, an assertion over* V*;*
- $\ell_0 \in L$*, the* initial location*; we assume that the initial condition satisfies the location invariant at the initial location, that is* $\Theta \models \psi(\ell_0)$*.*

The following definition of invariants for hybrid systems comes from [33].

Definition 2 (Invariant). *[33, Definition 3] An* invariant $\mathcal{I}$ *of a hybrid system at a location* ℓ *is an assertion* $\mathcal{I}$ *such that for any reachable state* $\langle \ell, \mathbf{x}\rangle$ *of the hybrid system,* $\mathbf{x} \models \mathcal{I}$*.*

The problem of generating invariants of arbitrary form is known to be computationally hard, intractable even for the simplest classes. The usual technique for generating invariants is to generate *inductive invariants*, which are defined as follows.

Definition 3 (Inductive Invariant). *An* inductive assertion map $\mathcal{I}$ *of a hybrid system* $\mathbf{H}$: $\langle V, L, \mathcal{T}, \Theta, \mathcal{D}, \psi, \ell_0\rangle$ *is a map that associates with each location* $\ell \in L$ *an assertion* $\mathcal{I}(\ell)$ *that holds initially and is preserved by all discrete* *transitions and continuous flows of* $\mathbf{H}$. *More formally an inductive assertion map satisfies the following requirements:*

(i) **[Initial]** $\Theta \models \mathcal{I}(\ell_0)$.

(ii) **[Discrete Consecution]** *For each discrete transition* $\tau : \langle \ell_1, \ell_2, \rho_\tau \rangle$, *starting from a state satisfying* $\mathcal{I}(\ell_1)$, *and taking* τ *leads to a state satisfying* $\mathcal{I}(\ell_2)$. *Formally,*

$$\mathcal{I}(\ell_1) \wedge \rho_\tau \models \mathcal{I}(\ell_2)',$$

where $\mathcal{I}(\ell_2)'$ *represents the assertion* $\mathcal{I}(\ell_2)$ *with the current state variables* $x_1, \ldots, x_n$ *replaced by the next state variables* $x_1', \ldots, x_n'$, *respectively.*

(iii) **[Continuous Consecution]** *For every location* $\ell \in L$ *and states* $\langle \ell, \mathbf{x}_1 \rangle, \langle \ell, \mathbf{x}_2 \rangle$ *such that* $\mathbf{x}_2$ *evolves from* $\mathbf{x}_1$ *according to the differential rule* $\mathcal{D}(\ell)$ *at* ℓ, *if* $\mathbf{x}_1 \models \mathcal{I}(\ell)$ *then* $\mathbf{x}_2 \models \mathcal{I}(\ell)$.

Remark 1. Clearly, Definition 3 generalizes the notions of invariants for both discrete and continuous dynamical systems. An assertion $\mathcal{I}$ is called a *discrete (inductive) invariant* if $\mathcal{I}$ satisfies the conditions (i) and (ii); and $\mathcal{I}$ is called a *differential invariant* if $\mathcal{I}$ satisfies the conditions (i) and (iii).

The usual way to compute invariants of hybrid systems is to compute discrete invariants and differential invariants separately. A lot of techniques are available for computing invariants of discrete loop programs. The reader can refer to [37, 8, 15, 9, 34, 30, 16, 5] for more details.

In this work, we only concentrate on how to compute differential invariants of nonlinear continuous dynamical systems given by polynomial vector fields, i.e., systems of the form

$$\dot{x}_1 = p_1(x_1, \ldots, x_n), \quad \cdots, \quad \dot{x}_n = p_n(x_1, \ldots, x_n) \qquad (1)$$

where $p_i \in \mathbb{R}[x_1, \ldots, x_n]$ for $1 \le i \le n$.

[**Convention**] In what follows, we will adopt the following notations:

(i) The boldfaced symbols $\mathbf{x}$ and $\mathbf{p}$ denote the vectors

$$(x_1, \ldots, x_n) \quad \text{and} \quad (p_1, \ldots, p_n),$$

respectively;

(ii) For a smooth function $V(\mathbf{x}) : \mathbb{R}^n \mapsto \mathbb{R}$, the notation $\mathbf{d}V$ denotes the $1 \times n$ row vector $(\frac{\partial V}{\partial x_1}, \ldots, \frac{\partial V}{\partial x_n})$ of partial derivatives of V with respect to the variables $x_1, \ldots, x_n$;

(iii) The expression $\mathbf{d}V \cdot \mathbf{p}$ represents the inner product of the vectors $\mathbf{d}V$ and $\mathbf{p}$. Actually we have $\mathbf{d}V \cdot \mathbf{p} = \frac{dV}{dt}$.

We will study how to generate inequality-form invariants of the system $\dot{\mathbf{x}} = \mathbf{p}$, i.e., invariants of the form $F \ge 0$ where F is an expression in the system variables $x_1, \ldots, x_n$ and the initial values $\mathbf{x}(0)$. The following lemma will be needed in latter discussion.

Lemma 1. *Let* $\dot{\mathbf{x}} = \mathbf{p}$ *be a nonlinear dynamical system with initial states* Θ *and state invariants* ψ. *For a polynomial* $V(\mathbf{x}) \in \mathbb{R}[\mathbf{x}]$, *define*

$$d_0 = \min\{V(\mathbf{x}(0)) : \mathbf{x}(0) \in \Theta\},$$

and set $r(\mathbf{x}) = \frac{dV}{dt} = \mathbf{d}V \cdot \mathbf{p}$. *If the polynomial* $r(\mathbf{x})$ *is positive semidefinite within the state invariants* ψ, *meaning that* $r(\mathbf{x})$ *is positive semidefinite for all* $\mathbf{x}$ *such that* ψ *hold, then the formula* $V(\mathbf{x}) - d_0 \ge 0$ *is a differential invariant of the dynamical system* $\dot{\mathbf{x}} = \mathbf{p}$.

Remark 2. In order to compute the differential invariant $V(\mathbf{x}) - d_0 \ge 0$ as in Lemma 1, we need compute both the polynomial $V(\mathbf{x})$ and the constant d_0. Actually, once $V(\mathbf{x})$ is obtained, there are several approaches to obtain the minimum d_0 of $V(\mathbf{x})$ in Θ. Hereafter, we will omit the discussion of d_0, and focus on computing polynomials $V(\mathbf{x})$ such that $r(\mathbf{x}) = \mathbf{d}V \cdot \mathbf{p}$ is positive semidefinite within the state invariants ψ. For simplicity, in the sequel, by saying $V(\mathbf{x})$ is a polynomial invariant we mean that the polynomial inequality $V(\mathbf{x}) - d_0 \ge 0$ is an invariant of the given nonlinear dynamical system, where $d_0 = \min\{V(\mathbf{x}(0)) : \mathbf{x}(0) \in \Theta\}$.

3. POLYNOMIAL INEQUALITY INVARIANT GENERATION

According to Lemma 1 and Remark 2, in order to compute invariants of a nonlinear system $\dot{\mathbf{x}} = \mathbf{p}$ with the state invariants ψ, one may obtain a polynomial $V(\mathbf{x})$ such that $r(\mathbf{x}) = \mathbf{d}V \cdot \mathbf{p}$ is positive semidefinite within ψ. In general, we consider the state invariants ψ with the following form:

$$\psi_1 \ge 0 \wedge \cdots \wedge \psi_k \ge 0, \qquad (2)$$

where $\psi_1, \cdots, \psi_k \in \mathbb{R}[\mathbf{x}]$. The problem of computing polynomial invariants can be transformed into the following problem

$$\begin{array}{ll} \text{find} & V \in \mathbb{R}[\mathbf{x}] \\ & \text{s.t.} \quad r = \mathbf{d}V \cdot \mathbf{p} \ge 0 \text{ if } \psi_1 \ge 0 \wedge \cdots \wedge \psi_k \ge 0. \end{array} \qquad (3)$$

Our idea of computing $V(\mathbf{x})$, based on SOS relaxation and rational vector recovery, is as follows.

Step 1: Predetermine a template of polynomial invariants with the given degree and convert the problem of computing polynomial invariants to the associated (parametric) polynomial optimization problem. SOS relaxation method is then applied to obtain an approximate polynomial invariant with floating point coefficients.

Step 2: Apply Gauss-Newton refinement and rational vector recovery on the approximate polynomial invariant to get one polynomial with rational coefficients, which satisfies exactly the conditions of invariants of the given dynamical system.

Steps 1 and 2 will be explained in more details in Sections 3.1 and 3.2, respectively.

3.1 Sum of Squares Relaxation

To solve the problem (3), let us predetermine a template of polynomial invariants with the given degree d, that is, we assume

$$V(\mathbf{x}) = \sum_{\alpha} \mathbf{c}_\alpha \mathbf{x}^\alpha, \qquad (4)$$

where $\mathbf{c}_\alpha \in \mathbb{R}$ are parameters, $\mathbf{x}^\alpha = x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ and $\alpha \in \mathbb{Z}^n_{\ge 0}$ with $\sum_{i=1}^n \alpha_i \le d$.

We first consider a simpler case where there exists no state invariants. The problem then becomes computing a polynomial $V \in \mathbb{R}[\mathbf{x}]$ such that

$$r(\mathbf{x}) = \mathbf{d}V \cdot \mathbf{p} \geq 0, \qquad \forall\, \mathbf{x} \in \mathbb{R}^n.$$

It is obvious that a sufficient condition for $r(\mathbf{x})$ to be positive semidefinite for arbitrary value $\mathbf{x} \in \mathbb{R}^n$ is that there exists an SOS decomposition of $r(\mathbf{x})$:

$$r(\mathbf{x}) = \sum_i {f_i}^2(\mathbf{x}), \quad \text{with } f_i(\mathbf{x}) \in \mathbb{R}[\mathbf{x}], \tag{5}$$

or, equivalently, there exists a following representation of $r(\mathbf{x})$:

$$r(\mathbf{x}) = m(\mathbf{x})^T \cdot W \cdot m(\mathbf{x}),$$

where W is a real symmetric and positive semidefinite matrix and $m(\mathbf{x})$ is a vector of terms in $\mathbb{R}[\mathbf{x}]$ with degree $\leq d/2$. The SOS problem (5) can be further converted into the following SDP program

$$\left.\begin{array}{l} \inf\limits_{W} \;\; \mathrm{Trace}(W) \\ \text{s. t. } r(\mathbf{x}) = m(\mathbf{x})^T \cdot W \cdot m(\mathbf{x}) \\ \qquad W \succeq 0, W^T = W. \end{array}\right\} \tag{6}$$

(here $\mathrm{Trace}(W)$ acts as a dummy objective function that is commonly used in SDP for optimization problem with no objective functions.)

Now let us consider the case where the state invariants are given as in (2). Similar to the case with no state invariants, we predetermine the template (4) of $V(\mathbf{x})$ with the given degree d, where the coefficients c_α are parameters. One can certainly apply quantifier elimination methods to solve the corresponding parametric semi-algebraic system. Some Maple packages such as DISCOVERER [38] are available to solve this kind of problems. For the given template, quantifier elimination methods yield the necessary and sufficient conditions for the existence of invariants with given degree. However, quantifier elimination methods are of high complexity since they rely on the cylindrical algebraic decomposition (CAD) algorithm. Instead in this paper, we will explore the SOS relaxation techniques based on semidefinite programming to obtain polynomial invariants. These techniques supply a sufficient condition for the existence of $V(\mathbf{x})$.

Theorem 1. *Let $\dot{\mathbf{x}} = \mathbf{p}$ be a nonlinear dynamical system with initial states Θ and state invariants*

$$\psi := \psi_1 \geq 0 \wedge \cdots \wedge \psi_k \geq 0.$$

Suppose there exists a polynomial $V(\mathbf{x}) \in \mathbb{R}[\mathbf{x}]$ such that $r(\mathbf{x}) = \mathbf{d}V \cdot \mathbf{p}$ can be written as

$$\begin{array}{rcl} r(\mathbf{x}) & = & \sum\limits_{i=0}^{k} \sigma_i \psi_i, \\ & & \text{where } \sigma_0, \ldots, \sigma_k \text{ are SOSs and } \psi_0 = 1, \end{array} \tag{7}$$

then the formula $V(\mathbf{x}) - d_0 \geq 0$ with

$$d_0 = \min\{V(\mathbf{x}(0)) : \mathbf{x}(0) \in \Theta\},$$

is a differential invariant of the dynamical system $\dot{\mathbf{x}} = \mathbf{p}$.

Proof. It is easy to verify that if $r(\mathbf{x})$ has the form (7) then $r(\mathbf{x}) \geq 0$ holds for any $\mathbf{x} \in \mathbb{R}^n$ such that ψ holds. The conclusion then follows from Lemma 1. □

A weaker sufficient condition than (7) can be presented using cross products of ψ_i's, as described in the following corollary:

Corollary 1. *Under the assumptions of Theorem 1, if the polynomial $r(\mathbf{x})$ can be written as*

$$\begin{array}{rcl} r(\mathbf{x}) & = & \sum\limits_{\mu \in \{0,1\}^k} \sigma_\mu \psi_\mu \qquad \text{where } \psi_\mu = \psi_1^{\mu_1} \cdots \psi_k^{\mu_k} \\ & & \sigma_\mu \text{ are all SOSs,} \end{array} \tag{8}$$

then the formula $V(\mathbf{x}) - d_0 \geq 0$ with

$$d_0 = \min\{V(\mathbf{x}(0)) : \mathbf{x}(0) \in \Theta\},$$

is a different invariant of the given dynamical system.

Given the degree bound $2N$ of $\sigma_i(\mathbf{x})$ with $N \in \mathbb{Z}^+$, one can easily construct an SOS program of the form (7), in which the decision variables are the coefficients of $V(\mathbf{x})$ and $\sigma_i(\mathbf{x})$, for $0 \leq i \leq k$. Similar to (6), the SOS program (7) is equivalent to the following SDP problem with a block form:

$$\left.\begin{array}{ll} \inf\limits_{W^{[0]},\ldots,W^{[k]}} & \sum_{i=0}^{k} \mathrm{Trace}(W^{[i]}) \\ \text{s. t.} & r(\mathbf{x}) = \sum_{i=0}^{k} m_i(\mathbf{x})^T \cdot W^{[i]} \cdot m_i(\mathbf{x})\psi_i \\ & W^{[i]} \succeq 0, (W^{[i]})^T = W^{[i]}, 0 \leq i \leq k. \end{array}\right\} \tag{9}$$

The SOS program corresponding to (8) would be obtained likewise. Many Matlab packages of SDP solvers, such as SOSTOOLS [29], YALMIP [18], and SeDuMi [35], are available to solve efficiently the problems (6) and (9).

3.2 Exact Certificate of Sum of Squares Decomposition

Since the SDP solvers in Matlab is running in fixed precision, the techniques in Section 3.1 will yield numerical solutions to the associated SDP problems of (5, 7, 8), and therefore yield numerical solutions to (5, 7, 8), respectively. For instance, applying an SDP solver to (9), we obtain a numerical polynomial $V^*(\mathbf{x})$ with floating point coefficients and some numerical positive semidefinite matrices $W^{[i]}$, which satisfy approximately

$$r^*(\mathbf{x}) \approx \sum_{i=0}^{k} m_i(\mathbf{x})^T \cdot W^{[i]} \cdot m(\mathbf{x})\psi_i \quad \text{and } W^{[i]} \succsim 0, \tag{10}$$

where $r^*(\mathbf{x}) = \mathbf{d}V^*(\mathbf{x}) \cdot \mathbf{p}$. However, due to round-off errors, $V^*(\mathbf{x}) - d_0 \geq 0$ may not necessarily be an invariant of the given dynamical system because $r^*(\mathbf{x})$ may not be positive definite within ψ exactly. Therefore in the next step, from the numerical polynomial $V^*(\mathbf{x})$ and the numerical positive semidefinite matrices $W^{[i]}$, we will try to recover a polynomial $V(\mathbf{x})$ with rational coefficients, which satisfies (7) *exactly*.

In [14, 11], the authors proposed a method to certify a lower bound of global optimum of a polynomial via SOS representation. In this paper, we will employ this method to compute invariants of continuous dynamical systems. We first recall the idea of this method. Given a multivariate polynomial $f \in \mathbb{R}[\mathbf{x}]$, they first constructed the associated SOS program and solve the SDP system to obtain an approximate lower bound r^* and an approximate positive semidefinite matrix W such that

$$f(\mathbf{x}) - r^* \approx m(\mathbf{x})^T \cdot W \cdot m(\mathbf{x}) \quad \text{with} \quad W \succsim 0.$$

Next, they rounded r^* to a nearby rational number r and, for a given tolerance τ, applied Gauss-Newton iteration to refine W such that

$$\|f(\mathbf{x}) - r - m(\mathbf{x})^T \cdot W \cdot m(\mathbf{x})\| \leq \tau \text{ and } W \succeq 0.$$

In the end, a positive semidefinite matrix $\widetilde{W}$ with rational entries would be obtained by orthogonal projection method or rational vector recovery method such that

$$f(\mathbf{x}) - r = m(\mathbf{x})^T \cdot \widetilde{W} \cdot m(\mathbf{x}), \quad \text{with } \widetilde{W} \succeq 0. \tag{11}$$

Therefore, r is a certified lower bound of the global optimum of f because r and $\widetilde{W}$ exactly satisfy (11).

In comparison with the problem of the lower bound verification in [14], finding a polynomial invariant $V(\mathbf{x})$ of nonlinear continuous dynamical systems is a bit different. To obtain a certified polynomial invariant $V(\mathbf{x})$, we need consider the following two problems:

P.1: Instead of obtaining a single rational number r, we need find a rational vector $\mathbf{v}$ denoting the coefficient vector of $V(\mathbf{x})$.

P.2: $V(\mathbf{x})$ should exactly satisfy (7) or (8), i.e., not only the σ_i's and σ_μ's satisfy the equalities (7) and (8) respectively, but also they have exact SOS representations simultaneously.

In Section 3.1, a numerical vector $\mathbf{v}^*$ that denotes the (numerical) coefficient vector of $V^*(\mathbf{x})$ is obtained by solving some SDP system. For the given bound of the common denominator of $\mathbf{v}^*$, we can recover the numerical vector $\mathbf{v}^*$ to a vector $\mathbf{v}$ with rational entries by a simultaneous Diophantine approximation algorithm [17]. This solves the first problem.

Let us focus on the second problem. Here we only consider the case (7), and the case (8) can be handled likewise. From (9), we will construct another SDP system given by one big matrix $W = \text{diag}(\mathrm{W}^{[0]}, \ldots, \mathrm{W}^{[k]})$, and, by use of SDP solvers, some numerical positive semidefinite matrices $W^{[0]}, \ldots, W^{[k]}$ and an approximate form of (9) will be obtained. The recovery of $W^{[0]}, \ldots, W^{[k]}$ into rational positive semidefinite matrices is split into two steps: we first recover $\widetilde{W}^{[1]}, \ldots, \widetilde{W}^{[k]}$ and then recover $\widetilde{W}^{[0]}$.

Step 1. Given the numerical positive definite matrices $W^{[i]}$, $1 \leq i \leq k$, we find the nearby rational positive semidefinite matrices $\widetilde{W}^{[i]}$. In practice, all $W^{[i]}$ are very simple, and by setting the small entries of $W^{[i]}$ to be zeros we easily get the nearby rational positive semidefinite matrices $\widetilde{W}^{[i]}$ for $i = 1, \ldots, k$.

Step 2. Having $\mathbf{v}$ and $\widetilde{W}^{[1]}, \ldots, \widetilde{W}^{[k]}$, (9) is converted to

$$\left.\begin{array}{r} r(\mathbf{x}) - \sum_{i=1}^{k} m_i(\mathbf{x})^T \cdot \widetilde{W}^{[i]} \cdot m_i(\mathbf{x})\psi_i \\ \approx m_0(\mathbf{x})^T \cdot W^{[0]} \cdot m_0(\mathbf{x}), \\ W^{[i]} \succeq 0,\ 1 \leq i \leq k,\ W^{[0]} \succsim 0, \end{array}\right\} \tag{12}$$

where $r(\mathbf{x}) = \mathbf{d}V(\mathbf{x}) \cdot \mathbf{p}$ with $V(\mathbf{x})$ the polynomial corresponding to the coefficient vector $\mathbf{v}$. Observing in (12), the matrix $W^{[0]}$, obtained from the computation in Step 1, has floating point entries, while the $\widetilde{W}^{[i]}$, $1 \leq i \leq k$, are rational positive semidefinite matrices. Therefore, the remaining task of (**P.2**) is to find a nearby rational positive semidefinite matrix $\widetilde{W}^{[0]}$ such that the equation in (12) holds exactly, which is exactly the same problem as described in [14]. Therefore, we can apply Gauss-Newton iteration to refine $W^{[0]}$, and recover a rational positive definite matrix $\widetilde{W}^{[0]}$ from the refined $W^{[0]}$ by orthogonal projection if $W^{[0]}$ is of full rank, or by rational vector recovery method otherwise.

Remark 3. In practice, we will do as follows to fulfil the above task of finding a rational positive semidefinite matrix $\widetilde{W}^{[0]}$ that satisfies (12) exactly. First, using the rational polynomial $V(\mathbf{x})$ determined by the coefficient vector $\mathbf{v}$, and rational positive semidefinite matrices $\widetilde{W}^{[1]}, \ldots, \widetilde{W}^{[k]}$, we will reconstruct an SDP system of the form (12), which gives us a better initial point for Gauss-Newton iteration. Then, we will compute numerical solutions for $W^{[0]}$ of (12) using SDP solvers, and finally apply the method in [14] to recover $\widetilde{W}^{[0]}$.

The above discussion leads to an algorithm of computing the certified polynomial inequality invariants of the dynamical system $\dot{\mathbf{x}} = \mathbf{p}$ with state invariants ψ. As stated above, we only present the case where the invariant $V(\mathbf{x})$ satisfies (7), and the case of (8) is similar.

3.3 Algorithm

Algorithm *Polynomial Inequality Invariant Generation*

Input:
- ▸ $\dot{\mathbf{x}} = \mathbf{p}$: a continuous dynamical system.
- ▸ $d \in \mathbb{Z}_{>0}$: the degree bound of the candidate polynomial invariant.
- ▸ $D \in \mathbb{Z}_{>0}$: the bound of the common denominator of the coefficient vector of the polynomial invariant.
- ▸ $N \in \mathbb{Z}_{\geq 0}$: the degree bound $\deg(\sigma_i) \leq 2N$ used to construct the SDP system.
- ▸ $\tau \in \mathbb{R}_{>0}$: the given tolerance.

Output:
- ▸ $V(\mathbf{x})$: a verified polynomial invariant.

1. Compute the candidates of polynomial invariants
   - (a) **Case 1:** the state invariant is empty:
     - (i) Predetermine the template of $V(\mathbf{x})$ with degree d and construct an SDP system of form (6).
       - If the SDP system (6) has no feasible solutions,
         return "we can't find polynomial invariants with degree $\leq d$;"
       - Otherwise,
         obtain a numerical vector $\mathbf{v}^*$ and a numerical positive semidefinite matrix W.
     - (ii) For the common denominator bound D, compute from $\mathbf{v}^*$ a rational vector $\mathbf{v}$ by Diophantine approximation algorithm, and then the associated rational polynomial $V(\mathbf{x})$.
   - (b) **Case 2:** the state invariant is given as

$$\psi : \psi_1 \geq 0 \wedge \ldots \wedge \psi_k \geq 0. \tag{13}$$

     - (i) Set up the SDP system (9) by predetermining the template of $V(\mathbf{x})$ with $\deg(V) = d$ and predetermining the template of σ_j with $\deg(\sigma_j) \leq 2N$.
       - If the SDP system (9) has no feasible solutions,
         return "we can't find polynomial invariants with degree $\leq d$;"

- Otherwise,
  obtain a numerical vector $\mathbf{v}$ and numerical positive semidefinite matrices $W^{[0]}$ and $W^{[i]}$, $1 \le i \le k$.

(ii) For the common denominator bound D, compute a rational vector $\mathbf{v}$ using Diophantine approximation algorithm and get the associate rational polynomial $V(\mathbf{x})$.

(iii) Convert all the $W^{[i]}$, $1 \le i \le k$, into rational matrices $\widetilde{W}^{[i]}$ that are positive semidefinite.

2. Compute the exact SOS decomposition

(a) **Case 1:** the state invariant is empty:

(i) Use $V(\mathbf{x})$ to reconstruct an SDP system of form (6) and get an approximate SOS decomposition:

$$r(\mathbf{x}) \approx m(\mathbf{x})^T \cdot W^{[0]} \cdot m(\mathbf{x}) \qquad (14)$$

with $r(\mathbf{x}) = \mathbf{d}V(\mathbf{x}) \cdot \mathbf{p}$.

(ii) Apply Gauss-Newton iteration to refine $W^{[0]}$ in (14).

Case 2: the state invariant is given as (13)

(i) Reconstruct an SDP system of the form (12) to get an approximate positive semidefinite matrix $W^{[0]}$ satisfying

$$r(\mathbf{x}) - \sum_{i=1}^{k} m_i(\mathbf{x})^T \cdot \widetilde{W}^{[i]} \cdot m_i(\mathbf{x})\psi_i \approx m_0(\mathbf{x})^T \cdot W^{[0]} \cdot m_0(\mathbf{x}) \qquad (15)$$

with $r(\mathbf{x}) = \mathbf{d}V(\mathbf{x}) \cdot \mathbf{p}$.

(ii) Apply Gauss-Newton iteration to refine $W^{[0]}$ in (15).

(b) From the refined $W^{[0]}$, compute a rational matrix $\widetilde{W}^{[0]}$ satisfying (15) exactly by orthogonal projection method if $W^{[0]}$ is of full rank, or by rational vector recovery if $W^{[0]}$ is singular.

(c) Check whether $\widetilde{W}^{[0]}$ is positive semidefinite.

- If so, return $V(\mathbf{x})$;
- Otherwise,
  return "we can't find polynomial invariants with degree $\le d$."

Remark 4. Our algorithm cannot guarantee a rational solution will always be found since there are some limitations of the above algorithm such as choosing the degree bound N and the common denominator bound D. Furthermore, it is difficult to determine in advance whether there exists differential invariants with rational coefficients or not. Therefore, even if our algorithm cannot find differential invariants, it does not mean that the given dynamical system has no differential invariants.

4. EXPERIMENTS

In this section, two examples are given to illustrate our algorithm for computing polynomial inequality invariants.

Example 1. Consider the nonlinear system

$$\dot{x_1} = x_1 + x_2^2, \qquad \dot{x_2} = x_1^2 + x_2.$$

Assume that $\deg(V) = 2k + 1$ for $k = 0, 1, 2, \ldots$. For $\deg(V) = 1$, there exists no feasible solutions of the corresponding SDP system. Now set $\deg(V(x_1, x_2)) = 3$ and let the coefficients of $V(x_1, x_2)$ be parameters. By solving the associated SDP system, we obtain a numerical polynomial $V^*(x_1, x_2)$ as follows

$$\begin{aligned} V^*(x_1, x_2) =& 0.0778x_1^3 + 0.1287x_1^2x_2 + 0.3406x_1^2 + 0.1287x_1x_2^2 \\ & - 0.06165x_1x_2 - 1.75 \times 10^{-6}x_1 + 0.0778x_2^3 \\ & + 0.3406x_2^2 - 1.75 \times 10^{-6}x_2. \end{aligned}$$

Let $\tau = 10^{-2}$, and round the coefficients of $V^*(x_1, x_2)$ to be rational numbers with the common denominator bound 10^4. We obtain a rational polynomial

$$\begin{aligned} V(x_1, x_2) =& \frac{33}{424}x_1^3 + \frac{191}{1484}x_1^2x_2 + \frac{1011}{2968}x_1^2 + \frac{191}{1484}x_1x_2^2 \\ & - \frac{183}{2968}x_1x_2 + \frac{33}{424}x_2^3 + \frac{1011}{2968}x_2^2. \end{aligned}$$

Therefore,

$$\begin{aligned} r(x_1, x_2) = \frac{\mathbf{d}V}{dt} =& \frac{99}{212}x_1^2x_2^2 + \frac{255}{1484}x_1^3 + \frac{191}{742}x_1x_2^3 + \frac{396}{371}x_1^2x_2 \\ & + \frac{396}{371}x_1x_2^2 + \frac{1011}{1484}x_1^2 + \frac{191}{1484}x_2^4 + \frac{255}{1484}x_2^3 \\ & - \frac{183}{1484}x_1x_2 + \frac{191}{1484}x_1^4 + \frac{191}{742}x_1^3x_2 + \frac{1011}{1084}x_2^2. \end{aligned}$$

By running the algorithm in Section 3.3, we find that $r(x_1, x_2)$ can be written as an SOS of 4 polynomials with rational coefficients. The SOS representation of $r(x_1, x_2)$ is given in Appendix. Therefore,

$$V(x_1, x_2) - d_0 \ge 0,$$

with $d_0 = \min\{V(x_1, x_2) : (x_1, x_2) \in \Theta\}$, is an invariant of the given nonlinear system.

The next example shows how to compute polynomial inequality invariants for nonlinear systems with state invariants.

Example 2. Consider the following nonlinear system

$$\dot{x_1} = x_2 + x_1x_2^2, \qquad \dot{x_2} = -x_1 + x_1^2x_2.$$

Assume that the state invariant is $x_1 \ge 0 \wedge x_2 \ge 0$. Let us assume that $\deg(V) = 2k, k = 1, 2, \ldots$. According to Corollary 1, we construct the associated SDP system as (8), that is, the polynomial

$$\mathbf{d}V \cdot \mathbf{p} - \sigma_1(x_1, x_2)x_1 - \sigma_2(x_1, x_2)x_2 - \sigma_3(x_1, x_2)x_1x_2$$

can be written as an SOS, and $\sigma_i(x_1, x_2), i = 1, 2, 3$ can also be written as SOSs.

Suppose the degree bounds of $V(x_1, x_2)$ and of $\sigma_i(x_1, x_2)$ are all 2.

Solve the associated SDP system and find numerical solutions:

$$V^*(x_1, x_2) = 0.424\,x_1^2 + 0.424\,x_2^2 + \cdots + 2.286 \times 10^{-13}x_2,$$

and

$$\begin{aligned} \sigma_1^*(x_1, x_2) =& -1.171 \times 10^{-12}x_1^2 + \cdots + 2.121 \times 10^{-5}x_2^2, \\ \sigma_2^*(x_1, x_2) =& 1.426 \times 10^{-5}x_1^2 + \cdots + 8.87 \times 10^{-13}x_2^2, \\ \sigma_3^*(x_1, x_2) =& 2.917 \times 10^{-11}x_1^2 + \cdots + 3.367 \times 10^{-11}x_2^2. \end{aligned}$$

Set $\tau = 10^{-2}$, and round the coefficients of $V^*(x_1, x_2)$ as rational numbers with the common denominator bound 100. We obtain a rational polynomial

$$V(x_1, x_2) = \frac{14}{33}x_1^2 + \frac{14}{33}x_2^2,$$

and

$$\sigma_i(x_1, x_2) = 0, i = 1, 2, 3.$$

It is easy to verify that

$$\frac{\mathbf{d}V}{dt} - \sigma_1(x_1, x_2)x_1 - \sigma_2(x_1, x_2)x_2 - \sigma_3(x_1, x_2)x_1x_2 = \frac{56}{33}x_1^2x_2^2,$$

and is thus positive semidefinite. So,

$$V(x_1, x_2) - d_0 \geq 0, \quad \text{or equivalently,} \quad x_1^2 + x_2^2 - d_0 \geq 0,$$

with $d_0 = \min\{V(x_1, x_2) : (x_1, x_2) \in \Theta\}$, is an invariant of the given nonlinear system with the state invariants $x_1 \geq 0$ and $x_2 \geq 0$.

For degree bounds 4 and 6 of the polynomial $V(x_1, x_2)$, the associated SDP systems have no feasible solutions.

Now suppose the degree bound of $V(x_1, x_2)$ is 8. By solving the associated SDP system, we get numerical solutions

$$V^* = 0.228x_1^4x_2^4 + 0.006x_1^4x_2^3 + 0.197x_1^4x_2^2 + \cdots + 0.251x_2^4 + 0.829x_2^2,$$

and

$$\sigma_1^* = 8.80 \times 10^{-13}\, x_1^5x_2 + 0.741x_1^2x_2^4 + \cdots + 2.27 \times 10^{-6}x_2^6,$$
$$\sigma_2^* = 1.80x_1^6 + 0.765\, x_1^4x_2^2 + \cdots - 2.95 \times 10^{-12}\, x_2^6,$$
$$\sigma_3^* = -1.763 \times 10^{-12}x_1^6 + 0.602x_1^4x_2^2 + \cdots - 5.15 \times 10^{-13}\, x_2^6.$$

Set $\tau = 10^{-2}$, and round the coefficients of V^* as rational numbers with the common denominator bound 100. We obtain the corresponding rational polynomial $V(x_1, x_2)$ and $\sigma_i(x_1, x_2), i = 1, 2, 3$:

$$\begin{aligned} V =& \frac{17}{76}\, x_1^4x_2^4 + \frac{15}{76}\, x_1^4x_2^2 + \frac{27}{76}\, x_1^4 + \frac{4}{19}\, x_1^3x_2^3 - \frac{15}{38}\, x_1^3x_2 \\ &+ \frac{15}{76}x_1^2x_2^4 + \frac{11}{19}\, x_1^2x_2^2 + \frac{63}{76}x_1^2 + \frac{15}{38}\, x_1x_2^3 + \frac{1}{4}x_2^4 + \frac{63}{76}x_2^2, \\ \sigma_1 =& \frac{14}{19}\, x_1^2x_2^4, \quad \sigma_2 = \frac{38}{29}\, x_1^4x_2^2, \quad \sigma_3 = \frac{23}{38}\, x_1^4x_2^2 + \frac{45}{38}x_1^2x_2^4. \end{aligned}$$

Now let us verify whether the polynomial

$$\frac{\mathbf{d}V}{dt} - \sigma_1(x_1, x_2)x_1 - \sigma_2(x_1, x_2)x_2 - \sigma_3(x_1, x_2)x_1x_2$$

is positive semidefinite. By running the algorithm in Section 3.3, we find that the above polynomial can be written as an SOS of 8 polynomials, and its SOS representation is listed in Appendix. With the state invariants $x_1 \geq 0 \wedge x_2 \geq 0$, it is obvious that $\frac{\mathbf{d}V}{dt}$ is always positive semidefinite. Therefore,

$$V(x_1, x_2) - d_0 \geq 0$$

with $d_0 = \min\{V(x_1, x_2) : (x_1, x_2) \in \Theta\}$, is an invariant of the given nonlinear system with the state invariants $x_1 \geq 0$ and $x_2 \geq 0$.

5. CONCLUSION

In this paper, we present a symbolic-numeric approach to compute inequality invariants of nonlinear continuous systems in hybrid systems. Employing SOS relaxation and rational vector recovery techniques, it can be guaranteed that an exact invariant, rather than a numerical one, can be obtained efficiently and practically. This approach avoids both the high complexity of symbolic invariant generation methods based on quantifier elimination, and the weakness of applying numerical approaches to verify correctness of hybrid systems.

Acknowledgments

We wish to thank Professor Lu Yang for his valuable suggestions on improving our two examples, and we also thank the reviewers for their helpful comments.

6. REFERENCES

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Their. Compute. Sci.*, 138:3–34, 1995.

[2] E. Asarin, T. Dang, and A. Girard. Reachability analysis of nonlinear systems using conservative approximation. In O. Maler and A. Pnueli, editors, *Hybrid Systems: Computation and Control*, volume 2623 of *Lecture Notes in Computer Science*, pages 20–35. Springer Berlin/Heidelberg, 2003.

[3] M. Branicky. General hybrid dynamical systems: Modeling, analysis, and control. In *Alur, R., Henzinger, T.A. Sontag, E.D., eds: Hybrid Systems, Volume 1066 of LNCS*, pages 186–200, 1995.

[4] E. R. Carbonell and A. Tiwari. Generating polynomial invariants for hybrid systems. In *HSCC, volume 3414 of LNCS*, pages 590–605, 2005.

[5] Y. Chen, B. Xia, L. Yang, and N. Zhan. Generating polynomial invariants with discoverer and qepcad. *LNCS*, 4700:67–82, 2007.

[6] G. E. Collins and H. Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *J. Symb. Compute.*, 12(3):299–328, 1991.

[7] M. Colón and H. Sipma. Synthesis of linear ranking functions. In *TACAS, volume 2031 of LNCS*, pages 67–81, 2001.

[8] M. A. Colón, S. Sankaranarayanan, and H. B. Sipma. Linear invariant generation using non-linear constraint solving. *Lecture Notes in Computer Science*, 2725:420–432, 2003.

[9] P. Cousot. Proving program invariance and termination by parametric abstraction, lagrangian relaxation and semidefinite programming. *LNCS*, 3385:1–24, 2005.

[10] J. Davoren and A. Nerode. Logics for hybrid systems. In *Proceedings of the IEEE*, volume 88, pages 985–1010, 2000.

[11] Z. Y. Erich Kaltofen, Bin Li and L. Zhi. Exact certification in global polynomial optimization via sums-of-squares of rational functions with rational coefficients. Accepted by Journal of Symbolic Computation, 2009.

[12] S. Gulwani and A. Tiwari. Constraint-based approach for hybrid systems. *LNCS*, 5123:190–203, 2008.

[13] T. Henzinger. The theory of hybrid automata. In *LICS, Los Alamitos*, pages 278–292. IEEE Computer Society, 1996.

[14] E. Kaltofen, B. Li, Z. Yang, and L. Zhi. Exact certification of global optimality of approximate

factorizations via rationalizing sums-of-squares with floating point scalars. In *Proceedings of the twenty-first international symposium on Symbolic and algebraic computation*, ISSAC '08, pages 155–164, New York, NY, USA, 2008. ACM.

[15] D. Kapur. Automatically generating loop invariants using quantifier elimination preliminary report-. In *IMACS Intl. Conf. on Applications of Computer Algebra*, 2004.

[16] L. Kovács. Reasoning algebraically about p-solvable loops. *Lecture Notes in Computer Science*, 4963:249–264, 2008.

[17] J. C. Lagarias. The computational complexity of simultaneous diophantine approximation problems. *SIAM J. Comp.*, 14:196–209, 1985.

[18] J. Löfberg. YALMIP: A toolbox for modeling and optimization in MATLAB. In *Proceedings of the CACSD*, Taipei, Taiwan, 2004. Available at `http://control.ee.ethz.ch/~joloef/yalmip.php`.

[19] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety.* Springer, New York, 1995.

[20] N. Matringe, A. V. Moura, and R. Rebiha. Morphisms for non-trivial non-linear invariant generation for algebraic hybrid systems. In *HSCC, volume 5469 of LNCS*, pages 445–449, 2009.

[21] A. Platzer. Differential-algebraic dynamic logic for differential-algebraic programs. *Journal of Logic and Computation*, 20:309–352, 2007.

[22] A. Platzer. Differential dynamic logic for verifying parametric hybrid systems. *LNAI*, 4548:216–232, 2007.

[23] A. Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reasoning Manuscript*, 41:143–189, 2008.

[24] A. Platzer and E. Clarke. Formal verification of curved flight collision avoidance maneuvers: A case study. In *Proceedings of Formal Methods*, pages 547–562, 2009.

[25] A. Platzer and E. M. Clarke. The image computation problem in hybrid systems model checking. In *Bemporad et al*, pages 473–486. Springer, 2007.

[26] A. Platzer and E. M. Clarke. Computing differential invariants of hybrid systems as fixedpoints. *Form. Methods Syst. Des.*, 35:98–120, August 2009.

[27] A. Platzer and J. Quesel. European train control system: A case study in formal verification. 5885:246–265, 2009.

[28] A. Podelski and S. Wagner. A sound and complete proof rule for region and stability of hybrid systems. In *HSCC, volume 4416 of LNCS*, pages 750–753, 2007.

[29] S. Prajna, A. Papachristodoulou, and P. Parrilo. SOSTOOLS: Sum of squares optimization toolbox for MATLAB, 2002. Available at `http://www.cds.caltech.edu/sostools`.

[30] E. Rodríguez-Carbonella and D. Kapur. Generating all polynomial invariants in simple loops. *Journal of Symbolic Computation*, 42:443–476, 2007.

[31] S. Sankaranarayanan. Automatic invariant generation for hybrid systems using ideal fixed points. In *ProcHSCC'10*, 2010.

[32] S. Sankaranarayanan, H. Sipma, and Z. Manna. Fixed point iteration for computing the time-elapse operator. In *HSCC, LNCS*. Springer, 2006.

[33] S. Sankaranarayanan, H. Sipma, and Z. Manna. Constructing invariants for hybrid systems. *Formal Methods in System Design*, 32:25–55, 2008.

[34] S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Non-linear loop invariant generation using Gröbner bases, 2004.

[35] J. F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11/12:625–653, 1999.

[36] A. Tiwari and G. Khanna. Nonlinear systems: Approximating reach sets. *Lecture Notes of Computer Science*, 2993:600–614, 2004.

[37] A. Tiwari, H. Ruess, H. Saïdi, and N. Shankar. A technique for invariant generation. In *TACAS 2001, vol. 2031 of LNCS*, pages 113–127. Springer-Verlag, 2001.

[38] B. Xia. Discoverer: a tool for solving semi-algebraic systems. *ACM Commun. Compute. Algebra*, 41(3):102–103, 2007.

Appendix

A solution to Example 1,

$$\mathbf{r} = \frac{1484}{1011} f_1^2 + \frac{285776}{193095} f_2^2 + \frac{11143727}{1016838} f_3^2 + \frac{1508987592}{30774371} f_4^2$$

where

$$f_1 = \tfrac{1011}{1484} x_2 - \tfrac{183}{2968} x_1 + \tfrac{255}{2968} x_2^2 + \tfrac{1209}{2968} x_1 x_2 + \tfrac{375}{2968} x_1^2,$$
$$f_2 = \tfrac{193095}{285776} x_1 + \tfrac{268305}{2000432} x_2^2 + \tfrac{126945}{285776} x_1 x_2 + \tfrac{194745}{2000432} x_1^2,$$
$$f_3 = \tfrac{1016838}{11143727} x_2^2 - \tfrac{4931}{454846} x_1 x_2 - \tfrac{1792057}{22287454} x_1^2$$
$$f_4 = \tfrac{30774371}{1508987592} x_1 x_2 - \tfrac{30774371}{1508987592} x_1^2.$$

□

A solution to Example 2,

$$\mathbf{r} = \frac{38}{15} f_1^2 + \frac{190}{147} f_2^2 + \frac{2793}{1010} f_3^2 + \frac{1919}{4172} f_4^2 + \frac{4756080}{7066109} f_5^2 + \frac{1074048568}{1371482325} f_6^2 + \frac{104232656700}{92227612763} f_7^2 + \frac{28037194279952}{24255149844713} f_8^2,$$

where

$$f_1 = \tfrac{15}{38} x_2^2 + \tfrac{3}{38} x_1 x_2 + \tfrac{3}{38} x_1^2 - \tfrac{2}{19} x_1^2 x_2 + \tfrac{15}{38} x_1 x_2^3 + \tfrac{4}{19} x_1^2 x_2^2 - \tfrac{3}{38} x_1^3 x_2 - \tfrac{3}{38} x_1^3 x_2^2,$$
$$f_2 = \tfrac{147}{190} x_1 x_2 + \tfrac{11}{95} x_1^2 + \tfrac{2}{19} x_1 x_2^2 - \tfrac{8}{95} x_1^2 x_2 - \tfrac{3}{190} x_1^2 x_2^2 - \tfrac{11}{95} x_1^3 x_2 + \tfrac{1}{38} x_1^2 x_2^3 + \tfrac{21}{380} x_1^3 x_2^2,$$
$$f_3 = \tfrac{1010}{2793} x_1^2 + \tfrac{250}{2793} x_1 x_2^2 + \tfrac{94}{2793} x_1^2 x_2 + \tfrac{563}{1862} x_1^2 x_2^2 - \tfrac{1010}{2793} x_1^3 x_2 - \tfrac{11}{2793} x_1^2 x_2^3 + \tfrac{1}{133} x_1^3 x_2^2,$$
$$f_4 = \tfrac{4172}{1919} x_1 x_2^2 - \tfrac{89}{3838} x_1^2 x_2 - \tfrac{1087}{3838} x_1^2 x_2^2 - \tfrac{17}{404} x_1^2 x_2^3 - \tfrac{119}{1919} x_1^3 x_2^2,$$
$$f_5 = \tfrac{7066109}{4756080} x_1^2 x_2 - \tfrac{1391293}{4756080} x_1^2 x_2^2 - \tfrac{175701}{3170720} x_1^2 x_2^3 - \tfrac{12287}{113240} x_1^3 x_2^2,$$
$$f_6 = \tfrac{1371482325}{1074048568} x_1^2 x_2^2 + \tfrac{23684501}{268512142} x_1^2 x_2^3 - \tfrac{4029727}{537024284} x_1^3 x_2^2,$$
$$f_7 = \tfrac{92227612763}{104232656700} x_1^2 x_2^3 - \tfrac{110608051}{208465313400} x_1^3 x_2^2,$$
$$f_8 = \tfrac{24255149844713}{28037194279952} x_1^3 x_2^2.$$

□

Univariate Polynomial Root-Finding by Arming with Constraints [*]

Victor Y. Pan
Department of Mathematics
and Computer Science
Lehman College of CUNY
Bronx, NY 10468 USA
victor.pan@lehman.cuny.edu
http://comet.lehman.cuny.edu/vpan/

ABSTRACT

Elimination methods are highly effective for the solution of linear and nonlinear systems of equations, but there are important examples where reversing the elimination idea can improve global convergence of iterative methods, that is their convergence from the start. We believe that these examples reveal a direction that the algorithm designers should explore systematically, and we show this principle at work for the approximation of a single root (zero) of a univariate polynomial of a degree n. We reduce the latter task to the solution of a multivariate polynomial system of n equations with n unknowns and achieve faster and more consistent global convergence of Newton's iteration applied to the system rather than a single equation. Global convergence behavior of our algorithms is similar to that of Durand–Kerner's (Weierstrass') iteration, but we direct convergence to a single root and use linear arithmetic time per iteration loop. Having m processors one can apply the algorithms concurrently, to approximate up to m roots within the same parallel time. Technically the computations boil down to solving Sylvester or generalized Sylvester linear systems of equations, linked to partial fraction decompositions. By solving these tasks efficiently, we arrive at effective algorithms for polynomial root-finding.

Categories and Subject Descriptors

F.2.1 [**Theory of Computation**]: Analysis of Algorithms and Problem ComplexityNumerical Algorithms and Problem

General Terms

Algorithms

[*]Supported by PSC CUNY Awards 62230–0040 and 63153–0041

Keywords

Computations on matrices, Computations on polynomials, Polynomial root-finding and factorization, Arming with constraints, Convolution, Sylvester matrices, PFD

1. INTRODUCTION

1.1 Principle of Arming with Constraints for polynomial root-finding and beyond

Elimination of variables is a fundamental technique, employed by Gauss, Macaulay and more recently Buchberger for the solution of linear and polynomial systems of equations. This is a natural, popular and highly successful tool, but we are going to demonstrate some benefits of reversing the elimination principle for the classical problem of univariate polynomial root-finding (see McNamee (1993, 1997, 2002 and 2007)).

Its study has begun at least four millennia ago and still remains a highly important subject of modern computations because of its applications to algebraic and geometric computations and signal processing. The algorithms of Pan (1995, 1996, 2001a, and 2002) solve both problems (as well as the related problems of polynomial factorization and root isolation) by using arithmetic and Boolean time which is optimal up to polylogarithmic factors in the input size under both sequential and parallel models of computing. The algorithms extend the work of Schönhage (1982), whose solution was slower by order of magnitude.

The cited papers of both authors employ Newton's iteration and invest substantial and relatively costly effort to ensure its global convergence (that is convergence from the start) for the worst case input. The users, however, skip this initial stage because empirically convergence of Newton's as well as some other popular iterations is rather satisfactory under various heuristic initialization policies. In this paper we concur with users' preference and focus on refining Newton's iteration towards improving its convergence power.

We first recall the extension of Schönhage's algorithms in Kirrinnis (1998), based on approximating and refining the associated partial fraction decompositions and, as in Schönhage (1982), directed to factorization of a degree n polynomial into the product of n linear factors. Hereafter we will use the acronym "*PFD*" for "partial fraction decomposition".

Such a factorization is an important goal in its own right due to its applications to the time series analysis, Weiner filtering, noise variance estimation, covariance matrix computation, and the study of multi-channel systems (see Wilson (1969), Box and Jenkins (1976), Barnett (1983), Demeure and Mullis (1989 and 1990), Van Dooren (1994)).

We can see a major advantage of approximating such a factorization rather than a single root because of implicit engagement of n independent equations versus the single polynomial equation $p(x) = 0$.

Indeed Newton's iteration

$$x_{i+1} = x_i - \frac{p(x_i)}{p'(x_i)} \tag{1.1}$$

as well as other known iterations for a single univariate equation readily lose their direction to a root unless the current approximation is already close to it. With additional equations we can expect to arrive at a larger system, which is more stable and cannot be so easily pushed astray from its trajectory.

This expectation has been consistently confirmed by the statistics of global convergence of univariate polynomial root-finders based on Viète's (Vieta's) equations such as Durand–Kerner iteration (traced back to Weierstrass (1903) and hereafter referred to as the *WDK iteration*) and Ehrlich–Aberth iteration (also called Aberth's) as well as matrix methods involving eigenvectors and thereby again reduced to systems of n equations for their coordinates (see Pan and Zheng (2011) and the bibliography therein on these methods).

These and other data and observations suggest conjecting the following general *Principle of Arming with Constraints*, for which we will use the acronym *PAC: one can strengthen global convergence of iterative root-finders by competently immersing the input equations into a larger system of equations.*

This principle is well reflected in the proverbs "One's as good as none", "There's strength in numbers", "One man does not make a team", "Odin v pole ne voin" (Russian) ("One man does not make an army"), "Unus homo non facit choream "(medieval Latin) ("One man does not make a dance ensemble"), "One swallow does not a summer make" (the Aristotelian), but seems to have never been stated formally in mathematics or computational sciences.

In addition to the cited support for some ad hoc applications of this principle to polynomial root-finding, one can recall similar statistics for the SNTLN techniques of Park, Zhang and Rosen (1999), Rosen, Park and Glick (1996 and 1999) and the efficiency of application of dual linear and nonlinear programming.

Our present work should motivate systematic exploration of the PAC. For yet another demonstration of its power we substantially advance Newton's approximation of a single root of a polynomial equation by redefining the task via n constraints.

1.2 Polynomial root-finding for a single root and many roots

We link Kirrinnis' algorithm to the WDK iteration, which empirically has strong global convergence. Both algorithms have been devised for the approximation of all n zeros of a polynomial $p(x)$ of a degree n, but can we employ them for approximating a single zero?

We do not mind to approximate the other $n-1$ zeros as by-product, but the arithmetic cost per an iteration loop increases from linear in (1.1) to nonlinear in Kirrinnis (1998) and quadratic in the WDK iteration, and both of the latter algorithms assume some additional initial information besides a single approximate root (zero).

Fortunately we readily fix these discrepancies by computing the solutions and least squares solutions of the associated structured linear systems of equations. Our *Laser Versions* of WDK and Kirrinnis' algorithms approximate a single root by using linear arithmetic time per iteration loop and preserve their fast and consistent global convergence. Surely one can concurrently apply our Laser WDK root-finder on m processors to approximate up to m roots for $1 < m \leq n$.

Devising our algorithms we follow the PAC: we employ the factorization equation $p = L_1 \cdots L_m$, which imposes n constraints on the coefficients and for $m = 2$ enables us to recover the factor L_1 where we are given L_2 and vice versa. The factor can be recovered as a rather noncostly least squares solution of the associated structured linear system of equations, but we further simplify this stage in Section 8.

Every step of Newton's iteration for the associated multivariate polynomial system of equations boils down to the solution of a Sylvester linear system of equations, and we exploit their structure and their association with PFDs to solve these systems efficiently.

1.3 Structured linear systems of equations and PFDs

Our association of Sylvester and generalized Sylvester linear systems of equations with PFDs immediately implies extension of the celebrated formula of Gohberg and Semencul (1972) from Toeplitz to Sylvester and generalized Sylvester matrix inverses.

Furthermore, we solve a nonsingular Sylvester linear system of n equations and compute the associated PFD by using a linear number of arithmetic operations $O(n)$ in the important case where one of the two polynomials defining the matrix of this system has a constant degree.

We extend our algorithm to generalized Sylvester linear systems, defined by a PFD for the reciprocal of a polynomial factorized into the product of m nonconstant factors L_1, $\dots, L_m$ provided

$$n - \deg L_m = O(1)\ . \tag{1.2}$$

In the important case of two basic polynomials one of which is linear, the associated Sylvester matrix turns into a scaled and shifted companion matrix. In this case we solve the associated Sylvester linear system of n equations with Gaussian elimination by using $6n - 5$ arithmetic operations.

Furthermore, we follow Pan and Zheng (2011) and numerically stabilize the elimination stage of this solution where $p = L_1L_2$ and one of the factors is linear.

1.4 Organization of the paper and further research

We organize our presentation as follows. Having introduced definitions in the next section, we recall Kirrinnis' expressions for Newton's updates of polynomial factorization and PFD in Section 3. We relate PFDs and the solution of Sylvester linear systems of equations in Section 4. In Sections 5 and 6 we present their fast solution under (1.2) and in Section 7 recall some initialization policies for root-finding. In Sections 8 and 10 we compute approximate solutions of a Cauchy convolution linear system and apply them in our root-finding construction. In Section 9 we recall some details of Kirrinnis' algorithm and link it to the WDK iteration. Section 11 is left for a summary and discussion.

We plan further theoretical and experimental study of our approach and its extensions, in particular further applications of the PAC.

2. DEFINITIONS AND PRELIMINARIES

We use the acronyms "PFD" for "partial fraction decomposition, "WDK" for " Weierstrass–Durand–Kerner", "GCD" for "greatest common divisor". We use the abbreviation "op" for "arithmetic operation".

We represent a polynomial $u(x)$ with its coefficient vector $\mathbf{u}$ and write u to denote such a polynomial wherever this causes no confusion.

$\deg u$ denotes its degree.

$\gcd(u, v)$ denotes the monic GCD of two polynomials $u = u(x)$ and $v = v(x)$.

p denotes a monic polynomial

$$p = p(x) = \sum_{i=0}^{n} p_i x^i = p_n \prod_{j=1}^{n} (x - z_j) \ , \quad p_n = 1 \ . \qquad (2.1)$$

p_{rev} denotes the reverse polynomial

$$p_{\text{rev}} = \sum_{i=n}^{n} p_{n-i} y^i = p_n \prod_{j=1}^{n} (z_j y - 1) \ , \quad p_n = 1 \ . \qquad (2.2)$$

We write

$$V_L = V \mod L$$

for two polynomials L and V.

Given five polynomials $F = F(x)$, $D = D(x)$, $N = N(x)$, $L = L(x)$ and $G = G(x)$ such that

$$FD = N + LG \ , \ \deg F < \deg L \ ,$$

and D and L are coprime, we write

$$F = (N/D) \mod L$$

and then observe that

$$F = (N_L/D_L) \mod L \ ,$$

$$(N(x)/D(x)) \mod (x - z) = N(z)/D(z) \ , \qquad (2.3)$$

and if $N = QL + R$ and $\deg R < \deg L$, then

$$F = Q_L + R/D_L \mod L \ .$$

FACT 2.1. *One can multiply and divide a pair of polynomials of degrees at most n by using $O(n \log n)$ ops.*

PROOF. See, e.g. Pan (2001a), Sections 2.4 and 2.5. □

FACT 2.2. *Given three polynomials N, D, and L where D and L are coprime, we can compute the polynomial*

$$F = (N_L/D_L) \mod L$$

by using $O(k \log^2 k + n \log n)$ ops provided $k = \deg L$ and $n = \max\{\deg D, \deg N\}$.

PROOF. Fact 2.2 follows from Fact 2.1 and Pan (2001a), Theorem 2.9.3. □

M^T denotes the transpose of a matrix or vector M; M^H is its Hermitian transpose.

$[M_1 \mid M_2 \mid \ \ldots \ \mid M_s]$ is a $1 \times s$ block matrix with the blocks $M_1, \ldots, M_s$.

$\text{diag}(M_1, M_2, \ldots, M_s)$ is an $s \times s$ block diagonal matrix with the diagonal blocks $M_1, \ldots, M_s$.

$I = I_n = [\mathbf{e}_1 \mid \ \ldots \ \mid \mathbf{e}_n]$ denotes the $n \times n$ identity matrix, having the coordinate vectors $\mathbf{e}_1, \ldots, \mathbf{e}_n$ as the columns.

T is a Toeplitz matrix if its entries are invariant in their shift into the diagonal direction, that is if $T = [t_{i-j}]_{i,j}$.

$$C_k(w) = \begin{pmatrix} w_l & & & O \\ \vdots & \ddots & & \\ \vdots & \ddots & \ddots & \\ \vdots & \ddots & \ddots & w_l \\ w_0 & \vdots & \ddots & \vdots \\ & \ddots & \vdots & \\ & & \ddots & \vdots \\ O & & & w_0 \end{pmatrix}$$

denotes the kth Cauchy convolution matrix of a polynomial $w(x) = \sum_{i=0}^{l} w_i x^i$, that is the $(k + l + 1) \times (k + 1)$ Toeplitz matrix defined by its first row $[w_l \mid 0 \mid \ \ldots \ \mid 0]$ and its first column $[w_l \mid w_{l-1} \mid \ \ldots \ \mid w_0 \mid 0 \mid \ \ldots \ \mid 0]^T$.

Note that $C_0(w) = \mathbf{w}$ is a vector.

Assume a factorization

$$L = L_1 \cdots L_m \approx p \ ,$$

define the polynomials

$$Q_i = L/L_j, \ j = 1, \ldots, m,$$

and the $1 \times m$ block matrix

$$S(Q_1, \ldots, Q_m) = [C_{n-d_1}(Q_1) \mid \ \ldots \ \mid C_{n-d_m}(Q_m)],$$

call it *generalized Sylvester matrix* and call $L_1,\ldots,L_m$ its *basic polynomials.* For $m = 2$ we arrive at a *Sylvester matrix*

$$S(Q_1, Q_2) = S(L_2, L_1) \ .$$

We call a generalized Sylvester matrix $S(Q_1, \ldots, Q_m)$ and a linear system of equations with such a coefficient matrix *regular* if the m basic polynomials $L_1, \ldots, L_m$ are pairwise coprime.

FACT 2.3. *A square Sylvester matrix $S(L_2, L_1)$ is nonsingular if and only if its basic polynomials L_1 and L_2 are coprime.*

3. KIRRINNIS–NEWTON'S UPDATES OF FACTORIZATION AND PFD

Assume a factorization

$$p = L_1 \cdots L_m$$

of a polynomial p of (2.1) and sufficiently close initial approximations

$$l_j^{(0)} \approx L_j \text{ for } j = 1, \ldots, m$$

and recursively improve them by applying Newton's iteration. In the kth iteration loop for $k = 0, 1, \ldots$ compute at first the polynomial

$$l^{(k)} = \prod_{j=1}^{m} l_j^{(k)} \approx p \ , \tag{3.1}$$

then Newton's corrections $\Delta_1^{(k)}, \ldots, \Delta_m^{(k)}$ defined by the PFD

$$\frac{p - l^{(k)}}{l^{(k)}} = \frac{\Delta_1^{(k)}}{l_1^{(k)}} + \cdots + \frac{\Delta_m^{(k)}}{l_m^{(k)}}, \ \deg \ \Delta_j^{(k)} < \deg \ l_j^{(k)} \ , \tag{3.2}$$

$j = 1, \ldots, m$, and finally the improved approximations

$$l_j^{(k+1)} = l_j^{(k)} + \Delta_j^{(k)}, \ j = 1, \ldots, m \ . \tag{3.3}$$

4. A PFD AND SYLVESTER LINEAR SOLVING

Our next goal is the computation of the corrections $\Delta_j^{(k)}$ of (3.2). Let us write

$$T = p - l^{(k)}, \ L_j = l_j^{(k)}, \ V_j = \Delta_j^{(k)}, \ d_j = \deg \ L_j \tag{4.1}$$

for $j = 1, \ldots, m$. Now for m pairwise prime monic polynomials $L_1, \ldots, L_m$, their product L, and a polynomial $T = T(x)$ of a degree at most $n-1$ and coprime with p, define a unique PFD

$$\frac{T}{L} = \frac{V_1}{L_1} + \cdots + \frac{V_m}{L_m} \ , \ L = L_1 \cdots L_m \tag{4.2}$$

by m polynomials $V_1, \ldots, V_m$ such that

$$\deg \ V_j < \deg \ L_j = d_j, \ j = 1, \ldots, m$$

where

$$2 \le m \le d_1 + \ldots + d_m = n \ .$$

Multiply this PFD by L and obtain an equivalent polynomial equation

$$Q_1 V_1 + \cdots + Q_m V_m = T \tag{4.3}$$

where

$$Q_j = \frac{L}{L_j} \ , \ V_j = \frac{T L_j}{L} \mod L_j \ , \ j = 1, \ldots, m \ . \tag{4.4}$$

Alternatively the coefficient vectors of the polynomials $V_1, \ldots, V_m$ can be obtained from a linear system of equations

$$S(Q_1, \ldots, Q_m)\mathbf{y} = \mathbf{T} \ . \tag{4.5}$$

Here

$$\mathbf{y}^T = [\mathbf{V}_1^T \mid \ \ldots \ \mid \mathbf{V}_m^T] \ ,$$

$\mathbf{V}_j$ denote the coefficient vectors of the polynomials V_j, $j = 1, \ldots, m$;

$$S(Q_1, \ldots, Q_m) = [C_{d_1-1}(Q_1) \mid \ \ldots \ \mid C_{d_m-1}(Q_m)]$$

is the generalized Sylvester matrix defined by its m basic polynomials $L_1, \ldots, L_m$, and $\mathbf{T}$ is the coefficient vector of a polynomial T; in particular

$$\mathbf{T} = \mathbf{e}_n = [0 \mid \ldots \ \mid 0 \mid 1]^T$$

is the nth coordinate vector of dimension n in the case where $T = 1$.

5. SOLUTION OF A SYLVESTER LINEAR SYSTEM OF EQUATIONS VIA PFD

5.1 Extension of Gohberg–Semencul formula

THEOREM 5.1. *Let $S(Q_1, \ldots, Q_m)$ denote an $n \times n$ regular generalized Sylvester matrix. Then for any right hand side vector $\mathbf{T}$ the generalized Sylvester linear system of equations (4.5) can be solved in $O((n \log n) \log m)$ ops provided its solution is known in the case where $\mathbf{T} = (0, \ldots, 0, 1)^T$ is the coefficient vector of the constant polynomial $T = 1$.*

PROOF. The theorem follows from equations (4.4) and Fact 2.1. □

This simple theorem extends the celebrated result in Gohberg and Semencul (1972) from Toeplitz to generalized Sylvester matrices.

5.2 The case where all but one basic polynomials have small degrees

According to the two following theorems we can solve a regular generalized Sylvester linear system of n equations (4.5) for any right hand side vector $\mathbf{T}$ by using $O(n)$ ops provided that equation (1.2) holds or equivalently that $m = O(1)$ and the factors $L_1, \ldots, L_{m-1}$ have degrees of $O(1)$.

We prove the theorems constructively by describing two supporting algorithms. The first theorem covers the simpler but important case where $m = 2$.

THEOREM 5.2. *Suppose $S = S(L_2, L_1)$ denotes the $n \times n$ Sylvester matrix defined by two coprime polynomials L_1 and L_2 such that*

$$d_1 = \deg L_1 = O(1) \text{ and } d_2 = \deg L_2 = n - d_1 \ .$$

Then a Sylvester linear system $S\mathbf{y} = \mathbf{T}$ of n equations with this matrix can be solved by using $O(n)$ ops.

PROOF. Here is our algorithm supporting the theorem. One can immediately verify that the algorithm is correct and uses $O(n)$ ops (cf. Facts 2.1 and 2.2).

ALGORITHM 5.1. **Sylvester Solving via PFD**

INPUT: *the coefficient vectors $\mathbf{L_1}$, $\mathbf{L_2}$ and $\mathbf{T}$ of two coprime polynomials, L_1 and L_2 and a polynomial T such that*

$$d_1 = \deg L_1 = O(1) \text{ and } degT < \deg(L_1 L_2) = d_1 + d_2 \ .$$

COMPUTATIONS. *Successively compute the polynomials*

1. $L = L_1 L_2$,
2. $V_1 = (T L_1 / L) \mod L_1$ *(cf. (4.4)),*
3. $U = T - V_1 L_2$ *and*
4. $V_2 = U / L_1$ *(cf. (4.3)).*

OUTPUT *the vector*

$$\mathbf{y} = [\mathbf{V}_1^T \mid \mathbf{V}_2^T]^T$$

satisfying

$$S(L_2, L_1)\mathbf{y} = \mathbf{T} \ .$$

Here $\mathbf{V}_j$ denote the coefficient vectors of the polynomials V_j for $j = 1, 2$.

□

THEOREM 5.3. *Suppose $S = S(Q_1, \dots, Q_m)$ is an $n \times n$ regular generalized Sylvester matrix defined by m pairwise coprime polynomials $L_1, \dots, L_m$ such that*

$$Q_j = \frac{p}{L_j} \ , \ L = \prod_{j=1}^{m} L_j \ , \ \deg L = n \ , \ \deg L_j = d_j \quad (5.1)$$

for $j = 1, \dots, m$ and constants $m, d_1, \dots, d_{m-1}$ of $O(1)$. Then a generalized Sylvester linear system $S\mathbf{y} = \mathbf{T}$ of n equations with this matrix and any vector T can be solved by using $O(n)$ arithmetic operations.

PROOF. Here is our algorithm supporting the theorem.

ALGORITHM 5.2. **Generalized Sylvester Solving via PFD**
INPUT: *m polynomials $Q_1, \dots, Q_m$ defined via (4.2) and (4.4) by m basic pairwise coprime polynomials $L_1, \dots,$ and a polynomial T of a degree less than* $\deg(L_1 \cdots L_m)$.

COMPUTATIONS. *Successively compute the polynomials*

1. $g_{1,m} = \gcd(Q_1, Q_m)$,
2. $L_1 = Q_m / g_{1,m}$,
3. $L = L_1 Q_1$,
4. $L_j = L/Q_j$ *for $j = 2, \dots, m$ (cf. (4.4)),*
5. $V_j = (TL_j/L) \mod L_j$ *for $j = 1, \dots, m-1$ (cf. (4.4)),*
6. $W_m = T - \sum_{j=1}^{m-1} Q_j V_j$ *and*
7. $V_m = W_m / Q_m$.

OUTPUT *the vector $\mathbf{y} = [\mathbf{V}_1^T \mid \dots \mid \mathbf{V}_m^T]^T$ satisfying (4.5).*

To verify correctness of steps 1 and 2 of the algorithm observe that

$$Q_1 = L_2 \cdots L_m \text{ and } Q_m = L_1 \cdots L_{m-1} \ ,$$

so that

$$g_{1,m} = L_2 \cdots L_{m-1} \text{ and } Q_m / g_{1,m} = L_1$$

because the polynomials $L_1, \dots, L_m$ are assumed to be pairwise coprime. Correctness of steps 4 and 5 follows from equations (4.4). Correctness of step 7 follows from equation (4.3).

The bound $O(n)$ on the number of arithmetic operations involved follows from Facts 2.1 and 2.2 because both m and

$$\deg Q_m = n - d_m = \sum_{j=1}^{m-1} d_j$$

are in $O(1)$. ☐

5.3 All but one basic polynomials are linear

Assume that

$$L_j = x - z_j$$

are monic linear factors of p for $j = 1, \dots, m-1$. Then

$$V_j = T(z_j)/Q_j(z_j) \text{ for } j = 1, \dots, m-1$$

(cf. (4.4) and (2.3)), and we can simplify the above computations based on (2.3) as follows.

ALGORITHM 5.3. **Generalized Sylvester Solving via PFD where all but one basic polynomials are linear**
INPUT *and* OUTPUT *as in Algorithm 5.2 where*

$$\deg Q_m = n - m + 1$$

or equivalently

$$\deg Q_j = 1 \text{ for } j = 1, \dots, m-1 \ .$$

COMPUTATIONS. *Successively compute the following scalars and polynomials,*

1. $Q_j(z_j)$ *and* $T(z_j)$ *for* $j = 1, \dots, m-1$,
2. $V_j = T(z_j)/Q_j(z_j)$ *for* $j = 1, \dots, m-1$,
3. $W_m = T - \sum_{j=1}^{m-1} V_j Q_j$ *and*
4. $V_m = W_m / Q_m$.

6. SOLUTION BY MATRIX METHODS

Alternatively we can solve a generalized Sylvester linear system (4.3) by matrix methods. We will specify the simpler cases where all but one basic polynomials are linear or $m = 2$.

6.1 All but one basic polynomials are linear

As in Section 5.3 assume that

$$L_j = x - z_j$$

are monic linear factors of p for $j = 1, \dots, m-1$, and so

$$Q_m = \prod_{j=1}^{m-1} (x - z_j) \ .$$

Then we have

$$S(Q_1, \dots, Q_m) = [Q \mid C_{n-m+1}(Q_m)]$$

where $C_{n-m+1}(Q_m)$ is the $(n-m+1)$st Cauchy convolution matrix of the polynomial Q_m and Q is the $n \times (m-1)$ matrix whose jth column is the coefficient vector of the polynomial Q_j for $j = 1, \dots, m-1$.

Interchange columns of $S(Q_1, \dots, Q_m)$ to obtain the matrix

$$[C_{n-m+1}(Q_m) \mid Q]$$

and apply either Gaussian elimination without pivoting or block cyclic reduction algorithms (cf. Golub and Van Loan (1996)). In both cases the computations take $O(n)$ ops if $m = O(1)$; watch for numerical problems and counter them with various heuristic tricks.

6.2 Two basic polynomials, one linear

Let us specify the above solution in the important case of the Sylvester linear system, where

$$m = 2, \ Q_1 = L_2 \text{ and } Q_2 = L_1 \ ;$$

furthermore assume that

$$L_2 = x - z_1$$

and the polynomials

$$L_1 = \sum_{i=0}^{n-1} l_i x^i$$

and L_2 are coprime. Then Algorithm 5.3 uses $8n-8$ ops, whereas the Sylvester matrix

$$S(Q_1, Q_2) = S(L_2, L_1)$$

turns into a scaled shifted companion matrix

$$\begin{pmatrix} 1 & & & & O & l_{n-1} \\ -z_1 & 1 & & & & l_{n-2} \\ & -z_1 & 1 & & & l_{n-3} \\ & & \ddots & \ddots & & \vdots \\ & & & \ddots & 1 & l_1 \\ & & & & -z_1 & l_0 \end{pmatrix} .$$

In this case Gaussian elimination with no pivoting solves the Sylvester linear system $S(L_2, L_1)\mathbf{y} = \mathbf{T}$ for any vector $\mathbf{T}$ by using $3n-3$ subtractions, $2n-2$ multiplications and n divisions.

One can replace Gaussian elimination with the cyclic reduction algorithm (see Golub and Van Loan (1996), Section 4.5.4, and references therein), which supports parallel acceleration to logarithmic time.

The elimination stage of Gaussian elimination is numerically stable where $|z_1| \leq 1$. For $|z_1| > 1$ and large n the $(n-1) \times (n-1)$ leading principal block of the matrix $S(L_2, L_1)$ is ill conditioned, but we can avoid numerical problems by working with the reverse polynomials

$$p_{\rm rev},\ T_{\rm rev},\ L_{2,{\rm rev}} = z_1 x - 1,\ L_{1,{\rm rev}},\ V_{1,{\rm rev}},\ V_{2,{\rm rev}} = V_2\ ,$$

defined in equation (2.2) (see Pan and Zheng (2011), Section 6 on such techniques of numeriical stabilization).

Under these reversions we can readily compute the solution by means of Gaussian elimination at the same arithmetic cost but with numerically stable elimination stage, and then we can output the constant $V_2 = V_{2,{\rm rev}}$ and the polynomial $V_1 = (V_{1,{\rm rev}})_{\rm rev}$.

We can ensure diagonal dominance of the matrix $S(L_2, L_1)$ by scaling the variable x, although scaling can make the coefficients of L_2 very uneven in magnitude.

In the case where $|z_1| > 1$ we have another simple recipe: cyclically interchange the rows of the linear system and then apply Gaussian elimination with no pivoting to the resulting linear system with the matrix

$$\begin{pmatrix} -z_1 & 1 & & & & O & l_0 \\ & -z_1 & 1 & & & & l_{n-1} \\ & & -z_1 & 1 & & & l_{n-2} \\ & & & \ddots & \ddots & & \vdots \\ & & & & -z_1 & 1 & l_2 \\ 1 & & & & & & l_1 \end{pmatrix} .$$

Finally we can compute the value $V_1 = \frac{T(z_1)}{L_2(z_1)}$ by using $4n-4$ ops (cf. Stage 2 of Algorithm 5.3) and then simplify the Sylvester linear system by substituting this variable. The substitution takes $2n-2$ ops, and the system turns into a bidiagonal Toeplitz linear system of $n-1$ equations, which we can solve in $2n-3$ ops. Overall we need $8n-9$ ops. This is slightly more than for the direct solution of the Sylvester linear system, but numerical stability may improve.

6.3 The case of any pair of basic polynomials

Keep assuming that

$$m = 2,\ Q_1 = L_2 \text{ and } Q_2 = L_1\ ,$$

but now assume a nonlinear factor L_2, that is, let

$$d_2 = \deg L_2 > 1 \text{ and } d_1 = \deg L_1 = n - d_2 < n-1.$$

Then we have

$$S(L_2, L_1) = [C_{d_2}(L_1) \mid C_{d_1}(L_2)]$$

where $C_{d_1}(L_2)$ is an $n \times n$ banded Toeplitz matrix having bandwidth $d_2 + 1$.

In this case the algorithm in Pan (1996a) enables us to solve a linear system

$$S(L_2, L_1)\mathbf{y} = \mathbf{T}$$

for any right hand side vector $\mathbf{T}$ by using $O(n \log d_2)$ ops, that is $O(n)$ ops in the case of a constant d_2.

By scaling the variable x we can ensure sufficiently strong domination of the leading coefficient of the polynomial L_2 over the other coefficients, and then we would have strongly diagonally dominant matrix $S(L_2, L_1)$, and consequently we would have strong numerical stability of Gaussian elimination and block cyclic reduction. If the leading coefficient of p is dominated by the trailing one, then we can shift to the reverse polynomial $p_{\rm rev}$ and use a smaller scaling factor.

6.4 The general case of variable leading coefficients

In numerical factorization of a polynomial and in numerical computation of its zeros it is natural to assume nonmonic input polynomials p and in particular to seek the factorization

$$p = \prod_{j=1}^{n} (u_j x - v_j) \qquad (6.1)$$

where for every j we have $u_j = 1$ or $v_j = 1$.

Then it is also natural to assume that the leading coefficients of the approximate factors (that is basic polynomials of the associated generalized Sylvester matrix) are variables, so that the factorization $p \approx L = L_1 \cdots L_m$ defines an $(n+1) \times (n+m)$ generalized Sylvester matrix and a system of $n+1$ equations with $n+m$ unknowns with this matrix.

To simplify our computations we wish to deal with square generalized Sylvester matrices. So far we achieved this by chosing leading coefficents of all polynomials $L, L_1, \ldots, L_m$ equal to one, but this is in conflict with factorization (6.1).

We can reconcile the assignment of the leading coefficients of the polynomials $L_1, \ldots, L_m$ with factorization (6.1), by fixing these coefficients for the polynomials $L_2, \ldots, L_m$ but varying the one of the polynomials L_1. This would lead us to solving a linear system with $(n+1) \times (n+1)$ generalized Sylvester matrix in every Newton's step. Our previous study is immediately extended to this case.

7. SOME INITIALIZATION POLICIES

So far we assumed that some crude or good initial approximations to the factors $L_1, \ldots, L_m$ have been supplied from outside by an initialization algorithm. E.g., a root-finder can supply approximate zeros $z_j^{(0)}$ and thus supply approximate linear factors of p equal to either

$$x - z_j^{(0)} \text{ where } |z_j^{(0)}| \leq 1$$

or

$$1 - x/z_j^{(0)} \text{ where } |z_j^{(0)}| \geq 1\ ,$$

$j = 1, \dots, m$. In this case our algorithms work just as *root-refiners.*

We can extend the algorithms to the case where $m = 2$ and where we are given only a single approximate factor $l_1^{(0)}$, e.g., $x - z_1^{(0)}$. Then we can compute the second approximate factor $y = l_2^{(0)}$ as an approximate solution of the convolution equation

$$l_1^{(0)} y \approx p \ ;$$

we detail this computation in the next section.

The iteration would recursively update both factors $l_1^{(k)} = x - z_1^{(k)}$ of degree one and $l_2^{(k)}$ of degree $n-1$ for $k = 0, 1, \dots$. We need to update the factor $l_2^{(k)}$ to support improvement of the factorization $l_1^{(k)} l_2^{(k)} \approx p$ even if we are only interested in approximating the zero z_1 of p.

We have consistently observed reasonably good global convergence of our algorithms in the previous sections when we tested them under the latter initialization and updating policies and the standard heuristic choices of random initial values of approximate zeros of p near the origin, near the center of gravity $-p_{n-1}/(np_n)$ of the n zeros, and on a circle

$$\{x : \ |x| = R\} \text{ for } R \geq c \max_{i>1} |p_{n-i}/p_n|$$

and a moderately large scalar c, say $c = 10$ or even $c = 2$.

To increase our chances for fast convergence we can choose up to n or even more than n initial approximations. Habbard, Schleicher and Sutherland (2001) proved that Newton's classical iteration (1.1) is expected to converge fast to all zeros of p provided order $n \log n$ random initial points on a large circle have been selected.

Towards further empirical improvement, one can employ the initial approximations on Bini's circles in Bini (1996) and Bini and Fiorentino (2000).

These choices naturally lead us to parallel algorithms for the approximation of all zeros of p or of a large fraction of them; in the latter case we can deflate the polynomial and reapply the algorithm.

8. APPROXIMATE SOLUTION OF THE CONVOLUTION EQUATION

The convolution equation

$$l_1^{(0)} l_2^{(0)} \approx p$$

is equivalent to the linear system

$$C_{n-d_1}(l_1^{(0)}) \mathbf{l}_2^{(0)} \approx \mathbf{p} \ . \tag{8.1}$$

Given polynomials p and $l_1^{(0)}$ we can compute an approximate factor

$$y(x) = l_2^{(0)}(x)$$

as the quotient of polynomial division

$$p = l_1^{(0)} y + r, \ \deg r < \deg l_1^{(0)} \ . \tag{8.2}$$

This computation takes $2n - 3$ ops where $l_1^{(0)} = x - z_1^{(0)}$ is a monic linear approximate factor of p, $z_1^{(0)} \approx z_1$ and $p(z_1) = 0$; the computation is numerically stable where $|z_1^{(0)}| \leq 1$, but if $|z_1^{(0)}| > 1$ we just need to shift to the reverse polynomial p_{rev}. We can strengthen numerical stability if we can afford scaling the variable x.

At a little higher arithmetic cost we can compute the coefficient vector of such an approximate factor

$$\mathbf{y} = \mathbf{l}_2^{(0)}$$

as a least squares solution to the linear system

$$C_{n-d_1}(l_1^{(0)}) \mathbf{y} \approx \mathbf{p}$$

under the additional constraint that the polynomial y is monic, that is the first coordinate of the vector $\mathbf{y}$ equals one; under this constraint we deduce PFD (4.2).

REMARK 8.1. *We can remove this constraint by allowing to vary the leading coefficients of the factors $l_2^{(k)}$ for $k = 0, 1, \dots$. Then every Newton's updating step would amount to the solution of a Sylvester linear system of $n+1$ equations with $n+1$ unknowns, and the residuals $T = p - l^{(k)}$ would generally have degree n (cf. Section 6.4). Our previous and subsequent study can be immediately extended to this case.*

Let us examine the least squares computation. We can apply QR factorization based on Givens rotations or the method of normal equations (see Golub and Van Loan (1996), Sections 5.3.2 and 5.3.3).

The latter method requires $O(n \min\{d_1, \log n\})$ ops in the case of structured linear systems (8.1) (see Corless et al. (1995)) and allows parallel acceleration to polylogarithmic time based on application of cyclic reduction.

The former method has stronger numerical stability; it uses $O(nd_1)$ ops and involves computation of square roots. One can employ the fast Givens transformations in Golub and Van Loan (1996), Section 5.1.13 to accelerate the algorithm (in particular by removing most of the computations of square roots); this weakens numerical stability but in a controlled way.

We can skip normalization and completely avoid computing square roots; this causes almost no penalty in the most important case where $d_1 = 1$, $l_1^{(0)} = x - z$ and the Cauchy convolution matrix

$$C_{n-1}(x-z) = \begin{pmatrix} 1 & & & O \\ -z & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ O & & & -z \end{pmatrix}$$

is bidiagonal. Here we write

$$z = z_1^{(0)}$$

to simplify the notation. Let us specify our computations where we choose monic polynomial $l_2^{(0)}$ and eliminate its leading coefficient from the linear system (8.1). After the elimination, this linear system turns into the following system of n linear equations with $n-1$ unknowns,

$$C_{n-2}(x-z) \bar{\mathbf{l}}_2^{(0)} = \bar{\mathbf{p}} \ .$$

Here $\bar{l}_2^{(0)} = l_2^{(0)} - x^{n-1}$ and $\bar{p} = p - x^{n-1}(x-z)$.

We first define the scaled Givens matrices

$$G_j = \text{diag}(I_j, G, I_{n-j-2}) \text{ for } G = \begin{pmatrix} 1 & -z \\ z & 1 \end{pmatrix}, \ j = 0, \dots, n-2$$

and write

$$W = \prod_{j=0}^{n-2} G_j \ .$$

By induction we readily verify the following facts.

FACT 8.1. *The matrix product $W = \prod_{j=0}^{n-2} G_j$ is the Toeplitz matrix defined by its first row $[1 \mid -z \mid 0 \mid \ldots \mid 0]$ and its first column $[1 \mid z \mid z^2 \mid \ldots \mid z^{n-2}]^T$.*

FACT 8.2. *The matrix product $\prod_{j=0}^{n-2} G_j C_{n-2}(x-z)$ is the bidiagonal upper triangular Toeplitz matrix*

$$\begin{pmatrix} B \\ \mathbf{0}^T \end{pmatrix} = \begin{pmatrix} y & -z & & & O \\ 0 & y & -z & & \\ & \ddots & \ddots & \ddots & \\ & & 0 & y & -z \\ & & & 0 & y \\ O & & & & 0 \end{pmatrix} ,$$

having the last row $\mathbf{0}^T$, filled with zeros, and having the first row $[y, -z, 0, \ldots, 0]$ where

$$y = 1 + z^2 \ .$$

The matrix B is nonsingular unless $z^2 = -1$ and is well conditioned if

$$|y| = |1 + z^2| \le |z| \ .$$

We can deflate the polynomial p if its root z satisfies $z^2 = -1$, so hereafter we assume that $z^2 \neq -1$ and the matrix B is nonsingilar.

Now for a given value z, we compute the entry $y = 1 + z^2$ and therefore the bidiagonal Toeplitz matrix B in two ops, compute the vector

$$\widehat{\mathbf{p}} = [I_{n-1} \mid \mathbf{0}] W \bar{\mathbf{p}}$$

by using $4n - 4$ ops, and finally compute the solution

$$\mathbf{y} = \bar{\mathbf{l}}_2^{(0)} = B^{-1} \widehat{\mathbf{p}}$$

to the bidiagonal Toeplitz linear system

$$B\mathbf{y} = \widehat{\mathbf{p}}$$

by using $n-2$ subtractions and either single division by y and $n-1$ multiplications or $n-1$ divisions.

Overall we use $6n-5$ or $6n-4$ ops to compute the vector $\mathbf{y} = \bar{\mathbf{l}}_2^{(0)}$, which is a least squares solution of the linear system

$$W C_{n-2}(l_1^{(0)}) \bar{\mathbf{l}}_2^{(0)} = W \bar{\mathbf{p}} \ ,$$

and so it defines an approximate solution to the convolution equation

$$l_1^{(0)} l_2^{(0)} \approx p \ .$$

The quality of the approximation can be poor only if the matrix W is ill conditioned. In virtue of Fact 8.1 this does not occur where $|z| < 1/2$, and we can ensure such a bound on $|z|$ by scaling the variable x.

We can assume that $|z| \le 1$ because we can shift to the reverse polynomial p_{rev} otherwise. Then it is sufficient to scale x by $1/4$ or 0.4, say, and for a large class of inputs even scaling by $1/2$ is sufficient.

9. KIRRINNIS' COMPUTATION OF PFDS AND LINKS TO THE WDK ITERATION

Kirrinnis (1998) refines factorization (3.1) and PFD (3.2) by working with polynomials and PFDs but does not use structured matrices. Let us recall his iteration and link it to the WDK iteration.

In its basic version (cf. Sections 3 and 4) the iteration starts with approximate factors $l_1^{(0)}, \ldots, l_m^{(0)}$, initializes the iteration cycles with setting $k = 0$, and successively computes the quotient polynomials

$$q_j^{(k)} = l^{(k)} / l_j^{(k)}, \ j = 1, \ldots, m \ , \qquad (9.1)$$

their reciprocals modulo $l_j^{(k)}$,

$$v_j^{(k)} = (1/q_j^{(k)}) \bmod l_j^{(k)}, \ j = 1, \ldots, m \ , \qquad (9.2)$$

the corrections

$$\Delta_j^{(k)} = (p v_j^{(k)}) \bmod l_j^{(k)} \ , \ j = 1, \ldots, m \ , \qquad (9.3)$$

and the updated approximate factors

$$l_j^{(k+1)} = l_j^{(k)} + \Delta_j^{(k)}, \ j = 1, \ldots, m \ ; \qquad (9.4)$$

then the iteration repeats the cycle for k replaced by $k+1$ until the approximate factors $l_1^{(k)}, \ldots, l_m^{(k)}$ are obtained within a fixed error tolerance.

REMARK 9.1. *The system of equations (9.1) and (9.2) is equivalent to the PFD*

$$\frac{1}{l^{(k)}} = \frac{v_1^{(k)}}{l_1^{(k)}} + \cdots + \frac{v_m^{(k)}}{l_m^{(k)}} \approx \frac{1}{p} \ .$$

Indeed multiply both sides of the PFD by $l_j^{(k)}$ and then reduce both sides modulo $l_j^{(k)}$.

One can combine equations (9.2) and (9.3) into the equations

$$\Delta_j^{(k)} = (p/q_j^{(k)}) \bmod l_j^{(k)} \ , \ j = 1, \ldots, m \ ,$$

and so

$$\Delta_j^{(k)} = p(z_j^{(k)}) / q_i^{(k)}(z_j^{(k)}) \ , \ j = 1, \ldots, m \ , \qquad (9.5)$$

provided $l_j^{(k)} = x - z_j^{(k)}$ (cf. (2.3)). These are the WDK corrections, such that

$$l_j^{(k+1)} = l_j^{(k)} + p(z_j^{(k)}) / q_j^{(k)}(z_j^{(k)}) \ ,$$

$$z_j^{(k+1)} = z_j^{(k)} - p(z_j^{(k)}) / q_j^{(k)}(z_j^{(k)}) \ .$$

Given n initial approximations to n distinct monic linear factors of p, we arrive at the WDK iteration, which is proved to have quadratic local convergence; empirically it has excellent global convergence. One can avoid computing the coefficients of the polynomials $q_j^{(k)}$ and compute the values

$$q_j^{(k)}(z_j^{(k)}) = \prod_{i \neq j} (z_j^{(k)} - z_i^{(k)})$$

by using $2n - 3$ ops for every pair $\{j, k\}$.

Kirrinnis (1998) has not pointed out this WDK link, but his algorithm generalizes the iteration to refining a factorization of a polynomial $p = p(x)$ into the product of m factors for any integer m, $1 < m \le n$. He extended to this case the classical results on local quadratic convergence of Newton's iteration and estimated the Boolean (that is bitwise) operation complexity provided that the factors $L_1, \ldots, L_m$ as well as their initial approximations

$$l_1^{(0)} \approx l_m^{(0)}$$

have their zero sets pairwise isolated and have all zeros lying in the unit disc $D(0,1) = \{x : |x| \le 1\}$. (We can move all zeros of p into this disc by scaling the variable x).

Under the latter assumptions he proposes to replace expressions (9.2) for updating the polynomials $v_i^{(k)}$ for $i = 1, \ldots, m$ and $k > 1$ by the following expressions,

$$v_i^{(k)} = v_i^{(k-1)} + (2 - v_i^{(k-1)} q_i^{(k)}) v_i^{(k-1)} p \bmod l_i^{(k)} \ . \quad (9.6)$$

This replacement is aimed at avoiding the computation of PFDs and the modular reciprocals together with the implied problems of numerical stability of these computations, but we rely on equations (3.1)–(3.3) because the problems of numerical stability are mild in our case of computing the reciprocal modulo a polynomial of a small degree d_i; these problems should disappear where $d_i = 1$ (cf. (9.5)).

Furthermore, in the case where $m = 2$ and $d_1 = O(1)$ we compute the corrections $\Delta_1^{(k)}$ and $\Delta_2^{(k)}$ by using $O(n)$ ops versus order $n \log n$ ops, required for updating $l_1^{(k)}$ and $l_1^{(k)}$ based on (9.6).

10. ROOT-FINDING WITH RECURSIVE LEAST SQUARES UPDATING

Given an initial approximate factor $l_1^{(0)}$ we can recursively update it according to the equations

$$l_1^{(k)} = p / l_2^{(k)} \mod l_1^{(k)}, \ k = 0, 1, \ldots$$

where for all k the second factors $l_2^{(k)}$ are updated according to the recipes of Section 8. For

$$d_1 = 1, \ l_1^{(0)} = x - z_1^{(0)} \ ,$$

we obtain the WDK corrections

$$\Delta_1^{(k)} = \frac{p(z_1^{(k)})}{l_2^{(k)}(z_1^{(k)})} \ ,$$

such that

$$l_1^{(k+1)} = l_1^{(k)} + \frac{p(z_1^{(k)})}{l_2^{(k)}(z_1^{(k)})} \ .$$

This *Laser WDK iteration* is directed towards a single zero of p, although it also produces approximate quotients $l_2^{(k)}$, $k = 0, 1, \ldots$

The resulting algorithm is reasonably effective. Compared to the algorithm of Section 6.2, its iteration loop takes about as many ops if we compute the second approximate factors $l_2^{(k)}$ as the quotients of polynomial division (cf. (8.2)).

The computation of such factors $l_2^{(k)}$ as constrained least squares solutions to the linear systems (8.1) (with subscripts k replacing zero) takes a little more ops (that is about $10n$ versus about $6n$ ops for updating both factors $l_1^{(k)}$ and $l_2^{(k)}$).

Our preliminary tests (to be continued) have showed similar convergence patterns for the three main algorithms, that is the one of Section 6.2 and the two algorithms based on approximate factorizations $p \approx l_1^{(k)} l_2^{(k)}$ and either polynomial division or computing constrained least squares solution of the linear systems extending (8.1).

11. SUMMARY AND DISCUSSION

We recalled Kirrinnis' algorithm for polynomial factorization, linked it to the WDK iteration, and more importantly, modified its PFD computation stage by reducing the task to the solution of a Sylvester or generalized Sylvester linear system of equations.

Empirically the resulting algorithms preserve strong global convergence property of the WDK iteration, but they also enable us to decrease its arithmetic cost to $O(n)$ ops per iteration loop when we narrow the task to the approximation of a single zero of p.

Clearly one can approximate up to m roots concurrently by using m processors.

Our auxiliary techniques for solving Sylvester and generalized Sylvester linear systems, their link to PFDs, our extension of the Gohberg–Semencul inversion formula from Toeplitz to generalized Sylvester matrices, numerical stabilization of the solution and our modified least squares solution of the bidiagonal Toeplitz linear systems of equations can be of independent interest.

We plan further theoretical and experimental study of all our univariate polynomial root-finders and root-refiners, including comparative tests of their global and local convergence and of the impact of various implementation "details" such as the recipes for shifting to the reverse polynomial $p_{\rm rev}$ and scaling the variable x and for adjusting our algorithms to the approximation of complex conjugate pairs of the zeros of a polynomial with real zeros, as well as allowing variable coefficients of one of the approximate factors (cf. Section 6.4).

As our more general undertaking we will explore the power of the PAC in devising root-finders and root-refiners for univariate and multivariate polynomial (and possibly nonpolynomial) equations and systems of equations and will study convergence of the resulting algorithms both theoretically and experimentally.

In particular for which classes of equations and systems of equations can we improve global convergence of Newton's iteration by immersing the original equations into a larger system defined with more equations and variables? What are the most effective ways for such enlargements?

12. REFERENCES

[1] Barnett, S. (1983), *Polynomial and Linear Control Systems*, Marcel Dekker, New York

[2] Bini, D. (1996), Numerical Computation of Polynomial Zeros by Means of Aberth's Method, *Numerical Algorithms* **13**, 179–200

[3] Bini, D. A. and Fiorentino, G. (2000), Design, Analysis, and Implementation of a Multiprecision Polynomial Rootfinder, *Numerical Algorithms* **23**, 127–173

[4] Box, G. E. P. and Jenkins, G. M. (1976), *Time Series Analysis: Forecasting and Control*, Holden-Day, San Francisco, California

[5] Corless, R. M., Gianni, P. M., Trager, B. M. and Watt, S. M. (1995), The Singular Value Decomposition for Polynomial Systems, *Proc. Intern. Symposium on Symbolic and Algebriac Computation (ISSAC'95)*, 195–207, ACM Press, New York

[6] Demeure, C. J. and Mullis, C. T. (1989), The Euclid algorithm and the fast computation of cross–covariance and autocovariance sequences, *IEEE Trans. Acoust., Speech, Signal Processing* **37**, 545–552

[7] Demeure, C. J. and Mullis, C. T. (1990), A Newton–Raphson method for moving-average spectral factorization using the Euclid algorithm, *IEEE Trans. Acoust., Speech, Signal Processing* **38**, 1697–1709

[8] Gohberg, I. and Semencul, A. (1972), On the Inversion of Finite Toeplitz Matrices and Their Continuous Analogs, *Matematicheskiie Issledovaniia* (in Russian) **7**, **2**, 187–224

[9] Golub, G. H. and Van Loan, C. F. (1996), *Matrix Computations*, Johns Hopkins University Press, Baltimore, Maryland (third edition)

[10] Habbard, J., Schleicher, D. and Sutherland, S. (2001), How to Find All Roots of Complex Polynomials by Newton's Method, *Invent. Math.* **146**, 1–33

[11] Kirrinnis, P. (1998), Polynomial factorization and partial fraction decomposition by simultaneous Newton's iteration, *J. of Complexity* **14**, 378–444

[12] McNamee, J. M. (1993), Bibliography on Roots of Polynomials, *Journal of Computational and Applied Mathematics* **47**, 391–394

[13] McNamee, J. M. (1997), A Supplementary Bibliography on Roots of Polynomials, *Journal of Computational and Applied Mathematics* **78**, 1

[14] McNamee, J. M. (2002), A 2002 update of the supplementary bibliography on roots of polynomials, *J. Computational and Applied Mathematics* **142**, 433–434

[15] McNamee, J. M. (2007), *Numerical Methods for Roots of Polynomials*, Elsevier

[16] Pan, V. Y. (1995), Optimal (up to polylog factors) sequential and parallel algorithms for approximating complex polynomial zeros, *Proc. 27th Ann. ACM Symp. on Theory of Computing*, 741–750, ACM Press, New York

[17] Pan, V. Y. (1996), Optimal and nearly optimal algorithms for approximating polynomial zeros, *Computers and Math. (with Applications)* **31, 12**, 97–138

[18] Pan, V. Y. (1996a), Computing $x^n \mod p(x)$ and an Application to Splitting a Polynomial into Factors over a Fixed Disc, *Journal of Symbolic Computations* **22**, 1–4

[19] Pan, V. Y. (2001), Univariate Polynomials: Nearly Optimal Algorithms for Factorization and Rootfinding, *Proc. International Symposium on Symbolic and Algebraic Computation (ISSAC '01)*, 253–267, ACM Press, New York

[20] Pan, V. Y. (2001a), *Structured Matrices and Polynomials: Unified Superfast Algorithms*, Birkhäuser/Springer, Boston/New York

[21] Pan, V. Y. (2002), Univariate Polynomials: Nearly Optimal Algorithms for Factorization and Rootfinding, *Journal of Symbolic Computations* **33, 5**, 701–733

[22] Pan, V. Y. and Zheng, A. (2011), New Progress in Real and Complex Polynomial Root-Finding, *Computers and Math. (with Applications)* **61**, 1305–1334

[23] Park, H., Zhang, L. and Rosen, J. B. (1999), Low Rank Approximation of a Hankel Matrix by Structured Total Least Norm, *BIT* **39**, 757–779

[24] Rosen, J. B., Park, H. and Glick, J. (1996), Total Least Norm Formulation and Solution for Structured Problems, *SIAM Journal on Matrix Analysis and Applications* **17, 1**, 110–128

[25] Rosen, J. B., Park, H. and Glick, J. (1999), Structured Total Least Norm for Nonlinear Problems, *SIAM Journal on Matrix Analysis and Applications* **20, 1**, 14–30

[26] Schönhage, A. (1982), The fundamental theorem of algebra in terms of computational complexity, *Department of Math., University of Tübingen*, Tübingen, Germany

[27] Van Dooren, P. M. (1994), Some numerical challenges in control theory, *Linear Algebra for Control Theory, IMA Vol. Math. Appl.* **62**

[28] Weierstrass, K. (1903), Neuer Beweis des Fundamentalsatzes der Algebra, *Mathematische Werker* **Tome III**, 251–269, Mayer und Mueller, Berlin

[29] Wilson, G. T. (1969), Factorization of the Covariance Generating Function of a Pure Moving-average Process, *SIAM Journal on Numerical Analysis* **6**, 1–7

Roots of the Derivatives of some Random Polynomials

André Galligo*
Universite de Nice-Sophia Antipolis, Mathematiques
Laboratoire de Mathématiques, Parc Valrose, 06108 Nice cedex 02, France
galligo@unice.fr

ABSTRACT

Our observations show that the sets of real (respectively complex) roots of the derivatives of some classical families of random polynomials admit a rich variety of patterns looking like discretized curves. To bring out the shapes of the suggested curves, we introduce an original use of fractional derivatives. Then we present several conjectures and outline a strategy to explain the presented phenomena. This strategy is based on asymptotic geometric properties of the corresponding complex critical points sets.

Categories and Subject Descriptors

I.1.2 [**Computing Methodologies**]: Symbolic and Algebraic Manipulation—*Algorithms*

General Terms

Experimentation, Theory

Keywords

random polynomials; random matrices; fractional derivatives; critical points; patterns; roots of real univariate polynomial; stem

1. INTRODUCTION

Computer algebra systems are powerful tools for performing experiments and simulations in Mathematics. They serve to illustrate known properties, already rigorously proved, or conjectures; to find examples, to show that a bound is sharp, to estimate some values or behaviors. Once in a blue moon, experiments reveal unexpected patterns or phenomena. After the surprise, the repetition of experiments and variations to test robustness, comes the time to share the observations and the quest for explanations.

*and INRIA Mediterrannée, Galaad project team. Partially supported by the European Marie Curie net SAGA

This paper relates my experiments, relying on the computer algebra system Maple, on the root sets of univariate polynomials of medium degrees, and of their iterated derivatives. Generations of mathematicians studied these basic objects, so it seemed unlikely that simple graphics should uncover any surprising feature. The originality of the presented approach was to choose random polynomials, compute roots of a great number of derivatives and consider averaged objects and phenomena. I started observing the real roots and then looked at the complex ones, as they are more amenable to algebraic interpretations.

Random matrices are matrix-valued random variables, their study have stimulated a great deal of interest in the last decades, since many important properties of disordered physical systems can be represented mathematically using eigenvectors and eigenvalues of matrices with elements drawn randomly from statistical distributions. Their characteristic polynomials form a special class of random polynomials. See [15], or for a first insight the *Random matrix* entry in Wikipedia. Random polynomials is a classical field of interest in Mathematics and Statistics; several families of random polynomials have been described in great detail, see [9]. The number and distribution of real and complex roots of random polynomial present regular structures (see section 2 below) which are statistical consequences of the properties of their coefficients distributions. This is also the case for eigenvalues of random matrices, see [6].

For a fixed degree n, we consider several bases $g_i(x)$ of polynomials of degree at most n. We form the polynomial $f := \sum_{i=0}^{n} a_i g_i(x)$, the set of coefficients a_i being instances of $n+1$ independent normal centered standard distributions. In our experiments, we also consider characteristic polynomials of matrices whose entries are independent normal centered standard distributions.

A critical point of a polynomial $f(x)$ is a root of its derivative $f'(x)$. Since random polynomials are almost surely generic, they admit only critical points that are not also roots of f; in the sequel we will only be interested by these critical points. By Rolle theorem, between two roots of f there is at least one root of f' while in the complex plane, by Gauss-Lucas theorem, the critical points of f are contained in the convex hull of the roots of f. There are several improved versions of this theorem, see the excellent book [17] which contains many results and enlightening historical notes.

Our general project is to concentrate on some families of random polynomials and present new conjectures, on the set of their critical points, suggested by experiments, observations and numerical evidence. It extends our previous works

[12], [8], [10], [11]. Our conjectures, hopefully transformed into theorems, could then act as an oracle and indicate the estimated number and locations of the roots of a random polynomial and of its derivatives. This information could be used to derive better average complexity bounds for root isolation algorithms.

The paper is organized as follows. In section 2, different families of random polynomials are introduced; some of their properties will be recalled and illustrated. Section 3 is devoted to our experiments on the sets of real roots of these polynomials and their derivatives; we organize them in a Variation diagram. Then we point out intriguing patterns and present a conjecture to try to express formally a part of the observed phenomena. Section 4 introduces our definition of a polynomial bivariate factor P of the fractional derivatives of a polynomial f, P induces a continuation between the roots sets of f and xf'. With this tool, we define an algebraic spline curve we call the "stem" of the polynomial f; it is of particular interest for random polynomials. In addition, section 4 describes experimental results on the stems, points out another intriguing phenomenon and presents a conjecture which leads to analyze the influence of the complex roots of f. Section 5 concentrates on the relative locations of the complex roots of f and f'. For some random polynomials, an interesting pairing is observed and its consequences explored. Finally we conclude discussing a tentative analysis and synthesis of our observations based on the symmetries of the limit distribution of the complex roots of f.

2. RANDOM POLYNOMIALS

The study of random polynomials is a classical and very active subject in Mathematics and Statistics It is at the core of extensive recent research and has also many applications in Physics and Economics; two books [3] and [9] are dedicated to it. Already in 1943, Mark Kac [14] gave an explicit formula for the expectation of the number of roots of a polynomial in a class that now bears his name (see below). The subject is naturally related to the study of eigenvalues of random matrices with its applications in Physics, see [6].

For a fixed degree n, we consider several bases $g_i(x)$ of polynomial of degree at most n, then we form the polynomial

$$f := \sum_{i=0}^{n} a_i g_i(x).$$

The coefficients a_i being instances of $n+1$ independent standard normal distributions $N(0,1)$. We are concerned with averaged asymptotic behaviors when n tends to infinity, but in our experiments we chose most n between 32 and 128, so the reader can easily repeat and test them. We also considered characteristic polynomials of matrices with various shapes whose entries are independent standard normal distributions.

Let's start with the following methodological point. In statistics, averaged properties are generally observed through a series of realizations forming a sample. However in our setting, some families of large degree random polynomials, the uniformity of a distribution of roots, a symmetry or an intriguing regular shape shows up in almost each experiment. A single large object is enough to represent the features of most objects of the whole ensemble, in other words a significant sample contains only one element. This convenient behavior can be related to a property of some disordered systems called "self-averaging". However, our situation is more complicated, since we did not fix in advance any feature of the observed shapes; they are extracted from the pictures.

We now list some classes of random polynomials, we specify their names and the corresponding bases $g_i(x)$ (cf. the above formula).

- Kac polynomials: the basis is $g_i(x) = x^i$.
- $SO(2)$-polynomials: $g_i(x) = \sqrt{\binom{n}{i}}x^i$.
- Weyl-polynomials: $g_i(x) = \sqrt{\frac{1}{i!}}x^i$.

Then, the following less commonly studied families; in this paper we give them the following names (NC stands for normal combination):

- NC-Bernstein : $g_i(x) = \binom{n}{i}(1+x)^i(1-x)^{n-i}$.
- NC-Chebyshev: the basis is made by the Chebyshev polynomials of degree i, for i between 0 and n.

Then the characteristic polynomials of several classes of random matrices

- matrices whose entries are instances of independent standard normal distribution,
- symmetric matrices whose entries are instances of independent standard normal distribution,
- random unitary matrices obtained by taking the eigenvectors of a matrice of the previous class.

Sparse analog of these classes and other distributions of their coefficients (or entries) are also very interesting; but the listed classes are already rich enough to express our observations and conjectures.

Number of real roots and distribution of complex roots, cf. [3] and [9]

- The asymptotic number of real roots of a Kac polynomial is about $\frac{2}{\pi}\ln n$, the distribution of the complex roots tends to a uniform distribution on the unit circle.
- The asymptotic number of real roots of a $SO(2)$- polynomial is about $\sqrt{n}$, the distribution of the complex roots tends to a uniform distribution on the Riemann sphere.
- The asymptotic number of real roots of a Weyl polynomial is about $\frac{2}{\pi}\sqrt{n}$, the distribution of the complex roots tends to a uniform distribution on the disc centered at the origin and of radius $\sqrt{n}$.
- The asymptotic number of real roots of the characteristic polynomials of a general random matrix is about $\sqrt{\frac{2}{\pi}}\sqrt{n}$, the distribution of the complex roots tends to a uniform distribution on the disc centered at the origin and of radius $\sqrt{n}$. Figure 1 shows them for a matrix of size 128.

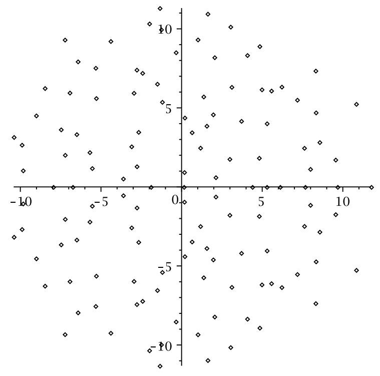


Figure 1: Complex eigenvalues of a random matrix

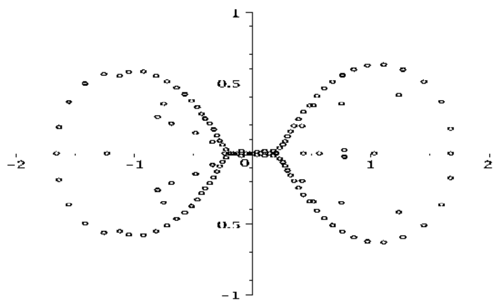


Figure 2: Complex roots of a NC Chebyshev

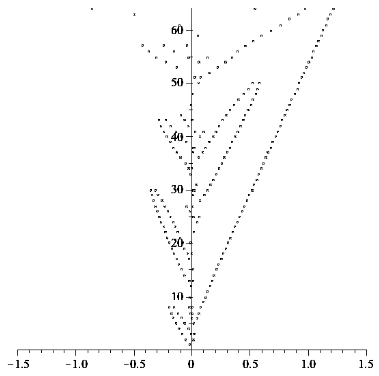


Figure 3: VD of a Kac polynomial of degree 64

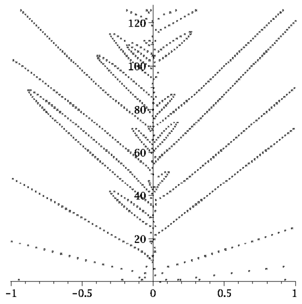


Figure 4: (Truncated) VD of a SO(2) polynomial

In Figure 1 we notice that the limit distribution is almost uniform in angles around the origin, (rotational symmetric) this property is completed by an axial symmetry over the real axis due to complex conjugation. It is also true for the three first distributions. This observation can be quantified: [18] computed for Kac polynomials the density function $h_n(x, y)$ of the number of complex roots near a complex point $x + iy$, completing Kac's computation of the density function of the number of real roots near a real point x.

For the two other classes, NC-Bernstein and NC-Chebyshev, the limit distribution has only a central symmetry.

- The asymptotic number of real roots of a polynomial in NC-Bernstein is about $\sqrt{2n}$, [8].
- Figures 2 shows, for a NC-Chebyshev polynomial of degree 128, the distribution of the complex roots, they concentrate along a segment of the real axis and two ovals around -1 and 1.

3. VARIATION DIAGRAM (VD)

We consider a polynomial $f(x)$ with real coefficients of degree n and its i-th derivative $F[i] = f^{(i)}(x)$ for $i = 0..n - 1$. The sets of real roots of the n polynomials $F[i]$, appears in what, in French high schools is called "tableau de variations". In the 19-th century, the number of sign variations were used by Budan and by Fourier to estimate the number of roots of f in an interval, see [17], chapter 10. In the 20-th century, R. Thom relied on the signs of $f^{(i)}$ to distinguish and label the different real roots of f, see [4].

We chose to organize all these roots with a 2D diagram, that we call Variation diagram (VD), the $(n - i)$-th row contains the real roots of the (i)-th derivative of f:

$$VD := \cup_i \ \{fsolve(F[i], x)\} \times \{n - i\}$$

Note that the second coordinate indicates the degree of the polynomial $F[i]$. As the iterated derivatives of a generic polynomial do not have multiple roots, they change sign at each root.

Example:

$$f := (x-5).(x^2-x+4) \ ; \ f' = 3(x-1).(x-3) \ ; \ f'' = 6(x-2).$$

These polynomials have respectively 1, 2, 1 real roots:

$$VD = \{[5, 3], [1, 2], [3, 2], [2, 1]\}.$$

Our first experiments with Kac polynomials found that their VD admit unexpected structured patterns. The roots of the successive derivatives present almost dotted curves and alignments. To our best knowledge, this phenomenon has not been explored before.

We made more experiments with different instances of Kac polynomials and got very similar patterns, then we repeated the experiments with the different bases defined in the previous section. See Figures 3 to 6.

As illustrated by these pictures and many more, for the cited families of random polynomial the observed feature (almost alignments along lines or ovals) seems robust. In particular following our observation we conjecture:

Conjecture 1. *When n tends to infinity, for Kac, $SO(2)$ or Weyl random polynomials; if the the root with largest,*

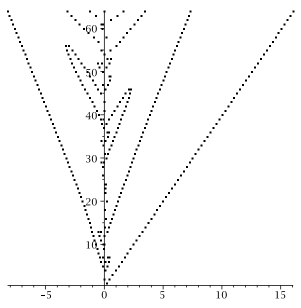


Figure 5: VD of a Weyl polynomial of degree 64

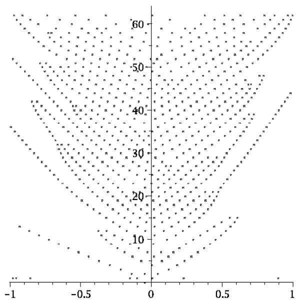


Figure 6: (Truncated) VD of a CN Bernstein

resp. smallest, real part is real then the largest, resp. smallest, real root of all (but the last) derivatives of f tend to be almost surely aligned.

Question 1. *When n tends to infinity, for Kac, $SO(2)$ or Weyl random polynomials; compute the probability that the root with largest real part is real.*

In order to strengthen these observations, we looked for a method to connect the points in coherence with our visual intuition. A natural strategy is to view the integer orders of derivation as discretized steps; hence to look for generalized derivatives with continuous orders.

4. FRACTIONAL DERIVATIVES

The attempt to introduce and compute with derivatives or antiderivatives of non-integer orders goes back to the 17-th century. In their book [16] dedicated to this subject, the authors relate that an integral equation, the tautochrone, was solved by Abel in 1823 using a semi derivative attached to the integral $\int_0^x \sqrt{x-t} f(t)dt$. In 1832 Liouville expanded functions in series of exponentials and defined q-th derivatives of such a series by operating term-by-term for q a real number. Riemann proposed another approach via a definite integral. The cited book provides in its introduction a nice presentation of the historical progression of the concept from 1695 to 1975 through a hundred citations.

An important property is that two such fractional derivations commute. However for non-integer orders of derivation, the fractional derivative at a point x of a function f does not only depend on the graph of f very near x; fractional derivations do not commute with the translations on the variable x. The traditional adjective "fractional", corresponding to the order of derivation, is misleading since it need not be rational.

Let us emphasize that nowadays in Mathematics, fractional derivatives are mostly used for the study of PDEs in Functional analysis. They are presented via Fourier or Laplace transforms. Fractional derivatives are seldom encountered in Polynomial algebra.

4.1 A new polynomial

In order to interpolate the previous dotted curves, we consider a polynomial factor of the fractional derivatives of the polynomial f. We rely on Peacock's rule (1833) for monomials:

$$\mathrm{Diff}_a(x^n, x) := \frac{n!}{(n-a)!} x^{n-a}; \; for\, a > 0, \; n\, integer.$$

we noticed the following simple but key fact, to our better knowledge it was not mentioned before.

Lemma 1. *Let $f(x)$ be a polynomial of degree n, then*

$$x^a \Gamma(-a) Diff_a(f)$$

is a polynomial in x and a rational fraction in a with denominator $(n-a)(n-a-1)...(-a)$.

To interpolate the non vanishing roots of the successive derivatives of a polynomial f, only fractional derivatives with $0 < a < 1$ are needed. Moreover if we consider separately the positive and the negative roots, then we can skip the factor fractional powers of x. So we set the following definition and notation.

Definition 1. *Let $f = \sum a_i x^i$ be a degree n polynomial. We call (monic) polynomial factor of a fractional derivative of order a of f,the polynomial $x^a \frac{(n-a)!}{n!} Diff_a(f, x))$. It is a polynomial of total degree n in x and a, which may be written:*

$$P_a(f) := a_n x^n + \sum_{i=0}^{n-1} (\prod_{j=i+1}^{n} 1 - \frac{a}{j}) a_i x^i.$$

4.2 Stem

Definition 2. *We call Stem of a polynomial f of degree n, the union of the real curves formed by the roots of all the monic polynomial factors of the derivatives $f^{(i)}$ of f, for i from 0 to $n-1$ and $0 \le a < 1$. A stem is a C^0 spline of algebraic curves.*

See Figures 7 to 9.

Here is a simple example to illustrate the regularity of the join between two successive curves forming the stem of f:

$$f = (x-1)(x-3) = x^2 - 4x + 3 \; ; \; f' = 2(x-2).$$

Hence,

$$P_a(f) = x^2 - 2x(2-a) + 3(2-a)(1-a)/2,$$

$$P_b(f'/2) = x - 2(1-b).$$

This shows that when a tends to 1, and b tends to zero, there is (only) a C^0 continuity between the adjacent pieces.

In a joint work with D. Bembe [2], we consider another curve associated to f which is regular but does not have the same shape: the real algebraic curve in the plane (x, a)

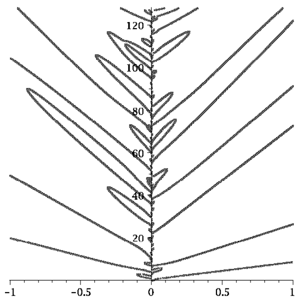


Figure 7: Stem of the previous SO(2) polynomial (deg=128)

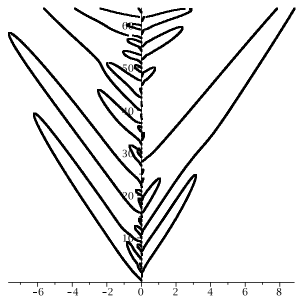


Figure 9: Stem attached to a random matrix

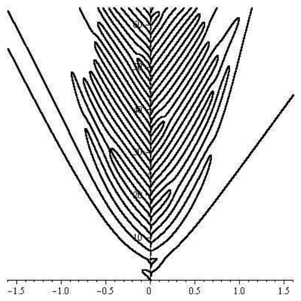


Figure 8: Stem of a NC Chebyshev of degree 64

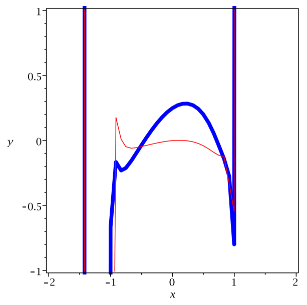


Figure 10: Graphs of f , xf' for a Kac polynomial

defined by the bivariate polynomial of degree n that we denoted above by $P_a(f)$. We study the relation between this curve and the so-called virtual roots of f introduced in [13] and [5].

4.3 From discrete to continuous

I made several experiments, and noticed that the patterns exhibited by the stems of most of our random polynomials presented common features:

- Long quasi lines joining the external real roots of f to the axis ($x = 0$),
- Curves (of smaller size) joining the inner real roots to the axis ($x = 0$),
- Closed curves, often shaped like an ear, starting and ending at ($x = 0$).
- Since the coefficients of $P_a(f)$ are random numbers, the corresponding curves are almost surely smooth, therefore the "ears" do not touch.
- Stems of a same family of random polynomials share more similarities.

Remark: In our pictures, the line $x = 0$ is a singularity and an axis of almost symmetry of the patterns. This is coherent with the considered random polynomials with centered distribution of coefficients. Our choice of fractional derivatives respects this symmetry.

4.4 Similarity of graphs

The graph of a (random) polynomial bears interesting features. Algebraically and visually its shape is related to the roots of f and its derivatives through extrema, inflection points, and generalized inflections. We compared the graphs of f and f' or xf', for our random polynomials, expecting that randomness acts as a filter: details are blurred and similarities are magnified.

We made several experiments with Maple for polynomials f of medium degrees. For a Kac polynomials of degree 64, we rescaled f, and xf', restricted the graphs to $x \in [-2, 2]$ and $y \in [-1, 1]$. As illustrated in Figure 10 we found that, very often, the graph of xf' is similar to the graph of f but shrunk towards the origin. This is coherent with the pattern exhibited by the variation diagram of f. The problem is how to quantify the observed transformation.

4.5 Rolle theorem

Generically, there are an odd number of roots of f' between two successive positive roots x_1 and x_2 of f. We aim to analyze the pairing between the roots established by the stem of f, the previous figures suggest that for our random polynomials, almost surely the stem connects the root x_2 of f to one of the roots of f'. Stems of random polynomials may have points with horizontal tangents, but we observed that at these points the graph is convex. In other words, we propose this property as a conjecture.

Conjecture 2. *For the chosen families of random polynomials, almost surely the stem between the roots of f and xf' does not connect two roots of f.*

Another interesting task would be to quantify the ratio defined by the consecutive roots of f and f'. Such estimates

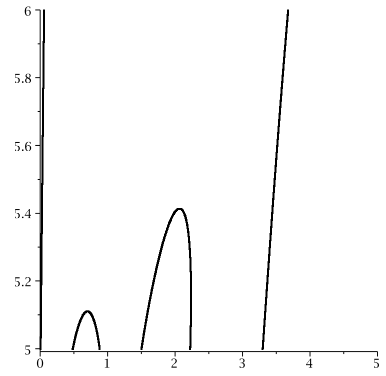


Figure 11: detail of a Stem

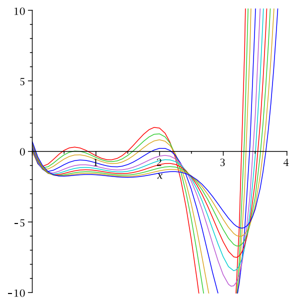


Figure 12: a "good" homotopy between f and xf'

were given obtained by P. Andrews [1] for hyperbolic polynomials.

Figures 11 and 12 show the detail of a "good" (with respect to the conjecture) example of the stem of a polynomial of degree 6, and the corresponding deformation between the graphs of f and xf', formed by the fractional derivatives. In contrast, Figures 13 and 14 show the corresponding pictures for an example which does not correspond to the situation encountered with our random polynomials. More precisely, in the "bad" example the continuation between the inner roots of f and xf' does not remain on the real line but passes through the complex plane, this is pictured by the dotted curve. So, an explanation of the connection between real roots of f and f' might be found exploring what happens in the complex plane.

5. COMPLEX CRITICAL POINTS

There is an important bibliography on the location of the critical points of a polynomial with respect to the location of its roots, going back to Gauss with Gauss-Lucas theorem. Several recent works concentrate on the following conjecture of Sendov, which has been proved for small degrees and in several special cases. Their main tools rely either on the implicit function theorem or on extremal polynomials, or on refinements of Gauss-Lucas theorem. See the book [17].

Sendov Conjecture: Let f be a polynomial having all its roots in the disk D. If z is a root of f, then the disk $z + D$ contains a root of f'.

I did not found mention in these researches, developed in analysis and approximation theory, of the case of polynomi-

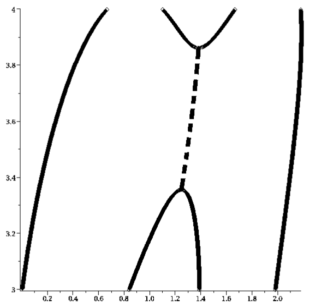


Figure 13: detail of a "bad" Stem

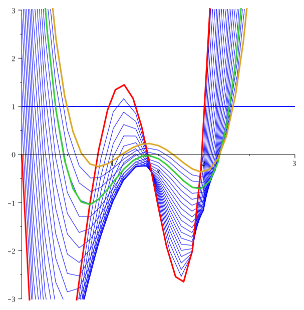


Figure 14: a "bad" homotopy between f and xf'

als with random coefficients.

5.1 Observations

I made experiments with the first classes (see section 2) of random polynomials, they exhibit interesting behaviors:

- for almost each root of f smaller disks $z + \epsilon D$, with $\epsilon << 1$, contain a critical point; in such a way that they describe a bijection between the roots of f and the roots of xf',

- one can restrict these disks to small sectors, which indicates a direction towards the real axis or towards the origin.

Figures 15 to 17 illustrate the relation between roots and critical points for an $SO(2)$ random polynomial of degree 32. Fractional derivatives are used to construct (as for real roots) an homotopy between the zero sets of f and xf'. The color chart is: the roots of f are blue, the roots of f' are red and the roots of the fractional derivatives are green. Figure 17 shows the "top" part of a complex analog of the variation diagram, notice the regular alignments towards the origin (it is slightly perturbed near the real axis).

5.2 Electrostatic attraction

The interpretation of the position of each critical point of f as an equilibrium of a logarithmic potential, where the roots of f are viewed as positively charged particles (or rods), goes back to F. Gauss. As reported in [17], the following equality to zero provides a quick proof of Gauss-Lucas theorem.

Denote by x_j the complex roots of the polynomial f assumed distinct from each other and distinct from z, another complex number. Then by logarithmic derivation and conjugation we deduce:

$$f'(z) = 0 \;\Rightarrow\; \sum \frac{z - x_j}{|z - x_j|^2} \;=\; 0.$$

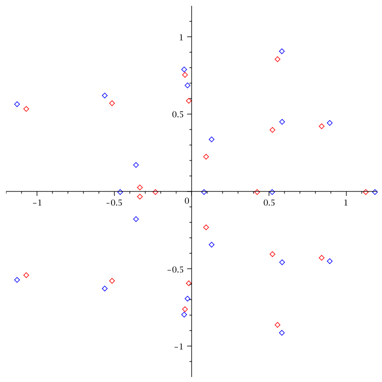


Figure 15: Roots of f and f'

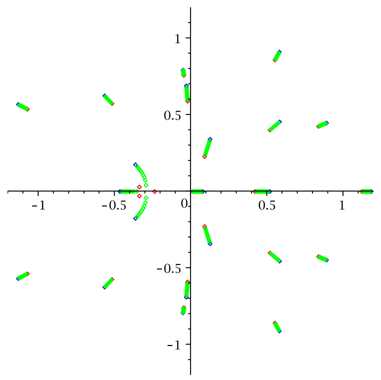


Figure 16: Pairing via fractional derivatives

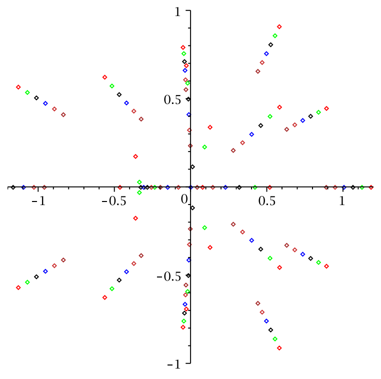


Figure 17: Truncated $SO(2)$ complex VD

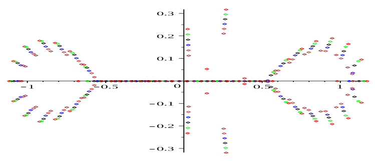


Figure 18: Higher derivatives, CN Chebyshev

The vector in the complex plane $\frac{-z+x_j}{|z-x_j|^2}$ is viewed as a force applied to z directed towards x_j proportional to the reciprocal of the distance. Summing these forces, the point z (viewed as an electron) is attracted by the roots system of f (viewed as positively charged particles). When the limit distribution of the roots of f is uniform in angles (also called rotational symmetric, i.e. only depends on the radius), the resulting electrostatic force on a point z inherits a limit symmetry, and tends to be directed towards the origin.

Let us denote by L_1 the real line joining the origin to a root x_k and by L_2 the real line orthogonal to L_1 through the origin. The number and distribution of roots below and above L_1 (respectively L_2) are asymptotically "almost" balanced. So, with a good probability, an equilibrium z_k can be found "near" L_1 with the vector x_kz_k oriented towards the origin. One can expect as well that the further x_k is far from the origin, the smaller the vector x_kz_k should be. This is what we observed in our experiments as illustrated with Figures 15 and 16.

The previous balanced count of forces is perturbed when we approach the real axis, because there is another axial symmetry due to complex conjugation, and a positive probability of real roots. This breaks the rotational symmetry, consequently the resulting electrostatic force is now also directed towards the real axis. The attraction toward the real axis is magnified when we consider the set of roots of a NC Chebyshev polynomial (see Figure 2), since it contains many real points and at the limit when n tends to infinity "only" a central symmetry. In this case, we observe that through successive pairing and after a rather small number of derivations, most complex roots of f give rise to a real root of a higher derivative of f. This process is illustrated in Figure 18 with a CN Chebyshev polynomial of degree 50, The colors (red, green, black, blue, orange, brown) correspond to the derivation orders (0,1,2,3,4,5).

Conjecture 3. *For f a Kac, $SO(2)$, Weyl random polynomial, the presented continuation process realizes a bijection such that each critical point z_k of f is attached to a root x_k. Moreover in the limit distribution of (x_k, z_k) when n tends to infinity, almost surely the vectors x_kz_k point towards the origin.*

For the other random cases with only a limit central symmetry, we also conjecture a pairing but the limit orientation of the x_kz_k will be dependant on the point x_k. Our intuition is as follows. Consider all the roots of f except x_k, the resulting electrostatic force will be regular in a small disc around x_k, the average of these forces in the disc is a good candidate for the direction we want to find, then z_k will be positioned on the corresponding line to realize the equilibrium.

6. TENTATIVE EXPLANATIONS

Let us summarize. In section 3 we described intriguing alignments of points and dotted curves, observed on the collection of all roots of a real polynomial f and its derivatives, organized in a 2D diagram (VD).

Trying to explain the curved parts of this phenomena, we presented in section 4 an original use of fractional derivatives. It allows an interpolation of the discrete set into a 2D curve which we called the stem. This construction gave rise to experiments, to other intriguing observations and to two questions: Why does the C^0 interpolation look so regular ? Is there any relation between the distribution of the complex roots of f and its stem ? These question lead us to new observations on the location of critical points of our random polynomials and again to new questions and conjectures.

A possible direction of research to mathematically answer some of these questions is to rely on the interpretation of a critical point as an equilibrium position under electrostatic forces. Indeed this viewpoint allows the use of the density functions of complex and real roots of a random polynomial f. Such density functions have been studied by several authors for classical families of random polynomials, e.g. for Kac polynomials we already cited [18].

As a first approach, one can consider a Mean Field approximation (where we replace the sum of the attractions of each individual root by their limit expectation) to establish as follows a pairing between a root of f and a root of xf'. One can estimate an equilibrium z_k near a fixed root x_k, relying on the density functions to average the attraction corresponding to the other roots of f. If the root x_k is real, complex conjugation obliges its image z_k to be real. Notice that the continuation curve between x_k and z_k, defined by the homotopy, needs not be real, we only conjectured this property with a good probability for some random polynomials.

This process will allow for random polynomials at each derivation step to "get down" towards the real axis and at least for rotational symmetric limit distribution of complex roots, towards the origin. Note that in order to prove that the continuation defined via fractional derivatives follows the same dynamic, we need to develop a generalized attraction interpretation. When a pair of conjugate complex roots of a fractional derivative reach the real axis they form a double root: in the stem of f, this event corresponds to a summit of an ear shaped curve.

We are still far from a rigorous presentation of our interpretation and all its consequences. Improvements of Gauss-Lucas, images of uniform distributions laws, might eventually explain the features exhibited by the variation diagrams, but that is a long way.

As a conclusion, I experimented, introduced new tools, presented pictures, observed phenomena. I sketched a possible strategy, which opens several problems and an exploratory research project.

Acknowledgments

I am grateful to the anonymous reviewer for his helpful careful reading and his interesting suggestions. I thank Julien Barre and Didier Clamond for valuable remarks. I thank the organizers of the conferences "Tomas Recio 60", "Analyse et probabilité", Nice 2010“ and of the "Algorithmes, Inria 2010", "University of Pisa" seminars who invited me to present preliminary versions of this work.

7. REFERENCES

[1] Andrews, P: Where not to find the critical points of a polynomial -variation on a Putnam theme. Amer. Math. Monthly, 102, pp 155-158 (1995).
[2] Bembe, D and Galligo, A: Virtual roots of real polynomials and fractional derivatives. Issac'2011 (2011).
[3] Bharucha-Reid, A. T. and Sambandham, M.: Random Polynomials. Academic Press, N.Y. (1986).
[4] Bochnack, J. and Coste, M. and Roy, M-F.: Real Algebraic Geometry. Springer (1998).
[5] Coste, M and Lajous, T and Lombardi, H and Roy, M-F : Generalized Budan-Fourier theorem and virtual roots. Journal of Complexity, 21, 478-486 (2005).
[6] Diaconis, P: Patterns in eigenvalues: the 70th Josiah Willard Gibbs lecture. Bull. Amer. Math. Soc. 40, 155-178. (2003)
[7] Edelman, A and Kostlan,E: How many zeros of a random polynomial are real? Bulletin AMS, 32(1):137, (1995).
[8] Emiris, I and Galligo, A and Tsigaridas, E: Random polynomials and expected complexity of bissection methods for real solving. *Proceedings of the ISSAC'2010 conference,* pp 235-242, ACM NY, (2010).
[9] Farahmand, K: Topics in random polynomials. Pitman research notes in mathematics series 393, Addison Wesley, (1998).
[10] Galligo, A and Miclo, L: On the cut-off phenomenon for the transitivity of subgroups. Submitted for publication (2009).
[11] Galligo, A and Poteaux, A: Computing Monodromy via Continuation methods on Random Reimann Surfaces. Theoretical Computer Science, to be published in (2011).
[12] Galligo, A and vanHoeij, M: Approximate bivariate factorization, a geometric viewpoint. In *Proceedings of the 2007 SNC conference*, pp 1-10, ACM NY, (2007).
[13] Gonzales-Vega, L and Lombardi, H and Mahé, L : Virtual roots of real polynomials. J. Pure Appl. Algebra,124, pp 147-166,(1998).
[14] Kac, M: On the Average Number of Real Roots of a Random Algebraic Equation. Bull. Amer. Math. Soc. 49, pp 314-320, (1943).
[15] Mehta, M-L: Random matrices. Pure and Applied Mathematics, 142. Elsevier/Academic Press, Amsterdam, (2004)
[16] Oldham, K and Spanier, J: The fractional calculus. Academic Press Inc. NY, (1974).
[17] Rahman, Q.I and Schmeisser, G.: Analytic theory of polynomials. Oxford Univ. Press (2005).
[18] Shepp, L.A. and Vanderbei, R.J. The complex zeros of random polynomials. Transactions of the AMS vol 347, 11, (1995).

Fast Estimates of Hankel Matrix Condition Numbers and Numeric Sparse Interpolation*

Erich L. Kaltofen
Department of Mathematics
North Carolina State University
Raleigh, NC 27695-8205, USA
kaltofen@math.ncsu.edu www.kaltofen.us

Wen-shin Lee
Mathematics and Computer Science Department
University of Antwerp
2020 Antwerpen, Belgium
wen-shin.lee@ua.ac.be

Zhengfeng Yang
Shanghai Key Laboratory of Trustworthy Computing
East China Normal University
Shanghai 200062, China
zfyang@sei.ecnu.edu.cn

ABSTRACT

We investigate our early termination criterion for sparse polynomial interpolation when substantial noise is present in the values of the polynomial. Our criterion in the exact case uses Monte Carlo randomization which introduces a second source of error. We harness the Gohberg-Semencul formula for the inverse of a Hankel matrix to compute estimates for the structured condition numbers of all arising Hankel matrices in quadratic arithmetic time overall, and explain how false ill-conditionedness can arise from our randomizations. Finally, we demonstrate by experiments that our condition number estimates lead to a viable termination criterion for polynomials with about 20 non-zero terms and of degree about 100, even in the presence of noise of relative magnitude 10^{-5}.

Categories and Subject Descriptors

F.2.1 [**Numerical Algorithms and Problems**]: Computations on matrices, Computations on polynomials; G.1.1 [**Interpolation**]; I.1.2 [**Algorithms**]: Algebraic algorithms, Analysis of algorithms

General Terms

Algorithm, Theory

*This material is based on work supported in part by the National Science Foundation under Grant CCF-0830347 (Kaltofen) and a 2010 Ky and Yu-Fen Fan Fund (Kaltofen and Yang).
This material is supported by the Chinese National Natural Science Foundation under Grant 10901055 and the Shanghai Natural Science Foundation under Grant 09ZR1408800 (Yang).

Keywords

Gohberg-Semencul formula, Hankel system, Vandermonde system, sparse polynomial interpolation, early termination

1. INTRODUCTION

The fast solution of Toeplitz- and Hankel-like linear systems is based on the classical 1947 Levinson-Durbin problem of computing linear generators of the sequence of entries. When with exact arithmetic on the entries a singular leading principal submatrix is encountered, look-ahead can be performed: this is the genesis of the 1968 Berlekamp/Massey algorithm. In the numeric setting, the notion of ill-conditionedness substitutes for exact singularity, and the algorithms need to estimate the condition numbers of the arising submatrices. By a result of Siegfried Rump [22], the structured condition number of an $n \times n$ Toeplitz/Hankel matrix H is equal to its unstructured condition number $\kappa_2(H)$: we have a Hankel perturbation matrix ΔH with spectral (matrix-2) norm $\|\Delta H\|_2 = 1/\|H^{-1}\|_2 = \|H\|_2/\kappa_2(H)$ such that $H + \Delta H$ is singular [22, Theorem 12.1]. Thus Trench's [23] (exact) inverse algorithm can produce upper and lower bounds for those distances from the Frobenius (Euclidean-vector-2) norm estimates for $1/\sqrt{n}\,\|H^{-1}\|_F \le \|H^{-1}\|_2 \le \|H^{-1}\|_F$. The look-ahead algorithms, such as the Berlekamp/Massey solver, however, need such estimates for all $k \times k$ leading principal submatrices $H^{[k]}$ for $k = 1, 2, \ldots$ of an unbounded Hankel matrix. Here we show how the 1972 Gohberg-Semencul "off-diagonal" (JL)U-representations [17, 7] can give estimates for all $\kappa_2(H^{[k]})$ in quadratic arithmetic overall time. Although our estimates are quite accurate (see Figures 1 and 2), the computation of the actual condition numbers of all leading principal submatrices in quadratic arithmetic time remains an intriguing open problem. We note that fraction-free implementations [2, 21] of the Berlekamp/Massey algorithms and inverse generators only produce values of polynomials in the entries, and our numeric Schwartz/Zippel lemma [20, Lemma 3.1] is applicable. In Section 3 we will discuss the numeric sensitivity of the expected values in that Lemma.

Our investigations are motivated by numeric sparse polynomial interpolation algorithms [11, 12, 20, 13]. In the Ben-Or/Tiwari version of this algorithm, one first computes for a sparse polynomial $f(x) \in \mathbb{C}[x]$ the linear generator for $h_l = f(\omega^{l+1})$ for $l = 0, 1, 2, \ldots$ In some exact as well as numeric algorithms, ω is selected as a random p-th root of unity. Then by the theory of early termination of that algorithm [18], the first $t \times t$ leading principle submatrix $H^{[t]}$ in the (infinite) Hankel matrix H with entries in row i and column j, $(H)_{i,j} = h_{i+j-2}$ where $k = 1, 2, \ldots, t$,

$$H^{[k]} = \begin{bmatrix} h_0 & h_1 & h_2 & h_3 & \ldots & h_{k-1} \\ h_1 & h_2 & h_3 & h_4 & \iddots & h_k \\ h_2 & h_3 & h_4 & h_5 & \iddots & h_{k+1} \\ h_3 & h_4 & h_5 & h_6 & \iddots & \vdots \\ \vdots & \iddots & \iddots & \iddots & \iddots & \\ h_{k-1} & h_k & h_{k+1} & \ldots & & h_{2k-2} \end{bmatrix} \qquad (1)$$

that is singular has with high probability dimension $t =$ the number of non-zero terms in f. For technical reasons in proof of the probabilistic analysis, we must skip over the value $f(\omega^0) = f(1)$. Hence, by the previously described algorithm for lower and upper bounds of the condition numbers of $H^{[k]}$, we can determine the number of terms t.

The Berlekamp/Massey algorithm solves the problem of computing Padé forms along a diagonal of the Padé table. Cabay and Meleshko gave an algorithm to bound from above the condition numbers of all the principal submatrices of the associated Hankel system in the well-conditioned case [8], which has been further refined and extended, for example, see [1, 6]. Here we focus less on stably computing the Gohberg-Semencul updates (see those cited papers—we require no look-ahead), but on fast and accurate lower and upper bounds of the condition numbers, which constitute the early termination criterion. We add that our approach here, namely testing submatrices for ill-conditionedness, is problematic for the approximate GCD problem (cf. [5, 4]): we know now that Rump's property is false for Sylvester matrices: the unstructured condition number of a Sylvester matrix can be large while the structured condition number is small (see [19, Example 4.2]).

We approach the problem of incrementally estimating Hankel submatrices through a numerical interpretation of the Berlekamp/Massey algorithm [21] that combines with an application of the numerical Schwartz/Zippel lemma [20]. We report some experimental results of our implementation in Section 4. For a 20 term polynomial of degree 100 the algorithm correctly computes $t = 20$ in quadratic time (given the polynomial evaluations h_l) (see Table 1). There are several issues to scrutinize. First, we only have bounds on the condition numbers and must verify that our estimates are sufficiently accurate. One would expect since for $h_l = f(\omega^{l+1})$ the Hankel matrix $H^{[k]}$ in (1) has additional structure that the upper bound estimates for the Rump condition numbers are good indicators of numeric non-singularity. Second, early termination is achieved by randomization, whose probabilistic analysis, namely the separation of the decision quantities, i.e., determinants or condition numbers, from very small values (in terms of absolute values), applies to exact computation. We can relate the numeric sensitivity of those decision quantities via a factorization of $H^{[t]}$ to clustering of term values on the unite circle, which is partially overcome by evaluation at several random ω simultaneously.

Throughout the paper, we will use the boldfaced letters $\mathbf{x}$ and $\mathbf{y}$ to denote the vectors $[x_1, \ldots, x_n]^T$, $[y_1, \ldots, y_n]^T$, respectively; $H^{[k]}$ denotes the $k \times k$ leading principle submatrix; $\kappa(H) = \|H\| \cdot \|H^{-1}\|$ denotes the unstructured condition number of H.

2. CONDITION NUMBER ESTIMATE

Suppose a $n \times n$ Hankel matrix H is strongly regular, i.e., all the leading principle submatrices of H are nonsingular. Following an efficient algorithm for solving Hankel systems [14], we present a recursive method to estimate the condition numbers of all leading principal Hankel submatrices in quadratic time.

For simplicity, in this section we only consider the condition number of the Hankel matrix with respect to 1 norm, i.e., $\kappa_1(H) = \|H\|_1 \cdot \|H^{-1}\|_1$, since all other operator norms can be bounded by the corresponding 1 norm.

In order to estimate the condition numbers of all leading principal submatrices, we need to estimate the norms of all leading principal submatrices as well as their inverses. Since a Hankel matrix H has recursive structure, the norm of $H^{[i+1]}$ can be obtained in $O(i)$ flops if the norm of $H^{[i]}$ is given, which implies that computing the norms of all leading principal submatrices can be done in $O(n^2)$ flops. Therefore, our task is to demonstrate how to estimate the norm of $(H^{[i+1]})^{-1}$ in linear time if one has the estimate of the norm of $(H^{[i]})^{-1}$.

As restated in Theorem 1, the Gohberg-Semencul formula for inverting a Hankel matrix plays an important role in our method. Note that for a given Hankel matrix H, JH is a Toeplitz matrix, where J is an anti-diagonal matrix with 1 as its nonzero entries.

Theorem 1 *Given a Hankel matrix H, suppose $\mathbf{x}$ and $\mathbf{y}$ are the solution vectors of the systems of the equations $H\mathbf{x} = e_1$, $H\mathbf{y} = e_n$, where $e_1 = [1, 0, 0, \ldots, 0]^T$ and $e_n = [0, \ldots, 0, 1]^T$. Then if $x_n \neq 0$ the inverse H^{-1} satisfies the identity*

$$H^{-1} =$$

$$\frac{1}{x_n} \underbrace{\begin{bmatrix} x_1 & x_2 & \cdot & x_{n-1} & x_n \\ x_2 & x_3 & \cdot & x_n & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ x_{n-1} & x_n & \cdot & 0 & 0 \\ x_n & 0 & \cdot & 0 & 0 \end{bmatrix}}_{JL_1} \underbrace{\begin{bmatrix} y_1 & y_2 & \cdot & y_{n-1} & y_n \\ 0 & y_1 & \cdot & y_{n-2} & y_{n-1} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & y_1 & y_2 \\ 0 & 0 & \cdot & 0 & y_1 \end{bmatrix}}_{R_1}$$

$$-\frac{1}{x_n} \underbrace{\begin{bmatrix} y_2 & y_3 & \cdot & y_n & 0 \\ y_3 & y_4 & \cdot & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ y_n & 0 & \cdot & 0 & 0 \\ 0 & 0 & \cdot & 0 & 0 \end{bmatrix}}_{JL_2} \underbrace{\begin{bmatrix} 0 & x_1 & \cdot & x_{n-2} & x_{n-1} \\ 0 & 0 & \cdot & x_{n-3} & x_{n-2} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & 0 & x_1 \\ 0 & 0 & \cdot & 0 & 0 \end{bmatrix}}_{R_2} \qquad (2)$$

The inversion formula of a Hankel matrix leads to a polynomial form, whose coefficients involve vectors $\mathbf{x}$ and $\mathbf{y}$.

Theorem 2 *Under the assumptions of Theorem 1, suppose $H^{-1} = [\gamma_1, \gamma_2, \ldots, \gamma_n]$, where γ_k denotes the k-th column of H^{-1}. Then we have*

$$\gamma_k = [\mathbf{v}_{n-1}, \mathbf{v}_{n-2}, \ldots, \mathbf{v}_0]^T,$$

where $\mathbf{v}_i$ denotes the coefficient of the polynomial $(1/x_n) \cdot$

$(\phi \cdot g_k - u_k \cdot v)$ *with respect to* Z^i, *where*

$$\phi = \sum_{i=1}^{n} x_i Z^{n-i}, \quad g_k = \sum_{i=1}^{k} y_i Z^{k-i},$$

$$v = \sum_{i=2}^{n} y_i Z^{n-i}, \quad u_1 = 0, \quad u_k = \sum_{i=1}^{k-1} x_i Z^{k-i}, k = 2, \dots, n.$$

PROOF. Considering the polynomial product $\phi \cdot g_k$, we obtain the coefficient subvector

$$\mu = [\psi_{n-1}, \psi_{n-2}, \dots, \psi_0]^T,$$

where ψ_j is the coefficient of $\phi \cdot g_k$ with respect to Z^j. Expanding $\phi \cdot g_k$, we get

$$\psi_j = \begin{cases} \sum_{i=0}^{j} x_{n-i} y_{k+i-j}, & \text{if } 0 \le j \le k, \\ \sum_{i=0}^{k-1} x_{n+i-j} y_{k-i}, & \text{if } k < j < n. \end{cases}$$

Observing the formula of H^{-1} in Theorem 1, We find that the coefficient subvector

$$\mu = JL_1.[y_k, y_{k-1}, \dots, 0, 0]^T, \tag{3}$$

where $[y_k, y_{k-1}, \dots, 0, 0]$ is the k-th column of R_1. Since $u_1 = 0$, it is obvious that the coefficient vector of $u_1 \cdot v$ is a zero vector. Considering the polynomial product $u_k \cdot v, 2 \le k \le n$, we obtain the coefficient subvector

$$\nu = [\chi_{n-1}, \chi_{n-2}, \dots, \chi_0]^T,$$

where χ_j is the coefficient of $u_k \cdot v$ with respect to Z^j. Expanding $u_k \cdot v$, we get

$$\chi_j = \begin{cases} 0 & \text{if } j = 0 \\ \sum_{i=0}^{j} y_{n-i} x_{k-j+i}, & \text{if } j \le k-1, \\ \sum_{i=1}^{k-1} y_{n+1-j} x_{k-i}, & \text{if } k \le j < n. \end{cases}$$

The coefficient subvector

$$\nu = JL_2.[x_{k-1}, x_{k-2}, \dots, 0, 0]^T, \tag{4}$$

where $[x_{k-1}, x_{k-2}, \dots, 0, 0]^T$ is the k-th column of R_2. According to (3) and (4), it can be verified that the column γ_k is able to be represented by the coefficients of $(1/x_n) \cdot (\phi \cdot g_k - u_k \cdot v)$. $\square$

Corollary 1 *Given a nonsingular Hankel matrix* H*, suppose* $\mathbf{x}$ *and* $\mathbf{y}$ *are the solutions of the equations of* $H\mathbf{x} = e_1$ *and* $H\mathbf{y} = e_n$*, then*

$$\|\mathbf{x}\|_1 \le \|H^{-1}\|_1 \le \frac{2\|\mathbf{x}\|_1 \cdot \|\mathbf{y}\|_1}{|x_n|}. \tag{5}$$

PROOF. Since $x_n = y_1$, (5) can be concluded from Theorem 2. $\square$

Given a strongly regular matrix H, we discuss how to use the bound in (5) to estimate $\|(H^{[i]})^{-1}\|_1$ for all the leading principal submatrices in quadratic time. A linear time recursive algorithm is presented in [14] to obtain the solution vector of $H^{[i+1]}\mathbf{x}_{i+1} = e_{i+1}$ from the solution vector of $H^{[i]}\mathbf{y}_i = e_i$, where $H^{[i]}$ and $H^{[i+1]}$ are the $i \times i$ and $(i+1) \times (i+1)$ leading principal submatrices of H respectively, and $e_i = [0, \dots, 0, 1]^T$, $e_{i+1} = [0, e_i]^T$. Moreover, according to the Lemma in [14], for an $i \times i$ strongly regular Hankel matrix H, the solution $\mathbf{x}$ of the linear system $H\mathbf{x} = e_1$ can be obtained in $O(i)$ flops if one has the solution $H\mathbf{y} = e_i$. In other words, for a given strongly regular $k \times k$ Hankel matrix H, all of the solution vectors $\mathbf{x}$ and $\mathbf{y}$ of $H^{[i]}\mathbf{x} = e_1, H^{[i]}\mathbf{y} = e_i, i = 1, 2, \dots, k$ can be obtained in $O(k^2)$ flops. Hence the recursive algorithm presented in [14] is applicable to obtain the bounds (5) of all $\|(H^{[i]})^{-1}\|_1$ in quadratic time.

Theorem 3 *Under the same assumptions as above, given a strongly regular* $n \times n$ *Hankel matrix* H*, we have*

$$\|\mathbf{x}^{[i]}\|_1 \|H^{[i]}\|_1 \le \kappa_1(H^{[i]}) \le \frac{2\|\mathbf{x}^{[i]}\|_1 \|\mathbf{y}^{[i]}\|_1}{|x_i^{[i]}|} \|H^{[i]}\|_1, \tag{6}$$

where $\mathbf{x}^{[i]}$ *and* $\mathbf{y}^{[i]}$ *are the solution vectors of* $H^{[i]}\mathbf{x}^{[i]} = e_1$ *and* $H^{[i]}\mathbf{y}^{[i]} = e_i$*, and* $x_i^{[i]}$ *denotes the last element of* $\mathbf{x}^{[i]}$*. Furthermore, computing the lower bounds and the upper bounds (6) of all* $\kappa_1(H^{[i]})$*,* $1 \le i \le n$*, can be achieved in* $O(n^2)$ *arithmetic operations.*

PROOF. We conclude (6) from Corollary 1. As discussed, $\|H^{[i]}\|_1$ and the bounds of $\|(H^{[i]})^{-1}\|_1$ can be obtained in quadratic time, which means that the computation of the condition number bounds (6) is $O(n^2)$. $\square$

3. EARLY TERMINATION IN NUMERICAL SPARSE INTERPOLATION

We apply our method to the problem of early termination in numerical sparse polynomial interpolation. Consider a black box univariate polynomial $f \in \mathbb{C}[x]$ represented as

$$f(x) = \sum_{j=1}^{t} c_j x^{d_j}, \quad 0 \ne c_j \in \mathbb{C} \tag{7}$$

and $d_j \in \mathbb{Z}_{\ge 0}, d_1 < d_2 < \cdots < d_t$.

Suppose the evaluations of $f(x)$ contain added noise. Let δ be a degree upper bound of f, which means, $\delta \ge \deg(f) = d_t$. Our aim is to determine the number of terms t in the target polynomial f.

In exact arithmetic, the early termination strategy [18] can determine the number of terms with high probability. Choose a random element ω from a sufficiently large finite set and evaluate $f(x)$ at the powers of ω

$$h_0 = f(\omega), h_1 = f(\omega^2), h_2 = f(\omega^3), \dots.$$

Consider a $(\delta+1) \times (\delta+1)$ Hankel matrix $H = (H)_{i,j} = h_{i+j-2}$ as in (1). The early termination algorithm makes, with high probability, all principal minors $H^{[j]}$ non-singular for $1 \le j \le t$ [18]. Moreover, $H^{[j]}$ must be singular for all $t < j \le \delta$. Hence the number of terms t is detected as follows: for $j = 1, 2, \dots$, the first time $H^{[j]}$ becomes singular is when $j = t + 1$.

In the numerical setting, such a singular matrix is often extremely ill-conditioned. So we need to measure the singularity for a given numerical Hankel matrix. In [22], it is proved that for a Hankel matrix the structured condition number with respect to the spectral norm is equivalent to the regular condition number. In other words, for a given Hankel matrix $H^{[j]}$, the distance measured by spectral norm to the nearest singular Hankel matrix is equivalent to $1/\|(H^{[j]})^{-1}\|_2$. In the following we investigate how to estimate the spectral norm of the inverse of a Hankel matrix.

Theorem 4 *Using the same quantities as above, we have*

$$\frac{\|\mathbf{x}\|_1}{\sqrt{n}} \le \|H^{-1}\|_2 \le \frac{2\|\mathbf{x}\|_1 \cdot \|\mathbf{y}\|_1}{|x_n|}. \tag{8}$$

Proof. Given a $n \times n$ arbitrary matrix A, we have the following bound of $\|A\|_2$ [16]

$$\frac{\max\{\|A\|_1, \|A\|_\infty\}}{\sqrt{n}} \le \|A\|_2 \le \sqrt{\|A\|_1\|A\|_\infty}.$$

According to Corollary 1 and $\|H^{-1}\|_\infty = \|H^{-1}\|_1$, it can be verified that $\|H^{-1}\|_2$ satisfies (8). $\square$

We following the bound of $\|H^{-1}\|_2$ in (8) and present a recursive algorithm to detect the sparsity of the interpolating polynomial. Suppose we are given an approximate black box polynomial of f represented as (7), its degree bound δ and a chosen tolerance τ, our aim is to recover the number of terms t. We proceed our recursive algorithm as the following. First choose a random root of unity ω with a prime order $p \ge \delta$ and obtain the approximate evaluations

$$h_\ell \approx f(\omega^{\ell+1}), \quad \ell = 0, 1, 2, \ldots.$$

Then we compute iteratively for $k = 1, 2, \ldots$ and $H^{[k]} = [h_{i+j-2}]$ the vectors $\mathbf{x}^{[k]}$ and $\mathbf{y}^{[k]}$ as stated in [14], where $\mathbf{x}^{[k]}$ and $\mathbf{y}^{[k]}$ are the solutions of $H^{[k]}\mathbf{x}^{[k]} = e_1$ and $H^{[k]}\mathbf{y}^{[k]} = e_k$. We break out of the loop until

$$\frac{|x_k^{[k]}|}{2\|\mathbf{x}^{[k]}\|_1 \cdot \|\mathbf{y}^{[k]}\|_1} \le \tau,$$

which means that for all $j = 1, 2, \ldots, k-1$ the distance between $H^{[j]}$ and the nearest singular Hankel matrix is greater than the given tolerance τ. This is because

$$\frac{1}{\|(H^{[k]})^{-1}\|_2} \ge \frac{|x_k^{[k]}|}{2\|\mathbf{x}^{[k]}\|_1 \cdot \|\mathbf{y}^{[k]}\|_1} > \tau.$$

At this stage, we claim that $H^{[k-1]}$ is strongly regular and obtain $t = k-1$ as the number of the terms in f.

In sparse polynomial interpolation, the associated Hankel matrix $H^{[t]}$ has additional structure that allows a factorization

$$H^{[j]} = \begin{bmatrix} h_0 & h_1 & \ldots & h_{j-1} \\ h_1 & h_2 & \ldots & h_j \\ \vdots & \vdots & \ddots & \vdots \\ h_{j-1} & h_j & \ldots & h_{2j-2} \end{bmatrix}$$
$$= \underbrace{\begin{bmatrix} 1 & 1 & \ldots & 1 \\ b_1 & b_2 & \ldots & b_t \\ \vdots & \vdots & \ddots & \vdots \\ b_1^{j-1} & b_2^{j-1} & \ldots & b_t^{j-1} \end{bmatrix}}_{V^{[j]}} \underbrace{\begin{bmatrix} c_1b_1 & 0 & \ldots & 0 \\ 0 & c_2b_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & c_tb_t \end{bmatrix}}_{D}$$
$$\times \underbrace{\begin{bmatrix} 1 & b_1 & \ldots & b_1^{j-1} \\ 1 & b_2 & \ldots & b_2^{j-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & b_t & \ldots & b_t^{j-1} \end{bmatrix}}_{(V^{[j]})^{\mathrm{T}}}, \tag{9}$$

in which $b_k = \omega^{d_k}$ for $1 \le k \le t$.

When $j = t$, the condition of the associated Hankel system is linked to the embedded Vandermonde system [11, 12]

$$\|(V^{[t]})^{-1}\|^2 \cdot \max_j \frac{1}{|c_j|} \ge \|(H^{[t]})^{-1}\| \ge \frac{\|(V^{[t]})^{-1}\|^2}{\sum_{1\le j\le t}|c_j|} \tag{10}$$

(see Proposition 4.1 in [12].)

Since all b_j are on the unit circle, according to [9] the inverse of the Vandermonde matrix $\|V^{-1}\|$ can be bounded by

$$\begin{aligned} \|(V^{[t]})^{-1}\| &\le \max_{1\le j\le t} \prod_{j=1, j\ne k}^{t} \frac{1+|b_j|}{|b_j - b_k|} \\ &= \max_{1\le j\le t} \frac{2^{t-1}}{\prod_{j\ne k}|b_j - b_k|}. \end{aligned} \tag{11}$$

As for the lower bound, $\|(V^{[t]})^{-1}\|_\infty = 1$ achieves the optimal condition if b_j are evenly distributed on the unit circle (see Example 6.4 in [10].)

In [11, 12], the sparsity t is given as input. So the randomization is used to improve the condition of the overall sparse interpolation problem, where the recovery of both the non-zero terms and the corresponding coefficients depend on the condition of the embedded Vandermonde system. While [11, 12] further exploit a generalized eigenvalue reformulation [15] in sparse interpolation, interestingly the condition of the associated generalized eigenvalue problem is still dependent on the condition of the embedded Vandermonde system [12, Theorem 4.2] (see [3] for further analysis and discussion.)

Now our purpose now to detect the number of terms. In other words, we intend to use the estimated conditions to determine the number of terms t. (After t is determined, for polynomial interpolation one still needs to recover the t exponents $d_1, \ldots d_t$ and the associated coefficients $c_1, \ldots, c_t$ in f.)

Recall that in exact arithmetic, $H^{[j]}$ is singular for $j > t$. In a numerical setting, such $H^{[j]}$ is expected to be extremely ill-conditioned. If $H^{[t]}$ is relatively well-conditioned, then one can expect a surge in the condition number for $H^{[t+1]}$. We apply the randomization idea in [11, 12] to obtain the heuristic of achieving relatively well-conditioned $H^{[t]}$ with high probability.

Following (11), the distribution of $b_1, \ldots, b_t$ on the unit circle affects the upper bound on the condition of the Vandermonde system $V^{[t]}$. Let $b_j = e^{2\pi i d_j/p}, b_k = e^{2\pi i d_k/p}$ for a prime $p \ge \delta$. Both b_j and b_k are on the unit circle. Let $\Delta_{jk} = |d_j - d_k| \ge 1$, the distance between b_j and b_k is

$$\begin{aligned} |b_j - b_k| &= \sqrt{(1-\cos(2\pi\Delta_{jk}/p))^2 + \sin^2(2\pi\Delta_{jk}/p)} \\ &= \sqrt{2 - 2\cos(2\pi\Delta_{jk}/p)}. \end{aligned}$$

For a fixed t, an upper bound of $\|(V^{[t]})^{-1}\|^2$ is determined by

$$\min_{1\le j\le t} \prod_{j\ne k} \left(2 - 2\cos\left(\frac{2\pi\Delta_{jk}}{p}\right)\right), \tag{12}$$

in which p is a chosen prime order for the primitive roots of unity.

According to (12), the corresponding Hankel system $H^{[t]} = V^{[t]}D(V^{[t]})^{\mathrm{T}}$ can be better conditioned if

1. p is smaller; and/or

2. the most (or all) of Δ_{jk} do not belong to smaller values.

In order to achieve a better condition, choosing a larger p would require larger Δ_{jk} and may require a higher precision. Thus we prefer a smaller p. Moreover, in general a smaller p does not affect too much on the overall random distribution of b_j on the unit circle (see Example 3 in Section 4.)

Remark: In the exact case, it may still happen that $H^{[j]}$ is singular for $j \leq t$, which corresponds to an ill-conditioned $H^{[j]}$ for $j \leq t$ in a finite precision environment. Therefore we perform our algorithm ζ times in the numerical setting. Given $\zeta \in \mathbb{Z}_{>0}$, we choose different random roots of unity $\omega_1, \omega_2, \ldots, \omega_\zeta$. For each j we perform our algorithm on $f(\omega_j^k)$ for $k = 1, 2, \ldots$ and obtain the number of terms as t_j. At the end, we determine $t = \max\{t_1, t_2, \ldots, t_\zeta\}$ as the number of the terms of f.

4. EXPERIMENTS

Our numerical early termination is tested for determining the number of the terms. We set *Digits* := 15 in Maple 13. Our test polynomials are constructed with random integer coefficients in the range between -10 and 10, and with a random degree and random terms in the given ranges.

For each given noise range, we test 50 random black box polynomials and report the times of failing to correctly estimate the number of the terms. In Table 1, *Deg. Range* and *Term Range* record the range of the degree and the number of the term in the polynomials; *Random Noise* records range of random noise added to the evaluations of the black box polynomial; *Fail* reports the times of failing to estimate the correct number of the terms in the target polynomials.

Ex.	*Random Noise*	*Term Range*	*Deg. Range*	*Fail*
1	$10^{-6} \sim 10^{-5}$	$10 \sim 15$	$100 \sim 150$	3
2	$10^{-7} \sim 10^{-6}$	$15 \sim 20$	$100 \sim 150$	1
3	$10^{-8} \sim 10^{-7}$	$20 \sim 25$	$100 \sim 150$	1
4	$10^{-9} \sim 10^{-8}$	$20 \sim 25$	$100 \sim 150$	1

Table 1: Numerical early termination: number of failures out of 50 random polynomials.

Example 1 Given a polynomial f with $\deg(f) = 100$ and the number of term 20, denoted by $T(f) = 20$. Its coefficients are randomly generated as integers between -10 and 10. We construct the black box that evaluates f with added random noises in the range $10^{-9} \sim 10^{-8}$. From such black box evaluations, we generated 1000 random Hankel matrices and compare the condition number $\kappa_1(H) = \|H\|_1 \cdot \|H^{-1}\|_1$ with their corresponding lower bounds $\kappa_{\text{low}}(H)$ and the upper bound $\kappa_{\text{up}}(H)$, shown in (6).

Let $\omega_j = \exp(2 s_j \pi \boldsymbol{i}/p_j) \in \mathbb{C}$ be random roots of unity, where $\boldsymbol{i} = \sqrt{-1}$, p_j prime numbers in the range $100 \leq p_j \leq 1000$, and s_j random integers with $1 \leq s_j < p_j$. We compute the evaluations of the black box for different root of unity ω_j, and construct the associated Hankel matrix $H^{[j]}$. Since $T(f) = 20$, we do 41 evaluations for each j to construct the Hankel matrix with $Dim(H^{[j]}) = 21$:

$$h_{j,l} \approx f(\omega_j^l) \in \mathbb{C}, \quad l = 1, 2, 3, \ldots, 41.$$

We use $H_j^{[k]}$ to denote the $k \times k$ leading principle submatrix of H_j and let

$$r_{\text{low}} = \frac{\|H\|_1 \cdot \|H^{-1}\|_1}{\kappa_{\text{low}}(H)}, \quad r_{\text{up}} = \frac{\kappa_{\text{up}}(H)}{\|H\|_1 \cdot \|H^{-1}\|_1}.$$

For $j = 1, 2, \ldots, 1000$, we obtain the ratios r_{low} and r_{up} for all $k \times k$ leading principle submatrices $H_j^{[k]}$, where $k = 1, 2, \ldots, 20$. Therefore, r_{low} and r_{up} are used to measure whether the lower and upper bounds are close to the actual condition number. We use the histogram to measure the ratios r_{low} and r_{up}. For instance, in Figure 1 (a) and (b) show the distribution of the ratios r_{low} and r_{up} for all 5×5 leading principle submatrices $H_j^{[5]}, j = 1, 2, \ldots, 1000$. In Figure 2, (a) and (b) show the distribution of the ratios r_{low} and r_{up} of all 20×20 leading principle submatrices $H_j^{[20]}, j = 1, 2, \ldots, 1000$.

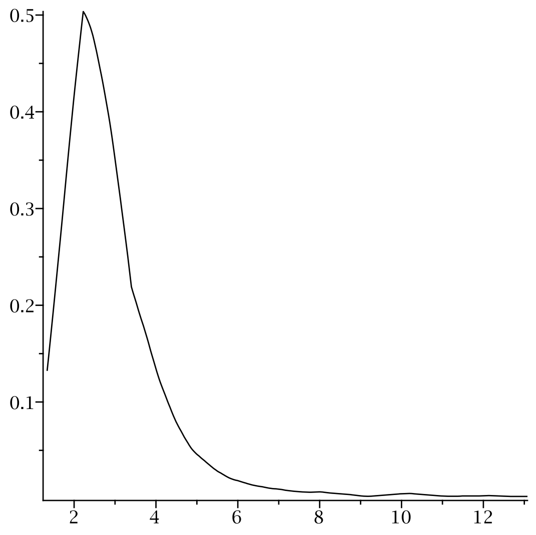
Example 2 We performed $m = 100$ random $n \times n$ Hankel matrices for $n = 4, 8, \ldots, 1024$ whose entries are randomly chosen and follow uniformly probabilistic distribution over $\{c \mid -1 \leq c \leq 1\}$. Table 2 displays the average value of r_{low} and r_{up}. Here n is the dimension of random Hankel matrices; r_{low} and r_{up} are the average ratios of $\frac{\|H\|_1 \cdot \|H^{-1}\|_1}{\kappa_{\text{low}}(H)}$ and $\frac{\kappa_{\text{up}}(H)}{\|H\|_1 \cdot \|H^{-1}\|_1}$; $\kappa_1 = \|H\|_1 \cdot \|H^{-1}\|_1$ is the average value of the actual condition number obtained for *Digits* := 15 in Maple 13. From the columns of r_{low} and r_{up}, we observe that the upper bound of the condition number depends on the dimension of the Hankel matrices. But the lower bound of the condition number does not seem to be affected much by the increasing of the dimension.

Ex.	n	r_{low}	r_{up}	$\kappa_1(H)$
1	4	3.1431	17.402	523.10
2	8	4.7675	14.023	69.965
3	16	6.4865	30.349	136.59
4	32	11.154	106.35	734.55
5	64	15.691	107.19	1996.1
6	128	24.120	220.42	13218
7	256	34.496	549.59	16932
8	512	49.425	1516.1	16783
9	1024	71.394	1385.0	111375

Table 2: Algorithm performance on condition number estimate

Finally, following the discussion on (10), (11), (12) in Section 3, we investigate the role of p in the conditioning of $H^{[t]}$.

Example 3 For each try, we choose $m = 10$ random polynomials with $50 \leq \deg(f) \leq 100$. The coefficients are randomly generated as integers between -5 and 5. The number of the terms of f is t, shown in the 3-th column of Table 3. In Table 1, *Term Range* records the range of random noise added to the evaluations of the black box polynomial; $\kappa_1(H_1{}^{[t]})$ and $\kappa_1(H_2{}^{[t]})$ denote the respective condition numbers of the $t \times t$ Hankel matrices $H_1{}^{[t]}$ and $H_2{}^{[t]}$. The matrices $H_1{}^{[t]}$ and $H_2{}^{[t]}$ are constructed by the approximate evaluations of the root of unity with orders corresponding to


(a) Distribution $r_{\text{low}} = \dfrac{\|H\|_1 \cdot \|H^{-1}\|_1}{\kappa_{\text{low}}(H)}$

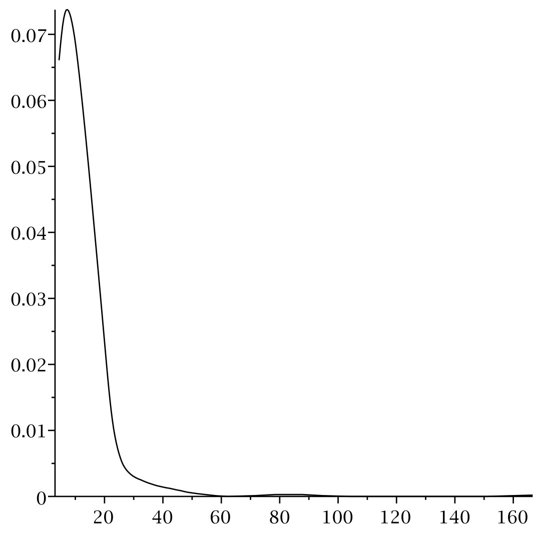
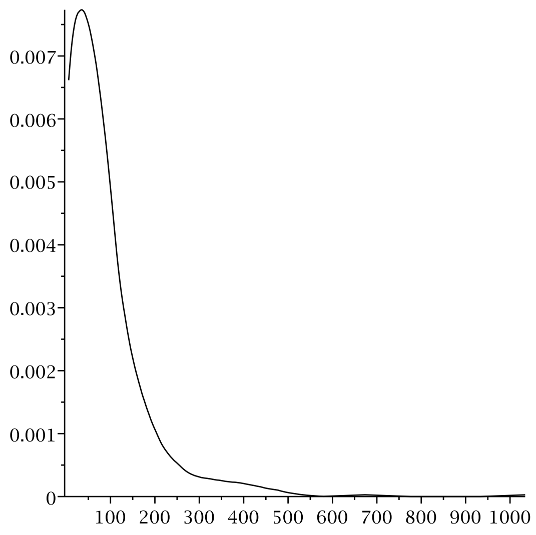

(b) Distribution $r_{\text{up}} = \dfrac{\kappa_{\text{up}}(H)}{\|H\|_1 \cdot \|H^{-1}\|_1}$

Figure 1: Distribution of the leading principle matrix $H^{[5]}$


(a) Distribution $r_{\text{low}} = \dfrac{\|H\|_1 \cdot \|H^{-1}\|_1}{\kappa_{\text{low}}(H)}$

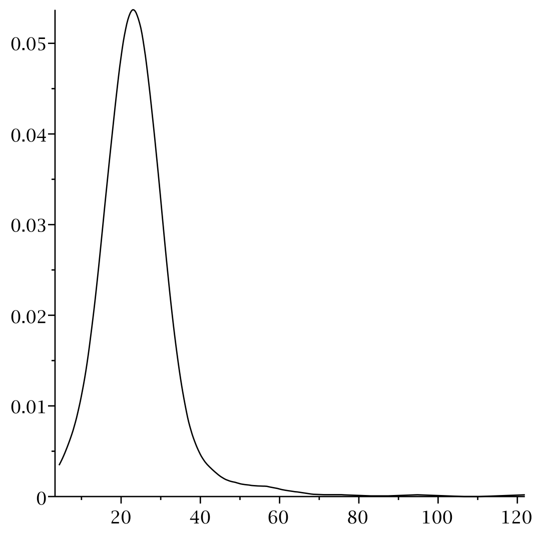

(b) Distribution $r_{\text{up}} = \dfrac{\kappa_{\text{up}}(H)}{\|H\|_1 \cdot \|H^{-1}\|_1}$

Figure 2: Distribution of the leading principle matrix $H^{[20]}$

the randomly chosen prime numbers p_1 and p_2, where p_1 is between 100 and 1000 and p_2 between 10^5 and 10^7.

Table 3 agrees with our discussion at the end of Section 3. When the number of terms t is fixed, the various scales of p_1 and p_2 do not seem to greatly affect the distribution of terms on the unit circle. But a larger p_2 may demand a higher precision hence cause additional computation problems.

5. CONCLUSION

By example of a randomized sparse interpolation algorithm, we have investigated a central question in algorithms with floating point arithmetic and imprecise date posed in [20]: how does one analyze the probability of success, in our case the correct determination of the discrete polynomial sparsity? Bad random choices may result in ill-conditioned intermediate structured matrices at the wrong place. Even if one can detect such ill-conditionedness, as we have via the Gohberg-Semencul formula, the distribution of such occurrences must be controlled. We do so by running multiple random execution paths next to one another, which successfully can weed out bad random choices, as we have observed experimentally. The mathematical justification requires an understanding of condition numbers of random Fourier matrices, which we have presented, and of additional instabilities that are caused by substantial noise in the data, which we have demonstrated, at least by experiment, to be con-

Ex.	*Random Noise*	t	$\kappa_1(H_1^{[t]})$	$\kappa_1(H_2^{[t]})$
1	$10^{-6} \sim 10^{-5}$	5	1.16×10^5	1.61×10^5
2	$10^{-6} \sim 10^{-5}$	8	9.54×10^5	2.91×10^6
3	$10^{-8} \sim 10^{-7}$	10	4.14×10^7	1.02×10^8
4	$10^{-6} \sim 10^{-5}$	12	1.51×10^6	2.04×10^6
5	$10^{-6} \sim 10^{-5}$	15	1.13×10^8	3.53×10^8
6	$10^{-7} \sim 10^{-6}$	12	5.24×10^7	3.92×10^7

Table 3: Condition numbers for $H^{[t]}$ constructed with roots of unity of different size prime orders p_1 and p_2.

trollable by our algorithms.

Acknowledgments

We thank anonymous referees for their helpful comments and information.

6. REFERENCES

[1] B. Beckermann. The stable computation of formal orthogonal polynomials. *Numerical Algorithms*, 11(1):1–23, 1996.

[2] B. Beckermann, S. Cabay, and G. Labahn. Fraction-free computation of matrix Pade systems. In W. Küchlin, editor, *Proc. 1997 Internat. Symp. Symbolic Algebraic Comput. (ISSAC'97)*, pages 125–132, 1997.

[3] B. Beckermann, G. H. Golub, and G. Labahn. On the numerical condition of a generalized Hankel eigenvalue problem. *Numerische Mathematik*, 106(1):41–68, 2007.

[4] B. Beckermann and G. Labahn. A fast and numerically stable euclidean-like algorithm for detecting relatively prime numerical polynomials. *Journal of Symbolic Computation*, 26(6):691–714, 1998.

[5] B. Beckermann and G. Labahn. When are two numerical polynomials relatively prime? *Journal of Symbolic Computation*, 26(6):677–689, 1998.

[6] B. Beckermann and G. Labahn. Effective computation of rational approximants and interpolants. *Reliable Computing*, 6(4):365–390, 2000.

[7] R. P. Brent, F. G. Gustavson, and D. Y. Y. Yun. Fast solution of Toeplitz systems of equations and computation of Padé approximants. *J. Algorithms*, 1:259–295, 1980.

[8] S. Cabay and R. Meleshko. A weakly stable algorithm for Padé approximants and the inversion of Hankel matrices. *SIAM Journal on Matrix Analysis and Applicatins*, 14(3):735–738, 1993.

[9] W. Gautschi. On inverse of Vandermonde and confluent Vandermonde matrices. *Numerische Mathematik*, 4:117–123, 1962.

[10] W. Gautschi. Norm estimates for inverse of Vandermonde matrices. *Numerische Mathematik*, 23:337–347, 1975.

[11] M. Giesbrecht, G. Labahn, and W. Lee. Symbolic-numeric sparse interpolation of multivariate polynomials (extended abstract). In J.-G. Dumas, editor, *ISSAC MMVI Proc. 2006 Internat. Symp. Symbolic Algebraic Comput.*, pages 116–123. ACM Press, 2006.

[12] M. Giesbrecht, G. Labahn, and W. Lee. Symbolic-numeric sparse interpolation of multivariate polynomials. *Journal of Symbolic Computation*, 44:943–959, 2009.

[13] M. Giesbrecht and D. Roche. Diversification improves interpolation. In A. Leykin, editor, *ISSAC 2011 Proc. 2011 Internat. Symp. Symbolic Algebraic Comput.* ACM Press, 2011. to appear.

[14] I. Gohberg and I. Koltracht. Efficient algorithm for Toeplitz plus Hankel matrices. *Integral Equations and Operator Theory*, 12:136–142, 1989.

[15] G. H. Golub, P. Milanfar, and J. Varah. A stable numerical method for inverting shape from moments. *SIAM Journal on Scientific Computing*, 21(4):1222–1243, 1999.

[16] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, Maryland, third edition, 1996.

[17] T. Huckle. Computations with Gohberg-Semencul formulas for Toeplitz matrices. *Linear Algebra and Applications*, 271:169–198, 1998.

[18] E. Kaltofen and W. Lee. Early termination in sparse interpolation algorithms. *Journal of Symbolic Computation*, 36(3–4):365–400, 2003. URL: http://www.math.ncsu.edu/~kaltofen/bibliography/03/KL03.pdf.

[19] E. Kaltofen, Z. Yang, and L. Zhi. Approximate greatest common divisors of several polynomials with linearly constrained coefficients and singular polynomials. In J.-G. Dumas, editor, *ISSAC MMVI Proc. 2006 Internat. Symp. Symbolic Algebraic Comput.*, pages 169–176, New York, N. Y., 2006. ACM Press. URL: http://www.math.ncsu.edu/~kaltofen/bibliography/06/KYZ06.pdf.

[20] E. Kaltofen, Z. Yang, and L. Zhi. On probabilistic analysis of randomization in hybrid symbolic-numeric algorithms. In S. M. Watt and J. Verschelde, editors, *SNC'07 Proc. 2007 Internat. Workshop on Symbolic-Numeric Comput.*, pages 11–17, 2007. URL: http://www.math.ncsu.edu/~kaltofen/bibliography/07/KYZ07.pdf.

[21] E. Kaltofen and G. Yuhasz. A fraction free matrix Berlekamp/Massey algorithm, Feb. 2009. Manuscript, 17 pages.

[22] S. M. Rump. Structured perturbations part I: Normwise distances. *SIAM Journal on Matrix Analysis and Applications*, 25(1):1–30, 2003.

[23] W. F. Trench. An algorithm for the inversion of finite Hankel matrices. *Journal of the Society for Industrial and Applied Mathematics*, 13(4):pp. 1102–1107, 1965.

Implicitization of curves and surfaces using predicted support

Ioannis Z. Emiris
University of Athens
Athens, Greece
emiris@di.uoa.gr

Tatjana Kalinka
University of Athens
Athens, Greece
tatjana.kalinka@gmail.com

Christos Konaxis
University of Athens
Athens, Greece
ckonaxis@di.uoa.gr

ABSTRACT

We reduce implicitization of rational parametric curves and (hyper)surfaces to linear algebra, by interpolating the coefficients of the implicit equation. For this, we may use any method for predicting the implicit support. We focus on methods that exploit input structure in the sense of sparse (or toric) elimination theory, namely by computing the Newton polytope of the implicit polynomial. We offer a public-domain implementation of our methods, and study their numerical stability and efficiency on several classes of plane curves and surfaces, and discuss how it can be used for approximate implicitization in the setting of sparse elimination.

Categories and Subject Descriptors

G.1 [**Mathematics of Computing**]: Numerical Analysis; I.1 [**Computing Methodologies**]: Symbolic and Algebraic Manipulation; I.3.5 [**Computing Methodologies**]: Computer Graphics—*Computational Geometry and Object Modeling*

General Terms

Algorithms, Experimentation

Keywords

Implicitization, Sparse elimination, Newton polytope, Numerical linear algebra

1. INTRODUCTION

Implicitization is the problem of changing the representation of parametric objects to implicit (or Cartesian) form. This problem lies at the heart of several questions in computer-aided geometric design (CAGD) and geometric modeling, including intersection problems and membership queries. In several situations, it is important to have both representations available. Implicit representations encompass a larger class of shapes than parametric ones. Moreover, the class of implicit curves and surfaces is closed under certain operations such as offsetting, while its parametric counterpart is not. Implicitization is also of independent interest, since certain questions in areas as diverse as robotics or statistics, e.g. [4], reduce to deriving the implicit form.

Here we follow a classical symbolic-numeric method, which reduces implicitization to interpolating the coefficients of the defining equation. We implement interpolation by (numeric) linear algebra operations, following a symbolic phase of implicit support prediction, i.e. computing a (super)set of the monomials appearing in the implicit equation. Standard methods to interpolate the unknown coefficients by linear algebra, are divided in two main categories, dense and sparse methods. The former require a bound on the total degree of the target polynomial, whereas the latter require only a bound on the number of its terms, thus exploiting any sparseness of the target polynomial. Moreover, its performance depends on the actual number of non-zero terms in this polynomial. In fact, a priori knowledge of the support essentially answers the first task of such a method. For more information see [25].

Our first contribution is to exploit sparse (or toric) variable elimination theory to predict the implicit Newton polytope, i.e. the convex hull of the implicit support.

Definition 1. *Given a polynomial $\sum_j c_{ij} t^{a_{ij}}$, its support is the set $A_i = \{a_{ij} \in \mathbb{N}^n : c_{ij} \neq 0\}$; its Newton polytope is the convex hull of the support.*

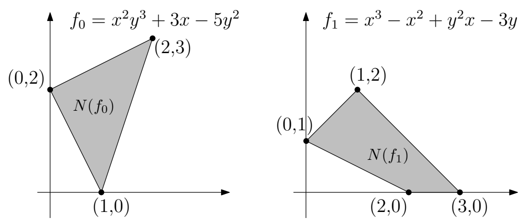


Figure 1: Example of Newton polygons

One reason for revisiting interpolation is the current increase of activity around various approaches capable of predicting the implicit support, e.g. [5, 6, 21, 22]. Although our team has been focussing on sparse elimination [9, 11, 12], the present work can use the implicit support predicted by any method.

In fact, [22, sec.4] states that *"Knowing the Newton polytopes reduces computing the [implicit] equation to numerical linear algebra. The numerical mathematics of this problem is interesting and challenging [...] "* We expect that our software will interface implicit support predictors, such as those developed in [5, 9, 11, 22], with linear algebra. We juxtapose the use of exact and numerical linear algebra, which eventually can rely on state-of-the-art libraries, such as Eigen or LAPACK, respectively.

We discuss approximate implicitization, which is of high practical importance in CAGD [8, 19], in the setting of sparse elimination. In our view, approximate implicitization is one of the main motivations for reducing implicitization to interpolation.

Our second contribution is to offer a public-domain Maple implementation. We study the numerical stability and efficiency of our algorithms on several classes of 2- and 3-d examples. The Maple code has been published on our Wiki webpage that contains support prediction and implicitization related experiments:

http://ergawiki.di.uoa.gr/index.php/Implicitization

One central question is how to evaluate the computed monomials to obtain a suitable matrix, when performing exact or numerical matrix operations. We compare results obtained by using random integers, random complex unitary numbers and complex roots of unity.

A *parametrization* of a geometric object of co-dimension one, in a space of dimension $n+1$, can be described by a set of parametric functions:

$$x_0 = f_0(t_1, \ldots, t_n), \ldots, x_n = f_n(t_1, \ldots, t_n),$$

where $t := (t_1, t_2, \ldots, t_n)$ is the vector of parameters and $f := (f_0, \ldots, f_n)$ is a vector of continuous functions, including polynomial, rational, and trigonometric functions, also called *coordinate functions.* These are defined on some product of intervals $\Omega := \Omega_1 \times \cdots \times \Omega_n,\ \Omega_i \subseteq \mathbb{R}^n$, of values of $t_1, \ldots, t_n$. Implicitization of planar curves and surfaces in 3-dimensional space corresponds to $n = 1$ and $n = 2$ respectively.

The *implicitization problem* asks for the smallest algebraic variety containing the closure of the image of the parametric map $f : \mathbb{R}^n \to \mathbb{R}^{n+1} : t \mapsto f(t)$. This image is contained in the variety defined by the ideal of all polynomials p s.t. $p(f_0(t), \ldots, f_n(t)) = 0$, for all t in Ω. We restrict ourselves to the case when this is a principal ideal, and we wish to compute its defining polynomial

$$p(x_0, \ldots, x_n) = 0, \tag{1}$$

given its Newton polytope or a polytope that contains it. We can regard the variety in question as the projection of the graph of map f to the last $n+1$ coordinates. If f is polynomial, implicitization is reduced to eliminating t from the polynomial system

$$F_i := x_i - f_i(t) \in (\mathbb{R}[x_i])[t],\ i = 0, \ldots, n,$$

seen as polynomials in t with coefficients which are functions of the x_i. This is also the case for rational parameterizations

$$x_i = \frac{f_i(t)}{g_i(t)},\ i = 0, \ldots, n, \tag{2}$$

which can be represented as polynomials

$$F_i := x_i g_i(t) - f_i(t) \in (\mathbb{R}[x_i])[t],\ i = 0, \ldots, n, \tag{3}$$

where we have to take into account that the $g_i(t)$ cannot vanish.

Several algorithms exist for this problem, including methods based on resultants, Gröbner bases, moving surfaces, and residues. Our approach relies on any support prediction method, see sect. 3. We focus on sparse (or toric) elimination: In the case of curves, the implicit support is directly determined for generic parametric expressions with the same supports. In the case of (hyper)surfaces, the implicit support is provided by that of a sparse resultant.

The rest of the paper is structured as follows. Previous work is discussed in the next two sections. Sect. 2 discusses existing methods for interpolating the implicit polynomial by linear algebra. Sect. 3 discusses implicit support prediction and sparse elimination. Our algorithm is detailed in sect. 4, and its use in experiments is analyzed in sect. 5. We conclude with open questions in sect. 6, whereas the Appendix contains further experimental results.

2. EXISTING INTERPOLATION METHODS

This section examines how implicitization had been reduced to a linear algebra question. Let S be (a superset of) the support of the implicit polynomial $p(x_0, \ldots, x_n) = 0$ with unknown coefficients P, $|P| = |S|$. We sometimes refer to S as implicit support, with the understanding that, later, interpolation may set some of the corresponding coefficients to zero.

2.1 Exact implicitization

The most direct method to reduce implicitization to linear algebra is to construct a $|S| \times |S|$ matrix M, indexed by S (columns) and $|S|$ different values (rows) at which all monomials get evaluated. Then, P is in the kernel of M. This idea was used in [12, 17, 22].

In [21], they propose evaluation at unitary $\tau \in (\mathbb{C}^*)^n$, i.e., of modulus 1. This is one of the evaluation strategies examined below. Another approach was described in [2], based on integration of matrix $M = SS^T$, over each parameter $t_1, \ldots, t_n$. Then, P is in the kernel of M. In fact, the authors propose to consider successively larger supports in order to capture sparseness. This method covers a wide class of parameterizations, including polynomial, rational, and trigonometric representations. The resulting matrix has Henkel-like structure [16]. When it is computed over floating-point numbers, the resulting implicit polynomial does not necessarily have integer coefficients. This is the situation we face when using numerical methods, such as SVD, see subsec. 4.2. In [2], they discuss some extra processing to yield the integer relations among the coefficients; we also use a similar method.

2.2 Approximate implicitization

In practical applications of CAGD, precise implicitization often can be impossible or very expensive to obtain. Approximate implicitization over floating-point numbers appears to be an effective solution. There are direct [8, 24] and iterative techniques [1].

We describe the basic direct method [8]. Given a parametric (spline) curve or surface $x(t),\ t \in \Omega \subset \mathbb{R}^n$, the goal of [8] is to find polynomial $q(x)$ such that

$$q(x(t) + \eta(t)g(t)) = 0,$$

where $g(t)$ is a continuous direction function with Euclidean norm $\|g(t)\| = 1$ and $\eta(t)$ a continuous error function with $|\eta(t)| \leq \epsilon$. Now,

$$q(x(t)) = (Mp)^T \alpha(t),$$

where matrix M is built from monomials in x. It may be constructed as in our case, or it may contain a subset of the monomials comprising the exact implicit support. Here, p is the vector of implicit coefficients, hence $Mp = 0$ returns the exact solution. Otherwise, $\alpha(t)$ is the basis of the space of polynomials that describes $q(x(t))$, and is assumed to form a partition of unity: $\sum \alpha_i = 1$, and to be nonnegative over Ω:

$$\alpha_i \geq 0, \ \forall i, \forall t \in \Omega.$$

One may use the Bernstein-Bézier basis with respect to the interval Ω, in the case of curves, or a triangle which contains Ω, in the case of surfaces.

In [8, p.176] the authors propose to translate to the origin and scale the parametric object, so as to lie in $[-1, 1]^n$, in order to improve the numerical stability of the linear algebra operations. In our experiments, we found out that using unitary numbers leads to better numerical stability. Since both our and their methods rely on SVD, our experiments confirm their idea.

The idea of the above methods is to interpolate the coefficients using successively larger supports, starting with a quite small support and extending it so as to reach the exact one. Existing approaches have used upper bounds on the total implicit degree, thus ignoring any sparseness structure.

In the context of sparse elimination, the Newton polytope captures the notion of degree. Given an implicit polytope we can naturally define candidates of smaller support, the equivalent of lower degree in classical elimination, by an inner integral offset of the implicit polytope:

Offset can be repeated, thus producing a list of implicit supports yielding implicit equations whose approximation error should be proportional to the number of peeling steps. Clearly, the computed implicit equation is of lower degree than the actual one. It is an open question to bound the difference of the corresponding zero sets.

3. SUPPORT PREDICTION

Most existing approaches employ total degree bounds on the implicit polynomial to compute a superset of the implicit support, e.g. [2]. This fails to take advantage of the sparseness of the input in order to accelerate computation, and to exploit sparseness in the implicit polynomial in the sense of Prop. 2. For this, computing the Newton polytope of a rational hypersurface was posed in [23] for generic Laurent polynomial parameterizations, in the framework of sparse elimination theory.

Algorithms based on tropical geometry have been offered in [7, 21, 22]. This method computes the abstract tropical variety of a hypersurface parameterized by generic Laurent polynomials in any number of variables, thus yielding its implicit support; it is implemented in `TrIm`. For non-generic parameterizations of rational curves, the implicit polygon is predicted. In higher dimensions, the following holds:

Proposition 1. *[21, prop.5.3] Let $f_0, \dots, f_n \in \mathbb{C}[t_1^{\pm 1}, \dots, t_n^{\pm 1}]$ be any Laurent polynomials whose ideal of algebraic relations is principal, say $I = \langle g \rangle$, and $P_i \subset \mathbb{R}^n$ the Newton polytope of f_i. Then, the polytope which is constructed combinatorially from $P_0, \dots, P_n$ as in [21, sec.5.1] contains a translate of the Newton polytope of g.*

The tropical approach was improved in [5, sec.5.4] to yield the precise implicit polytope in $\mathbb{R}^3$ for non-generic parameterizations of surfaces in 3-space.

In [6], they determine the Newton polygon of a curve parameterized by rational functions, without any genericity assumption. In a similar direction, an important connection with combinatorics was described in [13], as they showed that the Newton polytope of the projection of a generic complete intersection is isomorphic to the mixed fiber polytope of the Newton polytopes associated to the input data.

In [12] a method relying on sparse elimination for computing a superset of the generic support from the resultant polytope is discussed, itself obtained as a (non orthogonal) projection of the secondary polytope. The latter was computed by calling Topcom [18]. This approach was quite expensive and, hence, applicable only to small examples; it is refined and improved in this paper.

In [11], sparse elimination is applied to determine the vertex representation of the implicit Newton polygon of planar curves. The method uses mixed subdivisions of the input Newton polygons and regular triangulations of pointsets defined by the Cayley trick. It can be applied to polynomial and rational parameterizations, where the latter may have the same or different denominators. The method offers a set of rules that, applied to the supports of sufficiently generic rational parametric curves, specify the 4, 5, or 6 vertices of the implicit polygon. In case of non-generic inputs, this polygon is guaranteed to contain the Newton polygon of the implicit equation. The method can be seen as a special case of the following general approach based on sparse elimination.

3.1 Sparse elimination

Sparse elimination subsumes classical (or dense) elimination in the sense that, when Newton polytopes equal the corresponding simplices, the former bounds become those of the classical theory.

Proposition 2. *[12, sec.3] Consider polynomial system $F_0, \dots, F_n \in K[t]$ as in (3), defining a hypersurface, and let A_i be the support of F_i. Then, the total degree of the implicit polynomial is bounded by $n!$ times the volume of the convex hull of $A_0 \cup \dots \cup A_n$. The degree of the implicit polynomial in some x_j, $j \in \{0, \dots, n\}$ is bounded by the mixed volume of the F_i, $i \neq j$, seen as polynomials in t; cf also [23, thm.2(2)].*

The classical results for the dense case follow as corollaries. Take a surface parameterized by polynomials of degree d, then the implicit polynomial is of degree d^2. For tensor parameterizations of bi-degree (d_1, d_2), the implicit degree is $2d_1d_2$.

Let $\{F_0, F_1, \dots, F_n\}$ be a polynomial system where $F_i \in K[t_1, \dots, t_n]$, with symbolic coefficients c_{ij}. Its *resultant* $\mathcal{R}$ is a polynomial in $\mathbb{Z}[c_{ij}]$, vanishing iff $F_0 = F_1 = \dots = F_n = 0$ has a common root in a specific variety. This is the projective variety over the algebraic closure $\overline{K}$ of K, in the case of projective resultants, or the toric variety X defined by the supports of the F_i's in the case of sparse (or toric) resultants, s.t. it contains the topological torus as a dense subset: $(\overline{K}^*)^n \subset X$. The Newton polytope $N(\mathcal{R})$ of the resultant polynomial is the *resultant polytope.* We call any monomial which corresponds to a vertex of $N(\mathcal{R})$ an *extreme term* of $\mathcal{R}$.

The *Minkowski sum* $A+B$ of convex polytopes $A, B \subset \mathbb{R}^n$ is the set $A+B = \{a+b \mid a \in A, b \in B\} \subset \mathbb{R}^n$. Given $n+1$ convex polytopes P_i of dimension n, a *fine* or *tight mixed subdivision* of $P = \sum_{i=0}^n P_i$, is a collection of n-dimensional convex polytopes σ, called (Minkowski) *cells*, s.t.: (1) They form a polyhedral complex that partitions P, and (2) Every cell σ is a Minkowski sum of subsets $\sigma_i \subset P_i$: $\sigma = \sigma_0 + \cdots + \sigma_n$, where $\dim(\sigma) = dim(\sigma_0) + \cdots + dim(\sigma_n) = n$.

A cell σ is called *i-mixed*, or *v_i-mixed*, if it is the Minkowski sum of n 1-dimensional segments $E_j \subset P_j$ and one vertex $v_i \in P_i : \sigma = E_0 + \cdots + v_i + \cdots + E_n$. A mixed subdivision is called *regular* if it is obtained as the projection of the lower hull of the Minkowski sum of lifted polytopes $\hat{P}_i := \{(p_i, \omega(p_i)) \,|\, p_i \in P_i\}$. If the lifting function ω is sufficiently generic, then the induced mixed subdivision is tight. We recall a surjection from the regular fine mixed subdivisions to the vertices of the resultant polytope:

Theorem 3. *[20] Given a polynomial system and a regular fine mixed subdivision of the Minkowski sum of the Newton polytopes of the supports of the polynomials in the system, an extreme term of the resultant $\mathcal{R}$ equals*

$$c \cdot \prod_{i=0}^{n} \prod_{\sigma} c_{i\sigma_i}^{vol(\sigma)}$$

where $\sigma = \sigma_0 + \sigma_1 + \cdots + \sigma_n$ ranges over all σ_i-mixed cells, and $c \in \{-1, +1\}$.

Computing all regular fine mixed subdivisions reduces, due to the so-called Cayley trick, to computing all regular triangulations of a point set of cardinality $|A_0| + \cdots + |A_n|$ in dimension $2n$. This is the Cayley embedding of the A_i's. The set of all regular triangulations corresponds to the vertices of the secondary polytope. We enumerate all regular triangulations of the Cayley embedding: it is equivalent to enumerating all regular fine mixed subdivisions of $A_0 + \cdots + A_n$. Each such subdivision yields a vertex of $N(\mathcal{R})$.

We use sparse elimination to implicitize (hyper)surfaces, based on the implementation of [9]. This is an ongoing project that aims at optimizing the computation of $N(\mathcal{R})$ without enumerating all mixed subdivisions. Still, we sketch the theoretical procedure.

Let $A_i \subset \mathbb{Z}^n$, $i = 0, \ldots, n$, be the supports of the polynomials in (3), of cardinality m_i, and P_i, $i = 0, \ldots, n$ the corresponding Newton polytopes. For each A_i, we introduce symbolic coefficients c_{ij}, $j = 1, \ldots, m_i$, and define the polynomial F_i in the c_{ij}'s with the same support. Each c_{ij} corresponds to a (possibly constant) linear polynomial in x_i, with coefficients from f_i. The sparse resultant $\mathcal{R} = \mathcal{R}_{A_0,\ldots,A_n}(F_0, \ldots, F_n)$ of the F_i's with respect to t is well-defined, and is an irreducible integer polynomial in the c_{ij}'s. Consider an epimorphism of rings

$$\phi : K \to \mathbb{C}[x_i] : \; c_{ij} \mapsto \text{ linear polynomial in } x_i, \qquad (4)$$

yielding a specialization of the coefficients c_{ij} to the actual coefficients of (2). The specialized sparse resultant $\mathcal{R}' := \phi(\mathcal{R})$ is a polynomial in x_i, which coincides with the implicit equation, provided that $\mathcal{R}'$ does not vanish, a certain genericity condition is satisfied, and the parametrization is generically 1-1 [23, thm.2]. The following diagram illustrates these concepts:

$$\begin{tikzcd}
\{x_i = \frac{g_i(t)}{h_i(t)}\} \arrow[r] \arrow[rrrd] & \{x_i h_i(t) - g_i(t)\} \arrow[r, "\phi^{-1}"] & \{F_i\} \arrow[r] & \mathcal{R}_{A_0,\ldots,A_n}(F_0, \ldots, F_n) \arrow[d, "\phi"] \\
 & & & p(x_0, \ldots, x_n)
\end{tikzcd}$$

If the latter condition fails, then $\mathcal{R}'$ is a power of the implicit equation [3], [23, thm.3]. When the genericity condition fails for a specialization of the c_{ij}'s, the support of the specialized resultant is a superset of the support of actual implicit polynomial modulo a translation, provided the sparse resultant does not vanish. This follows from the fact that the method computes the same implicit polytope as the tropical approach, whereas the latter is characterized in prop. 1. In particular, the resultant polytope is a Minkowski summand of the *fiber polytope* $\Sigma_\pi(\Delta, P)$, where polytope Δ is a product of simplices, each corresponding to a support A_i, $P = \sum_{i=0}^n P_i$, and π is a projection from Δ onto P. Then, $\Sigma(\Delta, P)$, is strongly isomorphic to the secondary polytope of the point set obtained by the Cayley embedding of the A_i's, [20, sec.5].

4. IMPLICITIZATION ALGORITHM

The steps of our implicitization algorithm are given below, and apply to any support prediction method.
Input: Polynomial or rational parametrization $x_i = f_i(t_1, \ldots, t_n)$.
Output: Implicit polynomial $p(x_i)$ in the monomial basis in $\mathbb{N}^{n+1}$.
1. Support prediction determines (a superset of) the implicit polytope vertices.
2. Compute all lattice points $S \subseteq \mathbb{N}^{n+1}$ in the polytope.
3. Repeat $|S|$ times: Select value τ for t, evaluate $x_i(t)$, $i = 0, \ldots, n$, thus evaluating each monomial in S.
4. Given $|S| \times |S|$ matrix M, solve $M\vec{p} = 0$ for p; return the primitive part of polynomial $\vec{p}^\top S$.

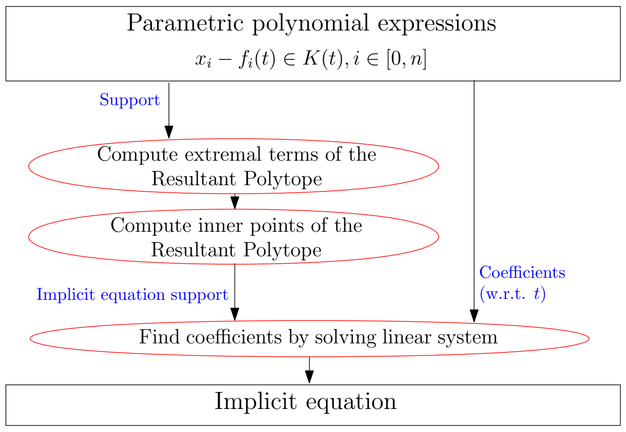

Parametric polynomial expressions
$x_i - f_i(t) \in K(t), i \in [0, n]$
Support
Compute extremal terms of the Resultant Polytope
Compute inner points of the Resultant Polytope
Implicit equation support
Coefficients (w.r.t. t)
Find coefficients by solving linear system
Implicit equation

Figure 3: Implicitization with support prediction

4.1 Building the matrix

We focus on two support prediction methods. The first applies only to curves and is described in [sect. 3], [11]. The second is general and computes the support of the resultant of system (3).

In the case of parametric curves, given the predicted polygon, we compute the m lattice points a_i that it contains.

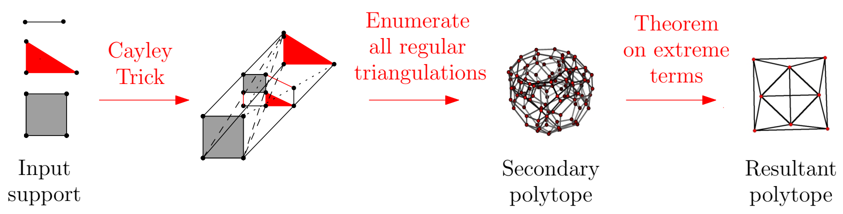


Figure 2: Computing the resultant polytope of 3 Newton polygons.

Each $a_i = (a_{i0}, a_{i1})$ is an exponent of a (potential) monomial of the implicit polynomial. In order to obtain the coefficients we evaluate $x_i(t),\ i = 0, 1$, at some $\tau_k,\ k = 1, \ldots, m$, and construct an $m \times m$ matrix M with rows indexed by $\tau_1, \ldots, \tau_m$ and columns by $a_1, \ldots, a_m$. Let $x(\tau_k)^{a_i}$ denote the evaluated ith monomial $x_0(\tau_k)^{a_{i0}} x_1(\tau_k)^{a_{i1}}$ at τ_k. Then

$$M = \begin{array}{c@{}c} \begin{bmatrix} x(\tau_1)^{a_1} & \cdots & x(\tau_1)^{a_m} \\ \vdots & \cdots & \vdots \\ x(\tau_m)^{a_1} & \cdots & x(\tau_m)^{a_m} \end{bmatrix} & \begin{array}{c} \tau_1 \\ \vdots \\ \tau_m \end{array} \end{array} \qquad (5)$$

To apply the general support prediction [9], consider the polynomials $F_i,\ i = 0, \ldots, n$ in Equation (3), with symbolic coefficients c_{ij}:

$$F_i = \sum_{j=1}^{d_i} c_{ij} t^{a_{ij}},$$

where d_i is the number of monomials in t with non-zero coefficient, i.e. the cardinality of support A_i. Given supports A_i, the output is the set V of vertices of the Newton polytope of the resultant of the F_i's. Each resultant vertex $a_i = (a_{i1}, \ldots, a_{id}) \in V$ is a d-dimensional vector, where $d = \sum_{i=0}^{n} d_i$, and corresponds to an extreme resultant monomial c^{a_i}, where c is the d-dimensional vector of the symbolic coefficients of all F_i's:

$$c := (c_1, \ldots, c_d) = \underbrace{(c_{01}, \ldots, c_{0d_0}}_{\text{from } F_0}, \underbrace{c_{11}, \ldots, c_{id_1}}_{\text{from } F_1}, \ldots\ldots, \underbrace{c_{n1}, \ldots, c_{nd_n})}_{\text{from } F_n}.$$

Given the resultant polytope vertices, we compute the set S of all m lattice points in the resultant polytope. We apply specialization ϕ to the set of monomials in c_{ij}'s with exponents in S to obtain a set of polynomials (products of linear polynomials) in x_i's. We abuse notation and denote this set also by $\phi(S)$, i.e. we identify each $\phi(c)^{a_k}$ with its exponent a_k.

When we substitute the symbolic coefficients c_{ij} by the actual univariate polynomials in x_i, some of the vertices of the implicit polytope may map to identical expressions: $\exists a_k \neq a_l \in V$ s.t. $\phi(c)^{a_k} = \phi(c)^{a_l}$. By examination of the $\phi(c)^{a_k}$, we remove duplicates. In the following we assume that $\phi(S)$ has no multiple entries.

Matrix M is constructed similarly as before. Let $\phi(S) = \{a_1, \ldots, a_m\} \subset \mathbb{N}^d$ be the lattice points, which form a superset of the resultant support. Each column contains a product $\phi(c)^{a_k} = \phi(c_1^{a_{k1}} \cdots c_d^{a_{kd}})$ evaluated at various τ_k, for $k = 1, \ldots, m$. Thus, we define the following $m \times m$ matrix, with columns indexed by the a_k's:

$$M = \begin{array}{c@{}c} \begin{array}{ccc} (a_{11}, \ldots, a_{1d}) & \cdots & (a_{m1}, \ldots, a_{md}) \end{array} & \\ \begin{bmatrix} \phi(c_1^{a_{11}} \cdots c_d^{a_{1d}})(\tau_1) & \cdots & \phi(c_1^{a_{m1}} \cdots c_d^{a_{md}})(\tau_1) \\ \vdots & \cdots & \vdots \\ \phi(c_1^{a_{11}} \cdots c_d^{a_{1d}})(\tau_m) & \cdots & \phi(c_1^{a_{m1}} \cdots c_d^{a_{md}})(\tau_m) \end{bmatrix} & \begin{array}{c} \tau_1 \\ \vdots \\ \tau_m \end{array} \end{array} \qquad (6)$$

Each $\phi(c)^{a_k} \in \phi(S)$ is a product of linear polynomials in x_i.

It appears to be usual case, that some $\phi(c)^{a_k}$ project into same polynomials in x_i's. This allows us to noticeably decrease size of the matrix M by removing the duplicates and using only $\phi(c)^{a_k}$ that have unique representations in x_i. After expanding and simplifying $\phi(c)^{a_k}$, we get a set of monomials in x_i's. Let m' be the number of integer points in the convex hull they define, thus yielding a superset of the implicit support. We form an $m' \times m'$ matrix M', whose kernel yields the implicit coefficients. The columns of M' are indexed by monomials in the x_i's, hence in $\mathbb{Z}^n$, and the matrix entries are evaluated monomials in the x_i's, while the entries of matrix M are evaluated polynomials in the x_i's.

Often, matrix M' is of larger size than matrix M, see tables 3, 5. This is not a paradox, since ϕ is *not* an orthogonal projection of the c_{ij}'s to a space of lower dimension. Applying ϕ to a monomial in the c_{ij}'s of total degree δ results to a product of linear polynomials in the x_i's, which are developed to yield all monomials of total degree $< \delta$. For example, consider the monomial $c_{01}^3 c_{12}^4$ and the mapping $c_{01} \mapsto x + 1, c_{12} \mapsto y$. Then, we obtain the monomials $x^3y^4, x^2y^4, xy^4, y^4$. If cancellations do not occur among the new monomials, the resulting matrix is larger.

Moreover, this new support may not be the set of vertices of a convex polytope, hence we must perform a convex hull computation and then compute the set of integer points it contains. However, both computations take place in a space of low dimension ($n \ll d$), while computing the integer points inside the polytope in $\mathbb{R}^d$ defined by the set of monomials in the c_{ij}'s can be a hard task.

4.2 Maple Implementation

Assume that matrix M has corank 1, i.e. $\text{rank}(M) = m-1$. Solving the linear system

$$M\vec{p} = \vec{0}, \qquad (7)$$

yields the implicit coefficient p_i for each $\phi(c)^{a_k}$. The kernel is one-dimensional, hence some entry p_i is set to 1. We form the inner product of the vector of the monomials indexing the columns of M with $\vec{p}$, and then take the primitive part of the resulting polynomial to define the implicit equation.

When M is evaluated at random integer values or unitary complex numbers, we tried Maple function `Linear-`

`Solve`, from package `LinearAlgebra`, and `Linear` from package `SolveTools`. Equivalently, we can compute the null-space null(M) of M using the command `NullSpace()` of the `LinearAlgebra` package. All of these functions return the same output in roughly the same runtime with command `LinearSolve` being slightly faster in larger examples. Exact Maple methods can treat indefinites usually encountered in parametric expressions (a, b, c in tables 2,4). For larger examples, we trade exactness for speed and apply Singular Value Decomposition (SVD) with command `SingularValues()`, thus computing

$$M\vec{p}^{\top} = (U\Sigma V^{\top})\vec{p}^{\top} = \vec{0}^{\top} \Leftrightarrow \Sigma\vec{v}^{\top} = \vec{0}^{\top},\ V\vec{v}^{\top} = \vec{p}^{\top}. \quad (8)$$

A basis of null(M) consists of the last columns of V corresponding to the zero singular values of M, because V is orthogonal. When corank$(M) = 1$, the last row of $V^{\top}$ corresponds to $\vec{p}$. The same derivation holds if M is rectangular, say $\mu \times m, \mu \geq m$. Then Σ is of the same dimensions, U is $\mu \times \mu$, and V is $m \times m$, where its last column is the sought vector.

A central part in our linear system construction is held by the evaluation of matrix M at convenient τ. Implementing our methods in Maple, we have tried integer and complex values. In the former case, we used random and mutually prime integers to achieve exactness. The chosen value is discarded if it makes some denominator vanish among the parametric expressions. We also tried complex values for τ: Given an $m \times m$ matrix, we used $2m$-th roots of unity, and random unitary complexes, i.e. with modulus equal to 1.

Table 1 shows representative timings about these options that we examined. In particular, SVD is about 10 times faster than exact linear algebra on significant inputs, as expected; see the last two columns of the table. We plan to use LinBox or Eigen as a faster alternative for our exact algorithm. In what concerns the various evaluation points, our experiments show that runtimes do not vary significantly in small examples but in larger ones the random integers and roots of unity evaluated as floats seem to give faster timings.

However, unitary complexes seem to offer the most stable results, from the numerical viewpoint. Random integers give matrices which are closer to having numerical corank 1, although this property is not reproducible in every experiment. The numerical stability of the result is measured by comparing ratios of singular values of matrix M; we employ the *condition number* $\kappa(M) = \sigma_1/\sigma_m$, and σ_1/σ_{m-1}, where σ_1 is the maximum singular value. By comparing these two numbers, we decide whether the matrix is of (numerical) corank 1, otherwise we repeat the computation. Another approach is to consider the row of $V^{\top}$ corresponding to the singular value σ_i of M that satisfies $\sigma_1/\sigma_i \gg \sigma_1/\sigma_{i-1}$, but experiments indicate that this does not improve neither the numerical stability or the correctness of the result. We plan to use LAPACK for further examination of numerical stability, while employing larger examples.

When using numerical methods the computed implicit equation is not a polynomial with integer coefficients. In [2], some post processing is done to discover the integer relations among the coefficients. Then, the polynomial is multiplied by an appropriate number to recover the implicit polynomial with integer coefficients. We utilize a similar method by converting the computed kernel-vector in real or complex space to a rational vector. In practice this is done by assigning a small value to the `Digits` environment variable of Maple,

Table 1: Runtimes on Maple (seconds)

	SVD			NullSpace
curve	root of 1	unitary $\mathbb{C}$	rand.$\mathbb{Z}$	rand.$\mathbb{Z}$
Cardioid	0.356	0.289	0.076	0.132
Conchoid	0.12	0.092	0.048	0.084
Nephroid	0.012	1.15	0.012	2.3
Talbot's	0.132	0.084	0.108	1.562
Ranunculoid	83.749	-	121.084	8809.43

then setting all coefficients smaller than a certain threshold, defined by the problem's condition number, equal to zero. The result is not always correct, so its validity is checked by plugging in the implicit equation the parametric expressions and testing if the result is identically zero. The overall process is computationally hard and it can be avoided whenever an implicit equation with floating point coefficients is sufficient for a specific problem.

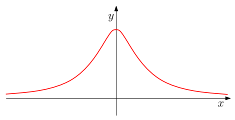
5. EXPERIMENTS

First, we describe some of the examples in details. Instead of x_0, x_1, x_2 we use x, y, z. In several cases, when the input is a parametric family we denote by $a, b, c \in \mathbb{R}^*$ the nonzero parameter. These are set equal to some nonzero constant when using numerical methods. For trigonometric parameterizations, we use the standard conversion to an algebraic form by

$$\sin\theta = \frac{2\tan\theta/2}{1+\tan^2\theta/2},\ \cos\theta = \frac{1-\tan^2\theta/2}{1+\tan^2\theta/2}.$$

5.1 Curves

Witch of Agnesi. $x = at, y = a/(1+t^2)$.



The curve support prediction provides 3 implicit polygon's vertices: $(0,0), (2,1), (0,1)$. Their convex hull contains one lattice point. We construct a 4×4 matrix and by solving the corresponding system we find 3 nonzero coefficients; the implicit polynomial is $-a^3 + x^2y + a^2y$.

General support prediction method: The polynomials are: $c_{00} + c_{01}t = 0,\ c_{10} + c_{11}t^2 = 0$, with coefficients $c_{00} = -x, c_{01} = a, c_{10} = a-y, c_{11} = -y$, and supports $\{0,1\}, \{0,2\}$. The resultant polytope is the segment $[(0,2,1,0)(2,0,0,1)]$ which contains no internal points. These two vectors index the columns of M and correspond to the monomials $\{c_{01}^2c_{10}, c_{00}^2c_{11}\}$, which are specialized to $\{a^2(a-y),\ (-x)^2(-y)\}$, that index the 2×2 matrix M. We solve $M\vec{p} = 0$ to find $\vec{p} = [p_0, p_1]$, yielding the implicit polynomial

$$p_0(a^3 - a^2y) + p_1(-x^2y),$$

hence $a^2y + x^2y - a^3$.

Alternatively, we can use the set $\{a^2(a-y),\ (-x)^2(-y)\}$ to obtain the implicit support in x, y: $\{(0,0), (0,1), (2,1)\}$. Its convex hull contains no internal points, hence M' is 3×3.

Folium of Descartes.

$$x = \frac{3at}{1+t^3},\ y = \frac{3at^2}{1+t^3}.$$

The curve support prediction yields polygon vertices $(0,0)$, $(0,3)$, $(1,1)$, $(3,0)$, thus 10 lattice points totally. Solving the 10×10 matrix yields 3 nonzero coefficients, and implicit polynomial $x^3 + y^3 - 3axy$.

General support prediction: The parametrization is represented by the polynomials $3at - x - xt^3$, $-yt^3 - y + 3at^2$ with supports $\{0,1,3\}, \{0,2,3\}$. Then, ϕ is defined as: $c_{00} = -x, c_{01} = 3a, c_{02} = -x$, $c_{10} = -y, c_{11} = 3a, c_{12} = -y$.

The method computes 6 vertices of the (symbolic) resultant polytope: $(0,0,3,3,0,0)$, $(0,2,1,1,2,0)$, $(0,3,0,1,0,2)$, $(2,0,1,0,3,0)$, $(2,1,0,0,1,2)$, $(3,0,0,0,0,3)$, which contains 4 inside points.

The straightforward approach would be to form a 10×10 matrix M. In this case, $\vec{p} = [1, p_1, p_1, -1 - p_3 - p_5, -p_2 - p_4, p_1, p_2, p_3, p_4, p_5]$, where $p_i \in \mathbb{N}^*$. We can safely reduce matrix size by keeping only the distinct monomials in x, y, namely $a^4 x_1 y_1$, ${x_1}^3 a^3$, $a^3 {y_1}^3$, ${x_1}^2 a^2 {y_1}^2$, ${x_1}^3 {y_1}^3$.

When substituting c_{ij} by the corresponding univariate polynomials we get 5 distinct monomials in x, y: x^3y^3, x^2y^2, xy, y^3, x^3. The convex hull of the supports of these monomials is defined by the vertices: $(3,0)$, $(0,3)$, $(1,1)$, $(3,3)$. We find 11 lattice points and build 11×11 matrix. Solving linear system gives us 3 nonzero coefficients, and we get implicit equation: $x^3 + y^3 - 3axy = 0$.

Nephroid. Rational representation:

$$x = \frac{6(1-t^2)(1+t^2)^2 - 4(1-t^2)^3}{(1+t^2)^3},\ y = \frac{32t^3}{(1+t^2)^3}$$

The curve support prediction method returns vertices $(0,0)$, $(6,0)$, $(0,6)$. Their convex hull contains 28 lattice points. Solving the linear system we find implicit polynomial $48x^2 - 12y^4 - 64 - 24x^2y^2 + y^6 - 12x^4 + 3x^2y^4 + x^6 - 60y^2 + 3x^4y^2$.

Conhoid. Rational representation:

$$x = a + \frac{1-t^2}{1+t^2}, y = \frac{2at}{1-t^2} + \frac{2t}{1+t^2}.$$

The curve support prediction has to be applied with care, because of the special structure of the supports: one has to use the more robust statement of [10]. For the general support prediction, we have: $x - a - 1 + xt^2 + at^2 - t^2 = 0$, $y - 2at - 2t - 2at^3 + 2t^3 - yt^4 = 0$, with coefficients $c_{00} = x - a - 1$, $c_{01} = x - a + 1$, $c_{10} = y$, $c_{11} = -2a - 2$, $c_{12} = -2a + 2$, $c_{13} = -y$, and supports: $\{0,2\}, \{0,1,3,4\}$. The predicted support has 6 lattice points: $(0,4,2,0,0,0)$, $(1,3,0,2,0,0)$, $(2,2,0,1,1,0)$, $(2,2,1,0,0,1)$, $(3,1,0,0,2,0)$, $(4,0,0,0,0,2)$, yielding the monomials ${c_{01}}^4 {c_{10}}^2$, $c_{00}{c_{01}}^3{c_{11}}^2$, ${c_{00}}^2{c_{01}}^2 c_{11} c_{12}$, ${c_{00}}^2{c_{01}}^2 c_{10} c_{13}$, ${c_{00}}^3 c_{01} {c_{12}}^2$, ${c_{00}}^4 {c_{13}}^2$. Solving $M\vec{p} = 0$ gives $\vec{p} = [1,1,-2,2,1,1]$, and implicit polynomial $16x^2a^2 - 16x^2 - 32x^3a - 32y^2xa + 16y^2x^2 + 16y^2a^2 + 16x^4$.

5.2 Surfaces

Infinite cylinder. The rational parameterization yields the following polynomial system: $1 - xt^2 - x - t^2$, $-y - yt^2 + 2t$, $-z + s$, with supports $\{(0,0),(2,0); (0,0),(1,0),(2,0); (0,0), (0,1)\}$. The support prediction gives 4 vertices of the resultant polytope: $(0,2,2,0,0,0,0)$, $(1,1,0,2,0,0,0)$, $(1,1,1,0,1,0,0)$, $(2,0,0,0,2,0,0)$.

Specialization ϕ maps the coefficients to polynomials in x, y, z: $\phi(c_{0j}) = \{1 - x, -1 - x\}$, $\phi(c_{1j}) = \{-y, 2, -y\}$, $\phi(c_{2j}) = \{-z, 1\}$. Substituting c_{ij} in the predicted monomials by the corresponding univariate polynomials we obtain 9 lattice points: $(0,0,0)$, $(0,2,0)$, $(2,2,0)$, $(2,0,0)$, $(0,1,0)$, $(1,0,0)$, $(1,1,0)$, $(1,2,0)$, $(2,1,0)$. We get the canonical form of the implicit polynomial: $x^2 + y^2 = a^2$.

Infinite cone. In some cases of rational parametrizations our method results in a multiple of the actual implicit equation. For example, the infinite cone is a quadratic surface with canonical equation $z^2 = \frac{x^2+y^2}{a^2}$, but our result has degree 4 (see Table 5). This is due to the fact that we do not compute the minimal variety when there are base points.

Sine surface. Rational representation:

$$x = \frac{2t}{1+t^2},\ y = \frac{2s}{1+s^2},\ z = \frac{2s + 2t - 2st^2 - 2ts^2}{1 + s^2 + t^2 + s^2t^2}.$$

Applying support prediction we obtain 1027 lattice points in the resultant polytope. Removing duplicates leaves 87 distinct matrix entries. Solving the linear system, we obtain a kernel vector with 66 nonzero entries. Meanwhile, substituting c_{ij} by the corresponding univariate polynomials leads to a polytope with 125 lattice points; solving the linear system yields 7 nonzero kernel-vector entries. The implicit polynomial is $-2y^2z^2 + 4x^2y^2z^2 - 2x^2y^2 - 2x^2z^2 + z^4 + y^4 + x^4$.

5.3 Results

See the tables in the Appendix for all results concentrated. All experiments where performed on a Celeron 1.60GHz, 1Gb memory, linux machine.

In particular, tables 2,3 contain information on our experiments for curves, including runtimes (given in sec.) on Maple v. 11. In table 3 columns are labeled as: degrees of parametric and implicit equations, number of monomials in the parametric equations, number of lattice points in the resultant polytope when applying the general support prediction method ("Lat.pts."), size of the matrix constructed when applying the curve support prediction [11] (field "Cur.matr."), time in seconds for the prediction routine and linear solving ("Cur.time"), matrix size for using the general support prediction ("Gen.matr."), runtime for the removal of duplicates and linear solving ("Gen.time"), and number of nonzero kernel-vector entries for the matrix based on the c_{ij}'s ("Gen.nonzero"), matrix size for the case when, after using general support prediction, we map the c_{ij} to univariate polynomials ("Map.matr."), runtime for mapping and linear solving ("Map.time"), number of monomials of the implicit curve ("Map.nonzero").

Table 4, 5 contain the information on our experiments for surfaces. In table 5 columns are labeled as: parametric and implicit degree, number of monomials in the parametric equations, number of lattice points in the resultant polytope ("Lat.pts."), matrix size with duplicates removed ("Gen.matr."), runtime for duplicate removal and solving ("Gen.time"), number of nonzero entries of the kernel vector ("Gen.nonzero"), matrix size for mapping c_{ij} to univariate polynomials ("Map.matr."), runtime for mapping and solving ("Map.time"), and number of monomials of the implicit surface ("Map.nonzero")

The tables contain blank entries (-) when the linear system appeared too large to try solving.

As expected (see table 3), the curve support prediction method shows far better results than the general support prediction method applied to curves. Comparing the two matrix constructions, based on the general support prediction, we conclude that, while in some cases mapping gives us a smaller matrix (e.g. Cardioid, Nephroid), in most cases duplicate removal proves to be more effective (e.g. Folium of Descartes, Sine surface). Another observation is that, right now, a significant fraction of the complexity is due to the general support prediction method, which can readily be improved by current work in our team or by using other prediction methods such as tropical geometry.

6. CURRENT WORK

To improve numerical stability, we may use more evaluation points than monomials, thus forming a rectangular matrix to which we apply SVD, as explained following equation (8). Our current work examines whether some matrix structure can be revealed while exploiting the freedom in choosing appropriate τ values to evaluate the monomials. Our team strives to improve the support calculating algorithm [9] for arbitrary-dimensional hypersurfaces.

We consider specific challenges, such as the bicubic surface [14], of implicit degree 18, containing 1330 terms: the approach of [12] could not handle it because it generates 737129 regular triangulations (by TOPCOM) in a file of 383MB. Also, there is a surface challenge suggested by [15] and meant for a practical problem, with parametrization expressions of total degrees 9,9,6, respectively.

Our approach represents an implicit (hyper)surface by a kernel vector. It is challenging to devise suitable CAGD algorithms that exploit this representation, e.g. for surface-surface intersection, as in [8]. Another practical problem is to implicitize curve or surface splines defined by k segments or patches, respectively. Assuming the k parametric representations yield polynomials with (roughly) the same Newton polytopes, one could use the implicit polytope defined by any of these systems. Then, we can form a single matrix M and evaluate it over points spanning all k segments or patches, thus expecting a single (approximate) implicit polynomial.

It is possible to approximate k (pieces of) manifolds with a single implicit equation, by applying SVD on $[M_1 \cdots M_k]^\top$. We could also extend our approach to interpolating the implicit polynomial in other bases, such as Bernstein or Lagrange, by predicting the resultant support in these bases.

Acknowledgement. IZE and TK are partially supported by Marie-Curie Initial Training Network "SAGA" (ShApes, Geometry, Algebra), FP7-PEOPLE contract PITN-GA-2008-214584.

7. REFERENCES

[1] M. Aigner, A.Poteaux, B. Juttler. Approximate implicitization of space curves. *Symbolic & Numer. Comp*, Springer, 2011. To appear

[2] R. Corless, M. Giesbrecht, I. S. Kotsireas, and S. Watt. Numerical implicitization of parametric hypersurfaces with linear algebra. In *Proc. AISC*, pp. 174–183, 2000.

[3] D. A. Cox, J. B. Little, and D. O'Shea. *Using Algebraic Geometry*, vol. 185 of GTM. Springer, 1998.

[4] M. Cueto, E. Tobis, and J. Yu. An implicitization challenge for binary factor analysis. *J. Symbolic Computation*, 45(12):1296–1315, 2010.

[5] M. A. Cueto. *Tropical Implicitization.* PhD thesis, Dept Mathematics, UC Berkeley, 2010.

[6] C. D'Andrea and M. Sombra. The Newton polygon of a rational plane curve. *Math. in Computer Science*, 4(1):3–24, 2010.

[7] A. Dickenstein, E. M. Feichtner, and B. Sturmfels. Tropical discriminants. J. AMS, 1111–1133, 2007.

[8] T. Dokken and J. B. Thomassen. Overview of approximate implicitization. *Topics in algebraic geometry and geometric modeling*, 334:169–184, 2003.

[9] I. Z. Emiris, V. Fisikopoulos, and C. Konaxis. Regular triangularions and resultant polytopes. In *Proc. Europ. Workshop Comp. Geometry*, pp. 137–140, 2010.

[10] I. Z. Emiris, C. Konaxis, and L. Palios. Computing the Newton polygon of the implicit equation. Tech. Report 0811.0103v1, arXiv, 2007.

[11] I. Z. Emiris, C. Konaxis, and L. Palios. Computing the Newton polygon of the implicit equation. *Math. in Computer Science, Special Issue*, 4(1):25–44, 2010.

[12] I.Z. Emiris and I. S. Kotsireas. Implicit polynomial support optimized for sparseness. In *Proc. Conf. Comput. science Appl.*, pp. 397–406, Springer, 2003

[13] A. Esterov and A. Khovanski. Elimination theory and Newton polytopes. *ArXiv Math.*, Nov. 2006.

[14] L. Gonzalez-Vega. Implicitization of parametric curves and surfaces by using multidimensional Newton formulae. *J. Symbolic Comput.*, 23(2-3):137–151, 1997.

[15] R. Krasauskas. Personal communication, 2011.

[16] I. S. Kotsireas and E. Lau. Implicitization of polynomial curves. In *Proc. ASCM*, pp. 217–226, Beijing, 2003.

[17] A. Marco and J. Martinez. Implicitization of rational surfaces by means of polynomial interpolation. *CAGD*, 19:327–344, 2002.

[18] J. Rambau. TOPCOM: Triangulations of point configurations and oriented matroids. *Intern. Conf. Math. Software*, pp. 330–340. World Scientific, 2002.

[19] M. Shalaby and B. Jüttler. Approximate implicitization of space curves and of surfaces of revolution. *Algebraic Geometry & Geom. Modeling*, pp. 215–228. Springer, 2008.

[20] B. Sturmfels. On the Newton polytope of the resultant. *J. Algebraic Combin.*, 3:207–236, 1994.

[21] B. Sturmfels, J. Tevelev, and J. Yu. The Newton polytope of the implicit equation. *Moscow Math. J.*, 7(2), 2007.

[22] B. Sturmfels and J. Yu. Tropical implicitization and mixed fiber polytopes. In *Soft. for Algebraic Geom*, vol. 148 of *IMA* pp. 111–131, Springer, 2008.

[23] B. Sturmfels and J. T. Yu. Minimal polynomials and sparse resultants. *In Proc. Zero-dimensional schemes (Ravello, 1992)*, pp. 317–324. De Gruyter, 1994.

[24] E. Wurm, J. Thomassen, B. Juttler, T. Dokken. Comparative benchmarking of methods for approximate implicitization. In *Geom. Modeling & computing 2003*, pp. 537–548. 2004

[25] R. Zippel. *Effective Polynomial Computation.* Kluwer Academic Publishers, Boston, 1993.

Appendix

Table 2: Curves

Curve	Parametric form	Implicit polynomial
Cardioid	$a(2\cos(t) - \cos(2t));\ a(2\sin(t) - \sin(2t))$	$-3a^4 - 6a^2y^2 + y^4 + 8a^3x - 6a^2x^2 + 2x^2y^2 + x^4$
Conchoid	$a + \cos(t);\ a\tan(t) + \sin(t)$	$a^2y^2 - 2axy^2 + a^2x^2 - x^2 + x^2y^2 - 2ax^3 + x^4$
Folium of Descartes	$\dfrac{3at}{1+t^3};\ \dfrac{3at^2}{1+t^3}$	$y^3 - 3xy + x^3$
Nephroid	$a(3\cos(t) - \cos(3t));\ a(3\sin(t) - \sin(3t))$	$-64 - 60y^2 - 12y^4 + y^6 + 48x^2-$ $24x^2y^2 + 3x^2y^4 - 12x^4 + 3x^4y^2 + x^6$
Ranunculoid	$6\cos(t) - \cos(6t);\ 6\sin(t) - \sin(6t)$	$-52521875 - 1286250x^2 - 1286250y^2-$ $-32025(x^2+y^2)^2 + 93312x^5 - 933120x^3y^2 + 466560xy^4-$ $-812(x^2+y^2)^3 - 21(x^2+y^2)^4 - 42(x^2+y^2)^5 + (x^2+y^2)^6$
Talbot's curve	$\dfrac{(a^2 + c^2\sin^2(t))\cos(t)}{a};$ $\dfrac{(a^2 - 2c^2 + c^2\sin^2(t))\sin(t)}{b}$	$-16a^4c^8 + 32a^6c^6 - 16a^8c^4 - 8a^6b^2c^2y^2+$ $+32a^4b^2c^4y^2 - 8a^2b^2c^6y^2 - a^4b^4y^4 + 10a^2b^4c^2y^4-$ $-b^4y^4c^4 + b^6y^6 + 8a^8c^2x^2 + 8a^6c^4x^2 - 32a^4c^6x^2+$ $+16a^2c^8x^2 - 2a^6b^2x^2y^2 + 2a^4b^2c^2x^2y^2-$ $-20a^2b^2c^4x^2y^2 + +3a^2b^4x^2y^4 - a^8x^4-$ $-8a^6c^2x^4 + 8a^4c^4x^4 + 3a^4b^2x^4y^2 + a^6x^6$
Tricuspoid	$a(2\cos(t) + \cos(2t));\ a(2\sin(t) - \sin(2t))$	$-27a^4 + 18a^2y^2 + y^4 + 24axy^2+$ $+18a^2x^2 + 2x^2y^2 - 8 * ax^3 + x^4$
Witch of Agnesi	$at;\ \dfrac{a}{1+t^2}$	$-a^3 + a^2y + x^2y$

Table 3: Curves - results

Curve	Param. degree	Impl. degree	Param. m.nr.	Lat. pts.	Cur. matr.	Cur. time	Gen. matr.	Gen. time	Gen. nonzero	Map. matr.	Map. time	Map. nonzero
Cardioid	4,4	4	3,4	33	15	0.128	33	0.248	33	25	0.656	7
Conchoid	2,3	4	2,4	6	15	0.094	6	0.096	6	15	0.308	6
Folium of Descartes	3,3	3	3,3	10	10	0.092	5	0.032	3	11	0.144	3
Nephroid	4,4	6	4,5	454	28	0.256	426	-	-	49	5.452	10
Ranunculoid	12,12	12	7,12	-	91	8809.43	-	-	-	-	-	-
Talbot's curve	6,6	6	4,7	1600	28	109.342	421	-	-	-	-	23
Tricuspoid	4,4	4	3,4	33	15	0.216	33	0.236	33	25	0.540	8
Witch of Agnesi	1,2	3	2,2	2	4	0.016	2	0.044	2	4	0.048	3

Table 4: Surfaces

Surface	Parametric form	Implicit polynomial
Infinite cylinder	$a\cos(t)$; $a\cos(t)$; s	$-a^2+y^2+x^2$
Hyperbolic paraboloid	ts; t; $\dfrac{s^2}{b^2}-\dfrac{t^2}{a^2}$	$-x^2b^2+y^2a^2-za^2b^2$
Infinite cone	$at\cos(s)$; $at\sin(s)$; t	$x^2z^2+y^2z^2-a^2z^4$
Whitney umbrella	ats; at; as^2	y^2z-ax^2
Monkey saddle	at; as; $a(t^3-3ts^2)$	$-3xy^2-a^2z+x^3$
Handkerchief surface	at; as; $a(\dfrac{t^3}{3}+2(t^2-s^2)+ts^2)$	$-3a^2z+3xy^2+x^3+6x^2a-6ay^2$
Crossed surface	at; as; at^2s^2	$-a^3z+x^2y^2$
Quartoid	t; s; $-\dfrac{(t^2+s^2)^2}{a^3}$	$za^3+x^4+2x^2y^2+y^4$
Peano surface	t; s; $(2t^2-as)(as-t^2)$	$z+2x^4-3x^2ya+y^2a^2$
Bohemian dome	$\cos(t)$; $\sin(t)+\cos(s)$; $\sin(s)$	$2x^2y^2-2x^2z^2-4y^2+x^4+z^4+2y^2z^2+y^4$
Swallowtail surface	$a(ts^2+3s^4)$; $a(-2ts-3s^3)$; at	$-15axy^2z+3ay^4+y^2z^3-4xz^4+12ax^2z^2-9a^2x^3$
Sine surface	$\sin(t)$; $\sin(s)$; $\sin(t+s)$	$-2y^2z^2+4x^2y^2z^2-2x^2y^2-2x^2z^2+z^4+y^4+x^4$
Enneper's surface	$t-\dfrac{t^3}{3}+ts^2$; $2-\dfrac{s^3}{3}+t^2s$; t^2-s^2	$352836-78732x^2y^2z+749412z^2+101088z^3x^2y-303264x^2yz^2-$ $-25272x^2y^2z^3-62127z^5+75816x^2y^2z^2+314928x^2yz-$ $-4860x^4z^3-2916x^6+69984x^2y^3+23328y^4z^2-26244y^4z+$ $+72576yz^5+997272yz-669222y^2z-18144y^2z^5+209952y^3z-$ $-186624y^3z^2+2592x^2z^5-106920x^2z^3+34992x^4+5832x^4z^2+$ $+8748x^4y^2-34992x^4y-183708x^2y^2-268272z^4y-1122660z^2y+$ $+602640z^3y+67068z^4y^2-6912z^6y+653913z^2y^2+1728z^6y^2-$ $-228420z^3y^2+38880z^3y^3-4860z^3y^4-2304z^8-536544y^3+$ $+183708y^4-34992y^5+2916y^6-577368z-5616z^6-8748x^2y^4-$ $-34992x^2+2916x^2z^4+305451x^2z^2+7776z^7-314928x^2z+256z^9-$ $-524151z^3+916353y^2+263898z^4+174960x^2y-866052y-1728x^2z^6$

Table 5: Surfaces - results

Surface	Param. degree	Impl. degree	Param. m.nr.	Lat. pts.	Gen. matr.	Gen. time	Gen. nonzero	Map. matr	Map. time	Map. nonzero
Infinite cylinder	2,2,1	2	2,3,2	4	4	0.036	4	9	0.072	3
Hyperbolic paraboloid	1,1,2	2	2,2,3	3	3	0.028	3	7	0.064	3
Infinite cone	3,2,1	4	4,3,2	14	8	0.040	6	19	0.224	3
Whitney umbrella	2,1,2	3	2,2,2	2	2	0.024	2	4	0.056	2
Monkey saddle	1,1,3	3	2,2,3	3	3	0.028	3	8	0.064	3
Handkerchief surface	1,1,3	3	2,2,5	2	2	0.020	2	4	0.036	5
Crossed surface	1,1,4	4	2,2,2	5	5	0.032	5	10	0.072	2
Quartoid	1,1,4	4	2,2,4	4	4	0.012	4	16	0.140	4
Peano surface	1,1,4	4	2,2,4	4	4	0.020	4	10	0.080	4
Bohemian dome	2,4,2	4	2,6,3	142	58	2.764	-	125	83.362	7
Swallowtail surface	4,3,1	5	3,3,2	12	12	0.052	12	25	0.432	6
Sine surface	2,2,4	6	3,3,8	1027	87	9.244	66	125	107.102	7
Enneper's surface	3,3,2	9	4,3,3	439	258	74.304	258	106	33.562	57

Numerical Stability of Barycentric Hermite root-finding

[Extended Abstract]

Piers W. Lawrence
ORCCA, UWO
London, ON, Canada
plawren@uwo.ca

Robert M. Corless
ORCCA, UWO
London, ON, Canada
rcorless@uwo.ca

Categories and Subject Descriptors

G.1.5 [**Numerical Analysis**]: Roots of Nonlinear Equations—*Polynomials, methods for, Error analysis*

General Terms

Algorithms

Keywords

Companion matrix, Hermite interpolation, stability

1. INTRODUCTION

Computing the roots of a polynomial expressed in the Lagrange basis or a Hermite interpolational basis can be reduced to computing the eigenvalues of the corresponding companion matrix [2]. The result we present here is that roots of a polynomial computed via this method are exactly the roots of a polynomial with slightly perturbed coefficients.

2. COMPANION MATRIX PENCIL $(\mathbf{A}_0, \mathbf{B}_0)$

We consider here polynomials expressed in a Hermite interpolational basis. We suppose we are given distinct nodes $\tau_i \in \mathbb{C}$, $1 \le i \le n$, with positive integer confluencies $s_i \ge 1$ and local Taylor coefficients $\frac{1}{j!} f^{(j)}(\tau_i) = \rho_{ij}$. We define the generalised barycentric weights γ_{ij} for $1 \le i \le n$ and $0 \le j \le s_i - 1$ computed from the partial fraction decomposition of

$$\frac{1}{w(z)} = \frac{1}{\prod_{i=1}^{n}(z-\tau_i)^{s_i}} = \sum_{i=1}^{n}\sum_{j=0}^{s_i-1} \frac{\gamma_{ij}}{(z-\tau_i)^{j+1}} . \tag{1}$$

The Hermite interpolant of degree $d = -1 + \sum_{i=1}^{n} s_i$ is given by

$$p(z) = w(z) \sum_{i=1}^{n}\sum_{j=0}^{s_i-1}\sum_{k=0}^{j} \frac{\gamma_{ij}}{(z-\tau_i)^{j+1-k}} . \tag{2}$$

A companion matrix pencil $(\mathbf{A}_0, \mathbf{B}_0)$ is

$$\left(\begin{bmatrix} \mathbf{J}_{s_1}^T(\tau_1) & & & \mathbf{\Pi}_1 \\ & \ddots & & \vdots \\ & & \mathbf{J}_{s_n}^T(\tau_n) & \mathbf{\Pi}_n \\ \mathbf{\Gamma}_1^T & \cdots & \mathbf{\Gamma}_n^T & 0 \end{bmatrix}, \begin{bmatrix} \mathbf{I} & \\ & 0 \end{bmatrix} \right) \tag{3}$$

where

$$\mathbf{\Gamma}_i = [\gamma_{i0}, \cdots, \gamma_{i,s_i-1}]^T,\ \mathbf{\Pi}_i = [\rho_{i0}, \cdots, \rho_{i,s_i-1}]^T \tag{4}$$

and $\mathbf{J}_{s_i}^T(\tau_i)$ are transposed Jordan blocks of the form

$$\mathbf{J}_{s_i}^T(\tau_i) = \begin{bmatrix} \tau_i & & & \\ 1 & \tau_i & & \\ & \ddots & \ddots & \\ & & 1 & \tau_i \end{bmatrix} . \tag{5}$$

3. NUMERICAL STABILITY OF $(\mathbf{A}_0, \mathbf{B}_0)$

In [3, 4, 5] we find significant discussion of the backward error of companion matrix pencils and other linearizations. Those works consider only the monomial basis, and it turns out to be worthwhile to recapitulate their work for the Lagrange and Hermite bases.

The deepest part of their works is an explanation of why generic matrix perturbations of $\mathbf{A}_0$ (and $\mathbf{B}_0$) are equivalent (to first order) to a much more restricted perturbation to the n $(+1)$ values of the polynomial coefficients. The reason as deduced by Arnol'd [1] and refined by others, is that perturbations tangent to the orbit $\mathbf{TCT}^{-1}$ do not matter, to first order, and the other (normal) direction can be accounted for by small changes in the polynomial coefficients.

The same is true in this case (as we would expect: it is just a different basis) but the computations are remarkably simpler.

We first observe that the "coefficients" of our polynomial are in fact the "values" of $f(z)$ and certain of its derivatives at distinct nodes. Thus a good backward stability result would be something like "The computed eigenvalues z_i, $1 \le i \le d$, are the exact roots of $\tilde{f}(z)$, which satisfies

$$\tilde{f}(\tau_i) = \rho_{i0} + \kappa_i \varepsilon + O(\varepsilon^2) \tag{6}$$

where κ_i is a moderate constant, and the perturbations $\Delta\mathbf{A}$ and $\Delta\mathbf{B}$ in $\{z_i\} = \{z | \det((\mathbf{B}+\Delta\mathbf{B})z - (\mathbf{A}+\Delta\mathbf{A})) = 0\}$ satisfy $\|\Delta\mathbf{A}\|$, $\|\Delta\mathbf{B}\| \le \varepsilon$".

One complication is that $\deg \tilde{f}(z) = d + 2$; however the two highest degree terms $\left[z^{d+1}\right]\tilde{f}$ and $\left[z^{d+2}\right]\tilde{f}$ are $O(\varepsilon)$ so we regard this as satisfactory.

We may say quite a bit about κ_i: it is proportional to $\max_{j,k} |\rho_{jk}|$, to $\max_{j,k} \frac{|\gamma_{jk}|}{|\gamma_{i,s_i-1}|}$, and inversely proportional to $\min_{j\neq i} |\tau_i - \tau_j|^{s_j}$, as shown below.

Lemma 1. *If z_i are the exact generalized eigenvalues of*

$$\mathbf{A}_0 + \Delta\mathbf{A}, \quad \mathbf{B}_0 + \Delta\mathbf{B} \tag{7}$$

where $(\mathbf{A}_0, \mathbf{B}_0)$ is as in (3) and $\Delta\mathbf{B} = \varepsilon\mathbf{E}$, $\Delta\mathbf{A} = \varepsilon\mathbf{F}$ where $\|\mathbf{E}\|$, $\|\mathbf{F}\| \le 1$, then

$$\begin{aligned} \tilde{f}(\tau_i) &= \det\left(\tau_i(\mathbf{B}_0 + \Delta\mathbf{B}) - (\mathbf{A}_0 + \Delta\mathbf{A})\right) && (8)\\ &= \det\left(\tau_i\mathbf{B}_0 - \mathbf{A}_0 + \varepsilon(\tau_i\mathbf{E} - \mathbf{F})\right) && (9)\\ &= \rho_{i0} + \mathrm{tr}(\rho_{i0}(\tau_i\mathbf{B}_0 - \mathbf{A}_0)^{-1}(\tau_i\mathbf{E} - \mathbf{F}))\varepsilon + O(\varepsilon^2) && (10) \end{aligned}$$

Proof. A direct consequence of the derivative formula. □

The structure of $(\tau_i\mathbf{B}_0 - \mathbf{A}_0)^{-1}$ is surprisingly simple:

$$\begin{bmatrix} \mathbf{J}_{s_1}^{-T}(\tau_i - \tau_1) & & \mathbf{v}_1 & & & \\ & \ddots & \mathbf{v}_2 & & & \\ & & \mathbf{v}_3 & -\mathbf{I}_{s_i-1} & & \\ \mathbf{h}_1 & \mathbf{h}_2 & \sigma & \mathbf{h}_3 & \mathbf{h}_4 & -\gamma_{i,s_1-1}^{-1} \\ & & \mathbf{v}_4 & \ddots & & \\ & & \mathbf{v}_5 & & \mathbf{J}_{s_n}^{-T}(\tau_i - \tau_n) & \\ & & \rho_{i0}^{-1} & & & 0 \end{bmatrix} \tag{11}$$

where

$$\sigma = \frac{1}{\rho_{i0}\gamma_{i,s_i-1}} \left(\sum_{\substack{j=1\\ j\neq i}}^{n} \sum_{k=0}^{s_j-1} \sum_{\ell=0}^{k} \frac{\gamma_{jk}\rho_{j,k}}{(\tau_i - \tau_j)^{k+1-\ell}} - \sum_{k=0}^{s_i-2} \gamma_{ik}\rho_{i,k+1} \right), \tag{12}$$

$\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_4, \mathbf{v}_5$ all contain expressions linear in

$$\frac{\rho_{jk}}{\rho_{i0}(\tau_i - \tau_j)^{k+1-\ell}} \quad 0 \le \ell \le k \le s_j - 1 \tag{13}$$

and $\mathbf{v}_3$ contains the ratios $\frac{\rho_{ij}}{\rho_{i0}}$ for $1 \le j \le s_i - 1$. Similarly $\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_4$ contain expressions linear in

$$\frac{\gamma_{jk}}{(\tau_i - \tau_j)^{k+1-\ell}} \quad 0 \le \ell \le k \le s_i - 1 \tag{14}$$

and $\mathbf{h}_3$ contains the ratios $\frac{\gamma_{ij}}{\gamma_{i,s_i-1}}$ for $1 \le j \le s_i - 1$.

Remark 1. *A bound on κ_i is given by*

$$|\kappa_i| \le |\rho_{i0}|(d+2)(|\tau_i|+1)\|(\tau_i\mathbf{B}_0 - \mathbf{A}_0)^{-1}\|_\infty \tag{15}$$

Remark 2. *This analysis confirms our experience, and our expectation that near-confluency (small $\tau_i - \tau_j$) will cause problems, and that widely-varying γ_{ij} such as occur with equally-spaced nodes will also cause problems.*

One must also consider derivatives, for example

$$\begin{aligned} \tilde{f}'(\tau_i) =& \det\left(\tau_i(\mathbf{B}_0 + \Delta\mathbf{B}) - (\mathbf{A}_0 + \Delta\mathbf{A})\right) \times \\ & \mathrm{tr}\left((\tau(\mathbf{B}_0 + \Delta\mathbf{B}) - (\mathbf{A}_0 + \Delta\mathbf{A}))^{-1} (\mathbf{B}_0 + \Delta\mathbf{B}) \right) \\ =& \rho_{i1} + \varepsilon\mathrm{tr}[\rho_{i1}(\tau_i\mathbf{B}_0 - \mathbf{A}_0)^{-1}(\tau_i\mathbf{E} - \mathbf{F}) \\ & + \rho_{i0}(\tau_i\mathbf{B}_0 - \mathbf{A}_0)^{-1}\left(\mathbf{E} - (\tau_i\mathbf{E} - \mathbf{F})(\tau_i\mathbf{B}_0 - \mathbf{A}_0)^{-1}\right)] \\ & + O(\varepsilon^2) \end{aligned}$$

and of higher order.

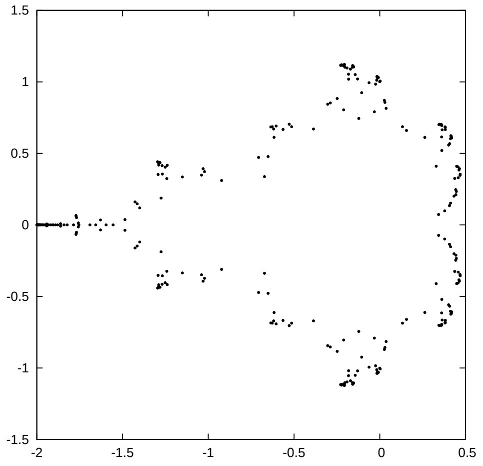


Figure 1: Roots of the Mandelbrot polynomial $p_8(z)$

4. EXAMPLE

The Mandelbrot polynomial $p_k(z)$ of degree $2^k - 1$ is defined through the recurrence relation:

$$p_0(z) = 1, \quad p_{k+1}(z) = zp_k(z)^2 + 1$$

The roots of this family of polynomials can be investigated by constructing the companion matrix pencil (3) for some particularly nice choices of interpolation nodes. At the roots $\xi^{(k)}$ of $p_k(z)$, the polynomial $p_{k+1}(z)$ takes on the value 1 and derivative 0, using these $2^{k+1}-2$ points and the points at $z = 0$ (as $p_k(0) = p'_k(0) = 1\,\forall k$), we construct the companion matrix pencil $(\mathbf{A}_0, \mathbf{B}_0)$ for the interpolant and locate the eigenvalues, giving the roots $\xi^{(k+1)}$ of $p_{k+1}(z)$.

The maximum backward error for the 255 roots of $p_8(z)$ is 2.4214×10^{-13} and 6.8781×10^{-9} for the derivative. Comparing this to our upper bounds of 1.7621×10^{-10} and 5.3936×10^{-7} we see satisfactory agreement.

5. ACKNOWLEDGEMENT

This work was supported by the Natural Sciences and Engineering Research Council of Canada.

6. REFERENCES

[1] V. I. Arnold. On matrices depending on parameters. *Russian Mathematical Surveys*, 26(2):29–43, 1971.

[2] R. Corless, A. Shakoori, D. Aruliah, and L. Gonzalez-Vega. Barycentric Hermite interpolants for event location in initial-value problems. *JNAIAM*, 3(1-2):1–18, 2008.

[3] A. Edelman and H. Murakami. Polynomial roots from companion matrix eigenvalues. *Mathematics of Computation*, 64(210):763–776, April 1995.

[4] N. J. Higham, R. C. Li, and F. Tisseur. Backward error of polynomial eigenproblems solved by linearization. *SIAM journal on matrix analysis and applications*, 29(4):1218–1241, 2008.

[5] D. Lemonnier and P. Van Dooren. Optimal scaling of block companion pencils. In *International Symposium on Mathematical Theory of Networks and Systems, Leuven, Belgium*, 2004.

A Construction of Injective Parameterizations of Domains for Isogeometric Applications

[Extended Abstract]

Thien Nguyen
Inst. of Applied Geometry
JKU Linz,
Austria

Bernard Mourrain
GALAAD
INRIA Sophia Antipolis
France

André Galligo
Nice University
Nice
France

Gang Xu
Comp. Sci. Dep.
Hangzhou Dianzi Univ.
China

ABSTRACT

We present a new method to construct a B-spline parameterization of a domain defined by its boundary curves or surfaces. The method is based on solving Laplace equations on the physical domain. The equations are then pulled back to the parameter domain to deduce an elliptic system of Partial Differential Equations with boundary conditions. This system, solved by relaxation techniques, yields the required parameterization.

Categories and Subject Descriptors

I.3.5 [**Computational Geometry and Object Modeling**]: Curve, surface, solid, and object representations, Splines

General Terms

Algorithms

Keywords

Isogeometric analysis, Surface parameterization, elliptic PDE

1. INTRODUCTION

The problem of parameterizing a domain defined by its boundary curves (or surfaces) has long been discussed in computer graphics and geometric modeling due to its important role in many applications. The well-known Gordons-Coons method, which is represented in almost all standard literature, see [3], is a simple but suitable to solve this problem for many cases. Later, Farin and Hansford [2] proposed discrete Coons patches which minimizes a surface energy to produce good shape Coons patches. With the recent development of isogeometric analysis (IGA for short) (see [4]), solving this problem once again is a big challenge to produce prerequisite geometry for IGA. In IGA, the computational domains are solid objects which are represented by tensor product B-splines or NURBS and the discretization space is spanned by B-splines or NURBS basis. However, in practice, the NURBS representations of solid objects are not always available since CAD systems usually only provide parameterization of the boundary, i.e. mappings from the parametric domains to the boundary of the physical domain, as described in [1].

Motivated by IGA, Xu et al. [8] have used the discrete Coons method as the initial solution for the iterative method minimizing a functional arising from shape optimization to obtain a better parameterization of a planar shape. Basically, Xu et al.'s method consists of finding a mapping from the parametric domain (usually the unit square) to the physical domain which is initially injective via suited inequalities on the coordinates of inner control points and then of minimizing some quality functionals. There the important point is to achieve injectivity in the initialization and keep it during the optimization process.

On the other way around, especially for volume parameterization, there are several methods for finding a mapping from the physical domain to the parametric domain by solving Laplace equations or minimizing some energy functionals. Then the physical domain is reconstructed by the approximation of the inverse mapping.

In [6], a parameterization method for a generalized cylinder-type volume defined by a tetrahedral mesh is proposed based on the investigation of discrete volumetric harmonic functions. In another paper [5], the authors also solve Laplace equations on the physical domain by the fundamental solution method to find the mapping between solid objects given that the surface mapping is available to be the boundary conditions for the PDEs.

In the previous works, Laplace equations are solved with respect to the coordinate functions separately. A different approach is given in [7]: the first coordinate function is found by minimizing a Dirichlet energy, then the second coordinate function is found by minimizing its harmonic energy on the level sets of previously found coordinate function. The important point is that this method naturally guarantees the injectivity of the mapping from the physical domain to the parameter domain. Once this mapping is obtained, the spline approximation for the inverse of the mapping is found by the least squares fitting method.

Inspired by these works, we propose a new iterative algorithm to obtain an injective parameterization of the physical domain from the initial parameterization by solving Laplace equations on the physical domain, via splines defined on the parameter domain.

2. PARAMETERIZATION CONSTRUCTION

We describe this method for a planar simply connected domain bounded by B-spline curves, but it can be extended straighforwardly to volumes bounded by B-spline surfaces. Our objective is to construct a parametrisation σ from a square domain $S = [0, 1] \times [0, 1]$ into the domain Ω which boundary is formed by the curves $\Gamma_1, \ldots, \Gamma_4$. We search a map σ in the B-spline basis associated to the parametric domain S of the form:

$$\sigma(\xi, \eta) = (x(\xi, \eta), y(\xi, \eta)) = \sum_{i=0}^{d_1} \sum_{j=0}^{d_2} N_i^{d_1}(\xi)\, N_j^{d_2}(\eta)\mathbf{p}_{ij}, \quad (1)$$

where $(N_i^{d_1}(\xi))_{i=0,\ldots,d_1}, (N_j^{d_2}(\eta))_{j=0,\ldots,d_2}$, are the bases of our spline spaces in ξ and η. In order to construct an injective map from S to Ω, we solve Laplace equations

$$\Delta\xi(x, y) = 0, \Delta\eta(x, y) = 0, \quad (2)$$

with boundary conditions; namely

- $\xi(x, y) = 0$ on Γ_1, $\xi(x, y) = 1$ on Γ_3 (Dirichlet) and $\nabla\xi(x, y) \cdot \mathbf{t}_{\partial\Omega}(x, y) = 0$ on Γ_2, Γ_4 (Neumann),
- $\eta(x, y) = 0$ on Γ_2, $\eta(x, y) = 1$ on Γ_4 and (Dirichlet) and $\nabla\eta(x, y) \cdot \mathbf{t}_{\partial\Omega}(x, y) = 0$ on Γ_1, Γ_3 (Neumann),

where $\mathbf{t}_{\partial\Omega}(x, y)$ is tangent to the boundary $\partial\Omega$ at the point $(x, y) \in \partial\Omega$. The equations (2) are pulled back in the parametric domain and give

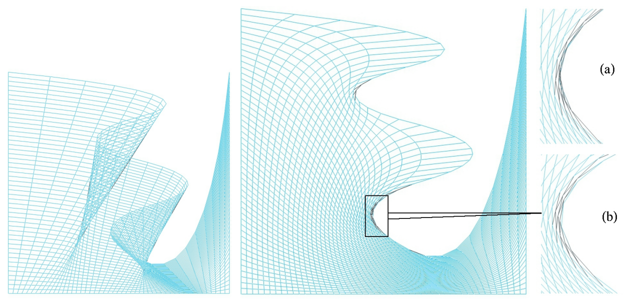
$$(x_\eta^2+y_\eta^2)\frac{\partial^2 U}{\partial\xi^2} - 2(x_\xi x_\eta + y_\xi y_\eta)\frac{\partial^2 U}{\partial\xi\partial\eta} + (x_\xi^2+y_\xi^2)\frac{\partial^2 U}{\partial\eta^2} = 0 \quad (3)$$

for $U = x(\xi, \eta)$ and $U = y(\xi, \eta)$.

Then the equation (3) is solved by a relaxation technique, which consists at step k to use the computed functions $x^k(\xi, \eta)$, $y^k(\xi, \eta)$ instead of $x(\xi, \eta)$ and $y(\xi, \eta)$ in the equation (3), in order to compute the solutions $x^{k+1}(\xi, \eta)$, $y^{k+1}(\xi, \eta)$ at step $k+1$. If U is expressed in the B-spline basis associated to the parametric domain S, each iteration requires the solution of a linear system in the coefficients $\mathbf{c}$ and the evaluation of the functions $x_\xi, x_\eta, y_\xi, y_\eta$ at some well-chosen interpolation points in S. The initial parametrisation $(x^0(\xi, \eta), y^0(\xi, \eta))$ is constructed using Coons patches [2].

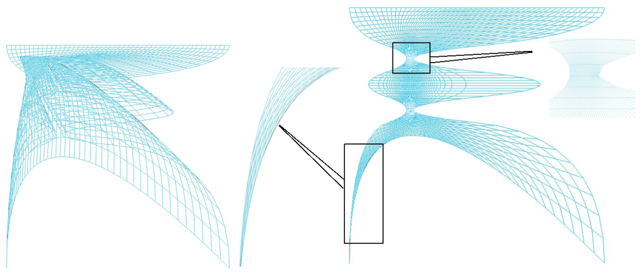
3. EXPERIMENTATION

We demonstrate our method by two examples in 2D. The implementation is conducted in the Axel[1] platform. In the first example, we start with a shape defined by 4 B-spline curves. Then we construct the Coons surface showed in the left picture. We solve the PDE system and check if the resulting surface is injective or not by the method in [8]. If not, we do knot insertion and consider the resulting surface as the initial solution in this next step. The right picture is the resulting after 4 steps of knot insertion. The parameterization has 82×82 control points and is injective. The zoom-in picture (a) is the result for 3 steps and it is still not injective. We get the better result in (b) after 4 steps.



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

The second example is a shape with very narrow segments. The left picture shows very tangled initial Coons surface. After 4 steps of knot insertion, we get the non-self-overlapping shape in the right.



In the comparison with the method described in [7], the advantages of our method are that the boundary curves are kept exactly as the original ones and the PDEs are solved in the parameter domain instead of the physical domain. Both methods seem to have difficulties with the high curvature regions where the iso-curves are nearly parallel.

The method is extendible to 3D by solving a PDE system of three variables. In the full paper, we will report on practical experimentation of this approach in 3D. We will test different boundary conditions and homotopy techniques to map one solution for a simple domain into a solution for a domain with a more complex boundary.

4. REFERENCES

[1] E. Cohen, T. Martin, R.M. Martin, T.Lyche, and R.F. Riesenfeld. Analysis-aware modeling: Understanding quality considerations in modeling for isogeometric analysis. *Comput.Methods Appl.Mech.Engrg*, 2009.

[2] G. Farin and D. Hansford. Discrete Coons patches. *Comput. Aided Geom. Des.*, 16(7):691–700, 1999.

[3] J. Hoschek and D. Lasser. *Fundamentals of computer aided geometric design.* Wellesley, 1993.

[4] T.J.R Hughes, J.A. Cottrell, and Y. Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry, and mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, 194(39-41):4135–4195, 2005.

[5] X. Li, H. Xu, S. Wan, Z. Yin, W. Yu. Feature-aligned harmonic volumetric mapping using MFS. *Computers & Graphics*, 34(3):242-251, 2010.

[6] T. Martin, E. Cohen, and R.M. Kirby. Volumetric parameterization and trivariate B-spline fitting using harmonic functions. *Computer Aided Geometric Design*, 26(6):648-664, 2009.

[7] T. Nguyen, B. Jüttler. Parameterization of Simply Connected Domains Using Sequences of Harmonic Maps. submitted to the Proceedings of Curves and Surfaces, Avignon, France, 2010.

[8] G. Xu, B. Mourrain, R. Duvigneau, and A. Galligo. Parametrization of computational domain in isogeometric analysis: methods and comparison. *Computer Methods in Applied Mechanics and Engineering*, In Press, Accepted Manuscript, 2011.

Synthetic division in the context of indefinite summation

[Extended Abstract]

Eugene Zima
Wilfrid Laurier University, Waterloo, Canada
ezima@wlu.ca

Categories and Subject Descriptors

G.4 [**Mathematical Software**]: Algorithm design and analysis

General Terms

Theory

Keywords

Efficient indefinite summation, synthetic division

Let $\mathbb{K}$ be a field of characteristic zero, x – an independent variable, E – shift operator with respect to x, i.e. $Ef(x) = f(x+1)$ for an arbitrary expression $f(x)$. The problem of indefinite summation in general is: given a closed form expression $F(x)$ to find a closed form expression $G(x)$, which satisfies the first order linear difference equation

$$(E-1)G(x) = F(x). \tag{1}$$

If found, write $G(x) = \sum_x F(x)$. Otherwise one can try to solve the *additive decomposition problem*: construct two closed form expressions $R(x)$ and $H(x)$, such that

$$F(x) = (E-1)R(x) + H(x), \tag{2}$$

where $H(x)$ is simpler than $F(x)$ is some sense (or as variation, as simple as possible). If closed form $G(x)$ satisfying (1) exists one takes $R(x) = G(x)$ and $H(x) = 0$ as answer to additive decomposition problem. The measure of simplicity can be different for different classes of functions. For example if $F(x)$ is a rational function over $\mathbb{K}$ one requires both $R(x)$ and $H(x)$ to be rational with $H(x)$ having the denominator of the lowest possible degree.

Let $F = f/g \in \mathbb{K}(x)$, with $f, g \in \mathbb{K}[x] \setminus \{0\}$ coprime. Using division with remainder to split off the polynomial part, which can be summed using, e.g., conversion to the falling factorial basis, we may assume that $\deg f < \deg g$, so that F is *proper*. Let ρ be the *dispersion* of F, the maximal integer distance between roots of the denominator g. If $\rho = 0$ then we take $R = 0$ and $H = F$ in (2), see [1, 3]. In fact, the condition that the denominator of H has minimal degree is equivalent to the dispersion of H being zero. Now, let $\rho > 0$.

It is very well known that, for the indefinite rational summation problem, the bit size of the dense representation of the output $G(x)$ in 1 can be exponentially large in the bit size of the dense representation of the input $F(x)$. For instance, if $F(x) = \frac{1}{x^2+\rho x}$, (with ρ natural number) then the solution (up to an additive constant) of the rational summation problem (1) is

$$G(x) = -\frac{1}{\rho}\left(\frac{1}{x} + \frac{1}{x+1} + \cdots + \frac{1}{x+\rho-1}\right),$$

whose numerator and denominator have degrees linear in ρ, and coefficients of bit-size linear in ρ.

We will consider two kinds of dependency of the running time of a summation algorithm on dispersion ρ in cases when ρ is exponentially large in the input size:

- *essential* (non-removable) dependency, when an algorithm is at least linear in ρ^ϵ for some positive $\epsilon \leq 1$ and the expanded output has size also linear in ρ^ϵ;
- *non-essential* (potentially removable) dependency, when an algorithm is at least linear in ρ^ϵ, but the expanded output has size polynomial in the input size.

The essential dependency is a feature of the summation problem (output happens to be exponentially large in the input size) and non-essential dependency is a feature of some particular algorithm (output is small but the algorithm exhibits exponentially large intermediate expressions).

Although the history of algorithmic treatment of the rational summation is relatively long (starting with the classical works of S.Abramov [1, 2]), many of existing algorithms still exhibit non-essential dependency of the running time on ρ which makes them unnecessary slow for the cases of large ρ and small output size.

Consider two popular problems associated with the indefinite rational summation (as in [6]):
P1: For a given proper rational function $F(x)$ find a pair of rational functions $R(x)$ and $H(x)$, satisfying (2) with $H(x)$ having denominator of minimal possible degree.
P2: For a given nonsummable proper rational function $F(x)$ find a solution of problem **P1** with minimal possible degree of the denominator of the summable part $R(x)$.

In formulation **P1** the problem of rational summation was considered in series of works (see for example [1, 2, 3, 4]). The classical algorithms for indefinite rational summation ([1, 3]) compute the dense representation of the output, and involve a number of polynomial computations in $\mathbb{K}[x]$ that is linear in the dispersion ρ of the input. Their bit (and arithmetic) complexity is potentially exponential in the bit size of $F(x)$, even for the mere decision problem. A recent algorithm [4] computes a "compact representation" of the output in polynomial complexity in the bit size of $F(x)$; it thus provides a polynomial complexity decision procedure for rational summability. The algorithm from [4] is an example of an algorithm that has only essential dependency of running time on the dispersion ρ of the input in the case when the input is rational summable. For example,

$$\sum_x \frac{-2\,x+999}{(x+1)\,(x-999)\,x\,(x-1000)} = \frac{1}{x\,(x-1000)}. \qquad (3)$$

Here dispersion of the input $\rho = 1000$, but running time of the algorithm from [4] will depend polynomially on the degree of the input denominator. A variant of this algorithm converts the compact representation of the output into the dense one, and it is output sensitive (while it inherently has exponential complexity in the worst case - for instance, when the output $G(x)$ has the size exponential in the input size.)

However, when the input is not summable this algorithm can still exhibit non-essential dependency of the running time on ρ. The reason is, in this case the problem of additive decomposition has infinitely many solutions with different summable and nonsummable parts. Consider simple example from [6] (Example 1). The following additive decomposition is a solution to particular problem **P1**

$$\sum_x \frac{x^2-100}{x\,(x+1)\,(x+100)} = \frac{2}{x}+\frac{1}{x+1}+\frac{1}{x+2}+\cdots+\frac{1}{x+99}+\sum_x \frac{1}{x}. \qquad (4)$$

with the denominator of nonsummable part of smallest degree. This is the form of the output that is returned by the algorithm from [4]. However, the same summation problem has another additive decomposition

$$\sum_x \frac{x^2-100}{x\,(x+1)\,(x+100)} = \frac{1}{x}+\sum_x \frac{1}{x+100} \qquad (5)$$

with the denominator of nonsummable part of the smallest degree. The difference of these decompositions is that the degree of the denominator of the summable part in (5) is minimal. Thus (5) is also a solution to problem **P2** in this case.

In formulation **P2** the problem was partially solved in [5], and [6] gives complete solution. However it suffers from two major drawbacks:
- as a preliminary step it uses classical algorithm from [3], which exhibits non-essential dependency of the running time on the dispersion of the input (probably author was unaware of [4]);
- the output of this preliminary step is "massaged" and transformed into the minimal form, however the size of the input of this step can already be exponential in the size of $F(x)$ even when the resulting minimal form is small.

This means that both steps of the algorithm from [6] exhibit non-essential dependency of the running time on the dispersion of the input: in order to produce the minimal possible output (as in the right-hand side of (5)) the algorithm first builds the right-hand side of (4), and then postprocesses it.

Our goal is to remove all cases of non-essential dependencies of the running time of indefinite rational summation algorithm on dispersion ρ of the input. We developed simple modification of the algorithm from [4] that exhibits only essential dependency even in cases when the input is not rational summable. It is based on *shiftless* factorization of polynomials and direct divisibility test. When the rational function is not summable our algorithm provides solution to the rational decomposition problem with minimal degree of the denominator of the rational part. The algorithm uses elementary properties of the left quotient of the simplest synthetic division in the noncommutative ring $\mathbb{K}[x,E]$. Namely, given linear difference operators $L_1, L_2, \ldots, L_k$ from $\mathbb{K}[x,E]$ such that not all of them are left divisible by $E-1$ our algorithm combines succinct representation of the divisors with the careful simultaneous choice of the reminder terms of the same form $c_iE^l, i=1,\ldots,k$. These remainder terms form nonsummbale part of the output and guarantee that the summable part has the denominator of smallest degree. We also show that our approach based on divisibility test provides clear and elementary explanation of the structure of solution of **P2**.

Prototype implementation of our algorithm in Maple outperforms implementations described in [4] and [6] especially in cases when dispersion ρ is large, input is nonsummable and minimal possible output is small.

As in [4] all but the very last step of our algorithm have polynomial in the input size running time. It can exhibit essential dependency of the running time on large dispersion only at the very last step - conversion of the compact representation of the output into the expanded form. In this case the running time will be proportional to the size of the output, as in the following example.

$$\sum_x \frac{2\,x^3+300\,x^2+10099\,x+4950}{x\,(x+100)\,(x+99)\,(x+1)} =$$

$$= \frac{3}{2x}+\frac{1}{x+1}+\cdots+\frac{1}{x+98}+\frac{1}{2(x+99)}+\sum_x \frac{2}{x}$$

The results are also extended to the case of quasi-rational indefinite summation with additional distinguishing property. Namely, if the quasi-rational function is summable, then the running time of the proposed algorithm is always polynomial in the input size.

1. REFERENCES

[1] S. A. Abramov. The summation of rational functions. *Ž. Vyčisl. Mat. i Mat. Fiz.*, 11:1071–1075, 1971.

[2] S. A. Abramov. The rational component of the solution of a first order linear recurrence relation with rational right hand side. *Ž. Vyčisl. Mat. i Mat. Fiz.*, 15(4):1035–1039, 1090, 1975.

[3] S. A. Abramov. Indefinite sums of rational functions. In *Proceedings of the 1995 international symposium on Symbolic and algebraic computation*, ISSAC '95, pages 303–308, New York, NY, USA, 1995. ACM.

[4] J. Gerhard, M. Giesbrecht, A. Storjohann, and E. V. Zima. Shiftless decomposition and polynomial-time rational summation. In *Proc. of ISSAC'03*, pages 119–126, New York, 2003. ACM.

[5] R. Pirastu. A note on the minimality problem in indefinite summation of rational functions. In *Séminaire Lotharingien de Combinatoire (Saint-Nabor, 1993)*, volume 1994/21 of *Prépubl. Inst. Rech. Math. Av.*, pages 95–101. Univ. Louis Pasteur, Strasbourg, 1994.

[6] S. P. Polyakov. Indefinite summation of rational functions with additional minimization of the summable part. *Programmirovanie*, (2):48–53, 2008.

GCD of multivariate approximate polynomials using beautification with the subtractive algorithm[*]

[Extended Abstract] [†]

Robert M. Corless
rcorless at uwo dot ca

Erik Postma
epostma at maplesoft dot com

David R. Stoutemyer
dstout at hawaii dot edu

Categories and Subject Descriptors

I.1.2 [**Computing Methodologies**]: Symbolic and Algebraic Manipulation Algebraic Algorithms

General Terms

Polynomial algorithms

Keywords

Approximate polynomial, GCD

1. OVERVIEW

The GCD problem for approximate polynomials, by which we mean polynomials expressed in some fixed basis but having approximately-known coefficients, has been well-studied at least since the paper of [6]. Important papers include those listed in [4, 2.12.3], and more recently includes [5], [8] and [9]. What is new about the present paper is that we hope to take advantage of some new technology, in order to improve our understanding of the GCD problem and not necessarily to try to improve on existing algorithms.

The paper [3] describes an AddThenBeautify function that, given two expressions and a requested number of significant digits, can justifiably artificially underflow some or all of the terms in the sum of the two expressions. The goals of that article included how to present approximate answers in an economical and comprehensible way. One of the techniques presented in that paper might also be useful in preventing 'roundoff-induced expression swell', in which a term that is supposed to be multiplied by a zero coefficient is mistakenly retained, with all the concomitant extra work that such a mistake entails. It is this goal that we pursue here.

This article shows how approximate polynomial gcd calculations can give more useful results by using this function together with a subtractive Euclidean gcd algorithm that chooses between annihilating the lead term or the 0-degree term based on minimizing rounding errors. This work is similar in some sense to that of [7] but relies on a separate theoretical development and implementation.

2. BEAUTIFICATION

The main goal of [3] was to develop a justifiable, scale-invariant, deterministic way to artificially underflow terms to 0, based only on a relative tolerance such as significantDigits. Another goal was a justifiable way to round to fewer digits coefficients in terms that always have correspondingly smaller magnitude than some other term.

Given a generalized multivariate polynomial[1] $P(\mathbf{z})$ with n complex variables $z_1, z_2, \ldots, z_n$, and a tolerance ε, typically expressed as

$$\varepsilon = \frac{1}{2} \cdot 10^{-\mathsf{significantDigits}}, \tag{1}$$

the BeautyClinic package finds a 'more beautiful' expression $P_B(\mathbf{z})$ such that

$$|P(\mathbf{z}) - P_B(\mathbf{z})| \le \varepsilon \|P(\mathbf{z})\| \tag{2}$$

for all $\mathbf{z} \in \mathbb{C}^n$ except possibly in the neighborhood of isolated singular points, and where $\|P\|$ is a norm related to the coefficients of P.

The definition of 'beauty' used ensured that 0 was the most beautiful object. This had the desired effect of removing unwanted terms. It is this feature of the BeautyClinic approach that we use in this attack on the GCD problem.

3. SUBTRACTIVE IS ATTRACTIVE

One of the ugly facts about approximate GCD computation, which motivated in part the great stream of work since [6], is that *an approximate GCD is not preserved by the Euclidean (division) algorithm.* Given the definition of approximate GCD used in, say, [2], it is not true that the approximate GCD of p and q is the same as the approximate GCD of q and q rem p. In particular, Schönhage's original example shows that one can have $1 = \gcd(p, q)$ for any reasonable definition of approximate GCD but that a degree one approximate GCD occurs for q and q rem p.

The remainder sequence for the Schönhage example is, with the Euclidean division algorithm,

$$z^4 + z + 1, z^3 - \eta\, z, \eta\, z^2 + z + 1, \left(1 - \eta - \eta^3\right) z + 1, \tag{3}$$

[1]A *generalized univariate polynomial* in z is 0, or else a term of the form bz^β with complex $b \ne 0$ and complex indeterminate z and real β, or else a sum of two or more such non-zero terms.

followed by a constant. For any value of η with $|\eta| < 0.4$, the original two polynomials have only the trivial GCD, and no approximate GCD unless the tolerance is $O(1)$. However, for small η, the third and fourth elements of the sequence have an approximate GCD near $z+1+\eta+O(\eta^2)$, where any tolerance of $O(\eta^2)$ will do. This fact is fatal for the Euclidean algorithm. Also, since the result is exact, in formal power series, no possible numerical stabilization can be achieved.

It is a surprise that the subtractive algorithm appears to behave differently. To be concrete, the sequence produced by using the subtractive algorithm to cancel leading terms gives the sequence

$$z^4+z+1, z^3-\eta\, z, \eta\, z^2+z+1, -z^2+\left(-\eta^2-1\right)z, \\ \left(-\eta^3-\eta+1\right)z+1, \eta\left(\eta^4+2\,\eta^2-\eta+1\right)z, \quad (4)$$

followed by a constant, where now no two successive elements have roots that differ by less than $O(\eta)$.

We investigate the behavior of this algorithm with beautification, and present evidence that this approach produces a more robust result.

4. MULTIVARIATE GCD AND THE OVERALL APPROACH

We here describe an implementation of this algorithm, extended to the multivariate case. In this implementation, we combine many ideas described previously, some ideas that seem to be novel, and the use of the beautifying algorithms described above and in [3].

The leading idea in the implementation is to factor the polynomials into a part primitive with respect to its most main remaining variable, and the corresponding content. This factorization is determined by recursively computing the GCD of the coefficients. The content can then be factored similarly with respect to the next variable. In particular, if polynomials $p_1, \ldots, p_n$ in variables $z_1, \ldots, z_m$ (in decreasing order) are given, we factor each p_i as $p_{i,1}p_{i,2}\cdots p_{i,m}$, with each $p_{i,j}$ a primitive polynomial in $z_j, \ldots, z_m$. We can now compute the GCDs of $p_{1,j}, \ldots, p_{n,j}$ for $j = 1, \ldots, m$, knowing that the result will be primitive in each case, and multiply these GCDs to obtain the GCD of $p_1, \ldots, p_n$.

The knowledge that the intermediate results will be primitive often enables shortcuts. For example, if the sets of variables V_p and V_q occurring in primitive polynomials p and q are different, then their GCD can be obtained as the GCD of the polynomials in $V_p \cap V_q$, occurring as coefficients of the variables in $V_p \setminus V_q$ in p, and as coefficients of the variables in $V_q \setminus V_p$ in q. The number of polynomials may now have grown, but their size is smaller. This is typically an advantage, in particular if one of the new polynomials is a constant.

There are several notions of primitivity that play a part in the implementation and the supporting lemmas. The strongest notion used, and the one that makes the above statements work, defines a primitive polynomial to be one where:

- there is a nonzero constant term,
- the (polynomial) coefficients in the recursive representation of the powers of the main variable are coprime,
- all numeric coefficients in the fully expanded representation are coprime (for a notion of GCD that is well-defined).

At a lower level, one step of the subtractive algorithm consists of taking an original pair of polynomials p, q and constructing $r = ap + bq$ from it. We can choose a and b using one of three strategies:

- so that the leading coefficients cancel, and retain the primitive part of r and the smallest of p and q; e.g. take $a = \operatorname{lcoeff}(q)$ and $b = -\operatorname{lcoeff}(p)z^d$ where d is the difference in degrees between p and q.
- so that the trailing coefficients cancel, and retain the primitive part of r modulo a power of the main variable, and the smallest of p and q;
- compute both versions of the primitive part of r and retain them instead of p and q.

This last *double step* option stems from Chrystal [1]. The decision is similar to pivoting decisions in Gaussian elimination, and requires the same sort of considerations.

Acknowledgments

This work funded in part by the Natural Sciences and Engineering Research Council of Canada.

5. REFERENCES

[1] G. Chrystal. *Algebra: an elementary text book for the higher classes of secondary schools and for colleges, vol. 2.* Chelsea, New York, 1964.

[2] R. M. Corless, P. M. Gianni, B. M. Trager, and S. M. Watt. The Singular Value Decomposition for polynomial systems. In *International Symposium on Symbolic and Algebraic Computation*, pages 195–207, Montréal, Canada, 1995.

[3] R. M. Corless, E. Postma, and D. R. Stoutemyer. Rounding coefficients and artificially underflowing terms in non-numeric expressions. *Communications in Computer Algebra*, to appear, 2011.

[4] J. Grabmeier, E. Kaltofen, and V. Weispfenning. *Computer algebra handbook: foundations, applications, systems.* Berlin ; New York : Springer, 2003.

[5] E. Kaltofen, Z. Yang, and L. Zhi. Approximate greatest common divisors of several polynomials with linearly constrained coefficients and singular polynomials. In *International Symposium on Symbolic and Algebraic Computation*, pages 169–176, New York, NY, USA, 2006.

[6] A. Schönhage. Quasi-gcd computations. *Journal of Complexity*, 1:118–137, 1985.

[7] K. Shirayanagi and H. Sekigawa. A new Gröbner basis conversion method based on stabilization techniques. *Theoretical Computer Science*, 409(2):311–317, 2008.

[8] A. Terui. An iterative method for calculating approximate GCD of univariate polynomials. In *International Symposium on Symbolic and Algebraic Computation*, pages 351–358, New York, NY, USA, 2009.

[9] Z. Zeng. Algorithm 835: Multroot—a Matlab package for computing polynomial roots and multiplicities. *ACM Transactions on Mathematical Software*, 30(2):218–236, 2004.

Empirical Study of an Evaluation-Based Subdivision Algorithm for Complex Root Isolation

Narayan Kamath
Google, Inc
and
Oxford Computing Lab, UK
kamath.narayan@gmail.com

Irina Voiculescu
Oxford Computing Lab
Oxford University, UK
irina@comlab.ox.ac.uk

Chee K. Yap*
Courant Institute, NYU, USA
and
Oxford Computing Lab, UK
yap@cs.nyu.edu

ABSTRACT

We provide an empirical study of subdivision algorithms for isolating the simple roots of a polynomial in any desired box region B_0 of the complex plane. One such class of algorithms is based on Newton-like interval methods (Moore, Krawczyk, Hansen-Sengupta). Another class of subdivision algorithms is based on function evaluation. Here, Yakoubsohn discussed a method that is purely based on an exclusion predicate. Recently, Sagraloff and Yap introduced another algorithm of this type, called `Ceval`. We describe the first implementation of `Ceval` in Core Library. We compare its performance to the above mentioned algorithms, and also to the well-known `MPSolve` software from Bini and Florentino. Our results suggest that certified evaluation-based methods such as `Ceval` are encouraging and deserve further exploration.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems —*Geometrical Problems and Computations* ; I.1.2 [**Symbolic and Algebraic Manipulation**]: Algorithms—*Algebraic Algorithms*

General Terms

Algorithms, Theory

Keywords

complex root isolation, solving bi-variate systems, subdivision algorithms, `Ceval` algorithm interval Newton methods

*This author's work was carried out on a sabbatical visit to Oxford Computing Lab (2009-10) and is supported by NSF Grants CCF-0728977 and CCF-0917093.

1. INTRODUCTION

Isolating roots of univariate polynomials is a highly classical subject in the mathematical and computational fields. From a complexity viewpoint, the problem of isolating all complex zeros of an integer polynomial (we call this the "benchmark problem") has been intensely studied [30, 31, 25]. The basic conclusion from Schönhage's classic paper [30] says the benchmark problem has bit-complexity $\widetilde{O}(n^3 L)$ where n is the degree and L a bound on the bit sizes of each coefficient ($\widetilde{O}$ means we ignore logarithmic terms in n and L). However, practitioners tend to favor other methods that do not match this asymptotic complexity. Among the many current implementations, we have the highly regarded `MPSolve` from Bini and Florentino [2], and an optimized Descartes method from Rouillier and Zimmermann [28], albeit for real roots. For instance, `Maple`'s default algorithm for finding real roots is based on the latter implementation [28].

Complex roots of univariate polynomials can be viewed as a special case of solving bivariate real systems. The latter has been the focus of several recent papers [1, 11, 8, 7]. Our work can also be regarded as a special case of the general problem of determining the topology of a collection of planar curves.

This paper is an empirical study of several complex root isolation methods based on **domain subdivision**. The basic paradigm is repeated subdivision of an initial box $B_0 \subseteq \mathbb{R}^2$, analogous to binary search. In Figure 1(a,b,c,d), we provide a visualization of the subdivision boxes produced by four of the algorithms to be discussed. The polynomial whose roots were isolated here is $x(9x^9 + 7x^8 + 8x^7 + 8x^6 + 6x^4 + 5x^3 + 2x^2 + 1) = 0$.

Many algorithms in this area can be viewed as having 2 or 3 phases, beginning with a subdivision phase. See [16] for a description of this framework. Such algorithms go back to an algorithm of Mitchell [17] for finding real roots of univariate polynomials to higher dimensional analogues for computing isotopic curves and surface approximations [26, 32].

This phase is controlled by an associated predicate $C(B)$ where $B \subseteq B_0$ is any axes-parallel box. Starting with B_0, we keep subdividing a box $B \subseteq B_0$ into two or four subboxes until every box B satisfies $C(B)$.

The complexity of this subdivision phase often determines the asymptotic complexity of the algorithm. Subdivision algorithms are popular with implementers because of merits such as ease of implementation, local complexity, and adap-

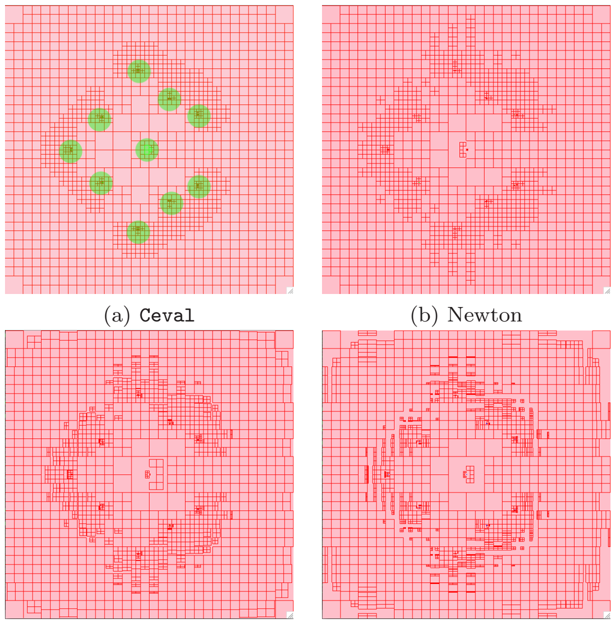

(a) `Ceval` (b) Newton (c) Krawczyk (d) Hansen-Sengupta

Figure 1: Roots of $x(9x^9+7x^8+8x^7+8x^6+6x^4+5x^3+2x^2+1)=0$

tive complexity. By “local complexity”, we mean that the computational effort can be localized to a region-of-interest like B_0; in contrast, the benchmark problem has global complexity. We remark that the predicate $C(B)$ can be algebraic or geometric ones (cf. [16]) but we are most interested in numeric ones.

The subdivision predicate $C(B)$ is typically a disjunction of an **exclusion** $C_{out}(B)$, and an **inclusion** $C_{in}(B)$, predicate. If a box is excluded, i.e., satisfies $C_{out}(B)$, it may be discarded, unless we wish to hold it for visualization purposes. Ultimately, we need a **confirmation predicate** which determines that box B has a unique root. We remark that $C_{in}(B)$ is a necessary but not sufficient condition for confirmation (this is dealt with in subsequent phases). Conceptually, the set of boxes form a **subdivision tree** T, and the the goal of subdivision is to ensure that every leaf of T is either excluded or included. The **effectiveness** of a predicate $C(B)$ is measured by the overall complexity of the subdivision process. Effectiveness of numerical predicates is seen to depend on two factors: **(predicate) efficiency** and **efficacy**. Efficiency measures the computational effort to evaluate each predicate. Efficacy measures the size of the subdivision tree T. The effectiveness of a predicate can be measured (lower bounded) by the product of the size of T and worst case cost to evaluate any predicate in T. However, it is shown in [29] that such a product bound may be a factor of n larger than the true overall complexity of subdivision. In general, effectiveness involves a tradeoff between efficacy and (predicate) efficiency. Such a efficiency-efficacy tradeoff for real root isolation is discussed in [29] (in the context of comparing predicates based on Sturm, Descartes and Bolzano principles). Numerical predicates are typically constructed from interval functions, and so predicate efficacy depends on the range of interval functions; Stahl [33] contains a comprehensive study of range functions from this efficacy-complexity viewpoint, and also its application to the solution of systems of nonlinear equations. This paper is a contribution along similar lines, except that we deal with the specific case of a bivariate system from the real and imaginary curves $u(x,y)$, $v(x,y)$ of a complex polynomial $p(x+\mathbf{i}y)=u(x,y)+\mathbf{i}v(x,y)$ ($\mathbf{i}=\sqrt{-1}$).

We first look at a class of predicates from the certified computation literature [19, 23]. For any real function $f:\mathbb{R}^n\to\mathbb{R}^m$, we postulate a corresponding interval function $\square f:\square\mathbb{R}^n\to\square\mathbb{R}^m$ where $\square\mathbb{R}$ is the set of closed real intervals. Typically, $m=1$ or $m=n$. We call $\square f$ a **box function** for f if it satisfies two properties: for all n-boxes $B, B_i\in\square\mathbb{R}^n$, (a) [Inclusion Property] $f(B)\subseteq\square f(B)$ and, (b) [Convergence Property] for any point $p\in\mathbb{R}^n$, we have $\square f(B_i)\to f(p)$ as $B_i\to p$ ($i\to\infty$). Although Newton's method is traditionally used for root refinement, Nickel [24], Moore [20], and Hansen [13] have shown that the interval forms can serve as confirmation predicate for roots. Such predicates form the basis for root isolation algorithms [18, 33]. We focus on three forms of Newton-type predicates: an interval Newton predicate due to Moore [19], Krawczyk's predicate [20, 15], and Hansen and Sengupta's predicate [12]. These Newton operators have the form $f:\mathbb{R}^n\to\mathbb{R}^n$ and their inclusion/exclusion predicates have the form

$$\left.\begin{array}{ll}\text{(inclusion)} & \square f(B)\subseteq B\\ \text{(exclusion)} & \square f(B)\cap B=\emptyset\end{array}\right\}\tag{1}$$

where B is a box and $\square f$ is a box function for f. Thus, the efficacy of these predicates may be reduced to the following order relation $\succeq$ on their underlying operators: we write “$\square f\succeq\square g$” if $\square f(B)\subseteq\square g(B)$ for all boxes B. Thus, $\square f\succeq\square g$ implies that, for predicates of the form (1), the predicates defined by $\square f$ will succeed whenever the predicates defined by $\square g$ succeed. I.e., $\square f$ is at least as efficacious as $\square g$. From Neumaier [23], we have:

$$\text{Interval Newton}\succeq\text{Hansen-Sengupta}\succeq\text{Krawczyk}.$$

The corresponding Krawczyk predicate is therefore the least efficacious among the three. However, these operators do not have equal computational costs: the interval Newton operator involves the inversion of an interval matrix, while the other two operators do not. Our aim is to implement all three predicates on a common platform, and compare their overall performance in isolating the roots of a wide variety of polynomials.

Another source of predicates comes from function evaluation [10, 36, 34, 6, 5]. Yakoubsohn and Dedieu [10] studied an exclusion predicate $C_{out}(B)$ that is basically an interval form of Taylor expansion. Their inclusion predicate $C_{in}(B)$ amounts to the ε-cutoff predicate (i.e., B is included if its diameter is $<\varepsilon$). Since ε is arbitrary, some adaptivity is lost and the algorithm only produces a collection of candidate boxes, which may contain no roots or multiple roots. Recently, Sagraloff and Yap [29] introduced another evaluation-based algorithm `Ceval`, but with inclusion and confirmation predicates. We provide the first implementation of `Ceval`, and compare these evaluation-based algorithms to each other, and to the interval Newton-type predicates above. We note that for the benchmark problem, the paper [29] proved that `Ceval` has bit complexity of $\widetilde{O}(n^4L^2)$, thus matching the best bit complexity for Descartes method or Sturm methods. However, the `Ceval` complexity is a stronger result, since

Descartes and Sturm methods only isolate real roots.

The last algorithm in our comparative study is MPSolve, a complex root algorithm from Bini and Florentino [2, 4]. This is a highly regarded implementation based on the Aberth-Erlich method, not subdivision. As such it is a global method (it must approximate all roots simultaneously). But no convergence proof is known for this method.

Our implementation is released as part of the free and open source Core Library [38, 9]. We exploit the unique ability of Core Library to compile a program in different levels of numerical accuracy. The work reported here is based on the first author's Masters Thesis [14].

Postscript. A "simplified" version of Ceval was proposed in the latest version of the paper [29]. The experiments reported here are based on the original version of Ceval. In the full version of this paper, we plan to report on the relative performance of these two versions.

2. CONTRIBUTIONS

The contributions of this paper are:

- An implementation of three interval Newton-type root isolation algorithms for bivariate systems on a common platform. We report on their relative performance for complex root isolation.
- The first implementation of the Ceval algorithm.
- Comparison of evaluation-based subdivision methods (Yakoubson, Ceval) with Newton-type methods and with a state-of-art root solver, MPSolve. We discuss algorithm engineering issues arising in these implementations.
- Our empirical evidence suggests that evaluation-based methods for complex roots are vastly superior to the (more general) Newton-type methods, and has competitive advantages compared with current state-of-art methods such as MPSolve.
- Our present work is a contribution to the general area of designing new exact and certified algorithms for approximation of zero sets using evaluation-based methods.
- Distribution of our code with the free open-source Core Library, allowing further experimentation.

3. EXPERIMENTAL SETUP

The implementation platform is a MacBook Pro with an Intel Core 2 Duo processor, clocked at 2.53 GHz with 4 GB of RAM. The operating system is Mac OS X (10.5.8). Our compiler is gcc-4.2, and our code is compiled using its most aggressive optimization flag -O3.

Our algorithms are implemented as C++ programs using the Core Library version 2.0. Recall that the Core Library [38] is a collection of C++ classes to support exact geometric computation (EGC) [37] with algebraic numbers. It is built over the multiprecision libraries of gmp and mpfr. A unique aspect of Core Library is its **numerical accuracy API**, which comes in four levels. Level 1 represents machine accuracy (IEEE 754 Standard). Level 2 represents arbitrary precision number types such as BigInt and BigFloat. Level 3 represents the "EGC Level" and the main number type here is called Expr (Expression). Exact comparison of an Expr is achieved as long as Expr is algebraic. Level 4 allows a mix of number types from all 3 previous levels. These four levels are integrated in the sense that any program that includes Core Library is able to compile its numbers to any these four levels. For instance, a number declared as "double" represents the standard machine double-precision number at Level 1, but it is promoted to a BigFloat in Level 2, and promoted to an Expr in Level 3. This exploits the operator overloading feature of the C++ language. We mainly use Levels 1 and 2. Level 1 is fast, but it limits the size of the polynomials that can be treated. All our experiments are assumed to run at Level 1 unless otherwise indicated.

For this work, two new number types IntervalT (intervals) and ComplexT (complex numbers) are essential. Core Library's polynomial classes are generic (i.e., templated) and this allows us to evaluate a polynomial $p(x)$ at interval valued x or complex number x, independent of the number type of p's coefficients:

```
template <typename T> class Polynomial {
  // Note that T and NT must be
  // either interoperable
  // or implicitly convertible.
  template <typename NT>
       NT eval(const NT &x);
};
```

The compiler can instantiate for NT=IntervalT, NT=ComplexT or NT=BigFloat. Typically, we have T=BigInt or T=int.

Our algorithms are exact because all predicates can be implemented exactly using BigFloats, assuming the input numbers are dyadic (i.e., numbers of the form $m2^n$ where $m, n \in \mathbb{Z}$). Level 1 accuracy is sufficient for exact computation provided no overflow or underflow occurs. In the implementation of Ceval, the 8-point test requires rational numbers, but this will be separately treated below. Speed without correctness is not worth much. So we validate the output of our algorithms in two ways: we compare our computed values to a reliable software such as MPSolve, and we compare Level 1 outputs to the outputs from the same algorithm at Level 2.

The main sources for our test polynomials are the FRISCO suite [35], and randomly generated polynomials. Our programs accepts both the FRISCO file format for polynomials as well as a very flexible string based format such as "$3x^4y^5 - (x^2+1)(y^3-1)$" or "$3x^4y^5 = (x^2+1)(y^3-1)$", which is useful for command line execution. For each run of our algorithm, we collect the following statistics:

- Running Time in microseconds, using C's gettimeofday API. This method is sufficiently reliable for our purposes because our programs are CPU intensive and do not perform any disk or network I/O.
- Number of Boxes Processed. In tables, this number is called "Iters" (as it is the number of iterations of the subdivision loop). Recall that each box is ultimately included, excluded, or subdivided.
- Number of Excluded (resp., Included) Boxes.
- Number of Unresolved Boxes. In principle, our algorithm will eventually terminate if there are no multiple roots in the region of interest. But to prevent

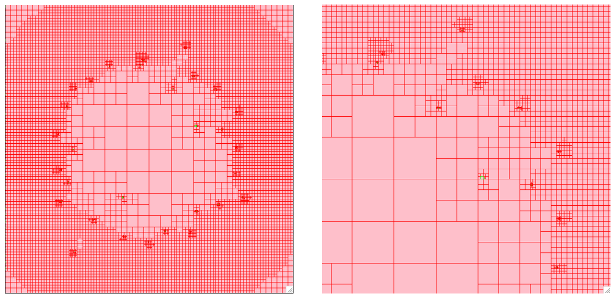

(a) All roots (b) Zoomed in view.

Figure 2: Subdivision tree of `Ceval` for degree 25 polynomial

underflow (level 1), or to keep the running time reasonable, or to avoid nontermination in case the region-of-interest contains multiple roots, we can specify a minimum box size or maximum tree depth; this may result in unresolved boxes on termination.

In our statistical tables, we will underline the entry that is the best among the several methods being compared. Also, the names of the polynomials (for instance, `chebyshev20`) have an integer suffix indicating its degree.

4. VISUALIZATION

Visualization is an important aspect of our software, for debugging and for gaining insight into the behavior of algorithms. Here we extend the previous visualization for curves analysis from [16], which is based on the `OpenGL` library. Typically we wish to visualize a subdivision tree (Figure 2(a)) and to zoom and pan into details of interest (Figure 2(b)). To do this we need to retain boxes that have been excluded in subdivision; command line flags control the handling of these extra information. The color code shows red for excluded boxes, green for confirmed boxes, blue for unresolved boxes.

Although the termination of the `Ceval` algorithm depends on the assumption that the input polynomial is square-free, the algorithm is still useful for arbitrary polynomials. Suppose we run `Ceval` on the polynomial $p(z) = (z^6+64)^2(z^6-729)$. This is a non-squarefree polynomial with 6 multiple roots on the circle (centered at the origin) of radius 2, and 6 simple roots on a concentric circle of radius 3. It is now essential to specify a minimum box size (or maximum subdivision depth) for termination. Choosing a minimum box size of 0.0001, and $B_0 = [-4,4] \times [-4,4]$, `Ceval` produces the visualization seen in Figure 3. It processed a total of 8645 boxes, confirming 6 green boxes and leaving unresolved 256 blue boxes. The simple roots were all isolated as shown in this output:

```
m= [1.498046875 + (-2.599609375)i], r= .01171875
m= [-1.501953125 + (-2.599609375)i], r= .01171875
m= [2.998046875 + (.001953125)i], r= .01171875
m= [1.498046875 + (2.599609375)i], r= .01171875
m= [-3.001953125 + (.001953125)i], r= .01171875
m= [-1.501953125 + (2.599609375)i], r= .01171875
```

Further details about our implementation may be found in the thesis [14] and in the `Core Library` distribution (under `progs/mesh`).

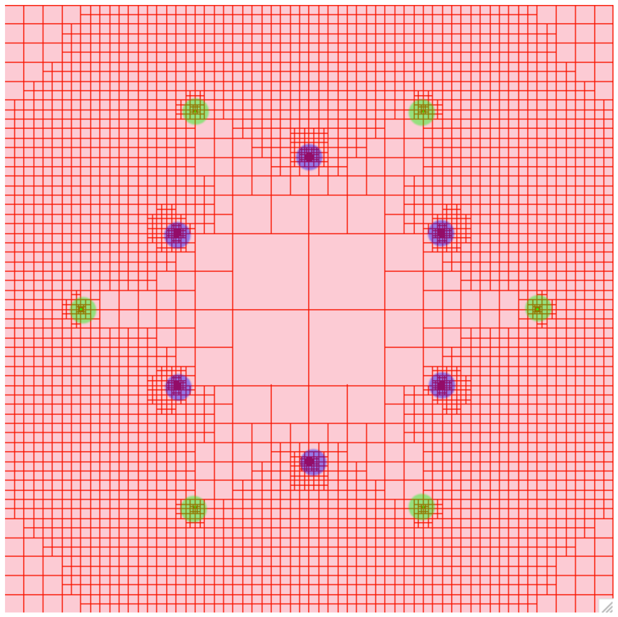


Figure 3: Subdivision tree for $p(z) = (z^6+64)^2(z^6-729)$. Blue boxes are unresolved ones around double roots, and green boxes isolates the simple roots.

5. INTERVAL NEWTON-TYPE METHODS

First, we consider three closely related pairs of predicates based on Interval Newton methods. They work by finding the solutions of the bivariate system $u(x,y) = v(x,y) = 0$ where $f(x+\mathbf{i}y) = u(x,y) + \mathbf{i}v(x,y)$. But these methods work more generally for any zero-dimensional system of n polynomials in n variables, so we describe them in these general terms. We view an n-tuple B of vectors as an n-dimensional box (or n-box), $B \subseteq \square\mathbb{R}^n$.

The **Interval Newton operator** $N : \square\mathbb{R}^n \to \square\mathbb{R}$ for a function $f : \mathbb{R}^n \to \mathbb{R}$ is given by

$$N(B) = N_f(B) := m(B) - (\square J(B))^{-1} \cdot f(m(B)) \qquad (2)$$

where $(\square J(B))^{-1}$ denotes the inverse of the interval Jacobian of f, and $m(B)$ denotes the midpoint of box B. For bivariate systems, computing the inverse is not really an issue, but we must address the situation where $\square J(B)$ contains 0.

The **Krawczyk operator** is defined by

$$K(B) = K_f(B) := y - Y \cdot f(y) + \{I - Y \cdot \square J(B)\} \cdot (B - y) \quad (3)$$

where $y \in B$, Y is any nonsingular real matrix, and I the identity matrix. Typically, we have $y \simeq m(B)$ and $Y \simeq J(y)^{-1}$, viewed as a preconditioner.

The Hansen-Sengupta approach is a Gauss-Seidel iteration to solve for x in the equation

$$\square J(B)(x-y) + f(y) = 0. \qquad (4)$$

Multiplying by some preconditioning matrix Y as before, the equation becomes $A(x-y) = -Yf(y)$ where $A := Y \cdot \square J(B)$ (cf. (3)). We can write A as $A = U + D + L$ (disjoint sum of a strict upper triangular U, a diagonal D, and a strict lower triangular L matrix).

The Gauss-Seidel iteration for solving $Ax = b$ is given by

$$x^{(k+1)} = D^{-1}(b - Lx^{(k+1)} - Ux^{(k)})$$

where $x^{(k)}$ is the k^{th} approximation, yielding the **Hansen-Sengupta operator**

$$H(B) = H_f(B) := y - D^{-1}\{Y \cdot f(y) + L(B' - y) + U(B - y)\} \tag{5}$$

where y, Y are as before and $B' = B \cap H(B)$. Note that the use of LB' in (5) is not recursive, but iterative because L is lower-triangular: we replace each component of box B by corresponding entries of $B \cap H(B)$ as they become available.

Let G be any of the three operators from (2,3,5) above. It can be shown [19, 24, 20, 21] that

(i) $G(B) \subseteq B$ implies B has a unique root

(ii) $G(B) \cap B$ contains all the roots in B

As a consequence of (ii), if $G(B) \cap B = \emptyset$ then B has no roots. This provides an exclusion predicate C_{out} for subdivision.

Clearly (i) provides a confirmation predicate, and hence an inclusion predicate C_{in}. Since this inclusion predicate is also a confirmation, the interval Newton-type algorithms do not need additional phases beyond the subdivision phase.

Another consequence of (ii) is that, in case the inclusion or exclusion predicates fail on B, we can replace B by $B' := B \cap G(B)$ and then subdivide B'. Although this results in smaller boxes, there is an associated cost because the bitsize (of the coordinates) of B' may increase considerable. If we simply subdivide B, the bitsize increases by a small constant number (depending on how you count bitsize of B). The effect of using B' instead of B implies that the shapes of boxes are rather arbitrary. This results in irregular children boxes as seen in Figure 1(b,c) but especially Figure 1(d) because of extended interval arithmetic.

§1. The Exclusion Predicate $C_0(B)$.

Instead of the exclusion predicate $G(B) \cap B = \emptyset$ above, we can use predicate $C_0(B)$, defined as $0 \notin \square u(B)$ or $0 \notin \square v(B)$. (cf. [16]). Preliminary experiments show that $C_0(B)$ is more efficient, and all our Newton-type algorithms use this predicate. The interval $\square u(B)$ can be computed easily using the Horner scheme. But our tests show that using centered form [27] of $\square u(B)$ yields remarkable improvement in efficacy over Horner. This becomes more apparent with increasing degree of the polynomial u. This gain in efficacy must be balanced against the cost of computing centered forms, which amounts to computing the bivariate Taylor coefficients of our polynomials when expanded at the midpoints of a box B. We achieve a balance by using a simple scheme: recompute the coefficients after a fixed number T (threshold) of predicate failures.

It is worth pointing out that this strategy is sensitive to the order in which we process boxes, a "depth first" type processing will yield better results than a "breadth first" type processing. Table 1 shows the influence of T on timing and subdivision for the Chebyshev polynomial of degree 20. For this particular example, $T = 16$ gives the best overall time (indicated by the underlined entry). As a general default, we have empirically set $T = 4$ in all our experiments.

§2. Comparing the Inclusion Predicates.

We now return to the inclusion predicate $G(B) \subseteq B$ where $G(B)$ is one of the operators $N(B), K(B)$ or $H(B)$. These operators admit several variants in their implementation.

T	Iterations	Ambiguous boxes	C0 Excludes as % of total	Time (secs)
0	15429	176	11396(73.8)	18.554
2	19845	504	13877(71.2)	3.410
4	23585	744	15218(64.5)	2.190
8	31229	1332	16936(54.2)	1.834
16	40585	2396	17201(42.3)	<u>1.720</u>
32	56945	5908	17700(31.0)	1.951
64	77949	9164	17772(22.7)	2.445

Table 1: Effect of Threshold T on Chebyshev Polynomial of degree 20

For the interval Newton operator (2), when $J(B)$ contains a singular matrix, we have a choice to use extended interval arithmetic (to support division by intervals containing zero) or we could choose to subdivide B. The latter turns out to be a consistently better choice.

For the Krawczyk operator (3), we could choose the non-singular matrix Y to be one of the following: $J(m(B))^{-1}$, $m(\square J(B)^{-1})$, or identity I. The first choice turns out to be best.

For the Hansen-Sengupta operator we choose to use extended interval arithmetic in its implementation. See [14] for more detail. In any case, we choose the best variant for each inclusion predicate and then compare them against each other. This final comparison is shown in Table 2.

The three operators are not very different from each other. It is somewhat surprising that Hansen-Sengupta did not perform better, but this may be because our special problem of complex roots does not allow the Hansen-Sengupta method to shine. Another issue is that these methods do not converge in case of roots on the boundary of a subdivision box; this is treated by Stahl [33] and also in Kamath's thesis [14].

6. CEVAL & YAKOUBSOHN'S ROOT ISOLATION ALGORITHM

For any function $f : \mathbb{C} \to \mathbb{C}$, and constant $K > 0$, Sagraloff and Yap [29] introduced the function $t_K^f : \mathbb{C} \times \mathbb{R}_{\geq 0} \to \mathbb{R}$ defined as follows:

$$t_K^f(m, r) = K \sum_{k \geq 1} \left| \frac{f^{(k)}(m)}{f(m)k!} \right| r^k. \tag{6}$$

The function $M(z, t)$ introduced by [36, 10] corresponds to the case $K = 1$. The predicate $T_K^f(m, r)$ is then defined as "$t_K^f(m, r) < 1$". We are also interested in the predicate $T_K^{f'}(m, r)$ where f' is the derivative of f. If f is understood, we write $T_K(m, r)$ and $T'_K(m, r)$ instead of $T_K^f(m, r)$ and $T_K^{f'}(m, r)$.

For the T-predicates, circular geometry is more natural than box geometry. Let $D(m, r)$ denote the disk centered at m with radius r. It is clear that $T_1(m, r)$ is an exclusion predicate for the disc $D(m, r)$; it is also shown that if $T'_{\sqrt{2}}(m, r)$ holds then B has at most one root; therefore we use this as an inclusion predicate.

In order to have a confirmation predicate, we need to ensure that there is at least one root in B. For this purpose, the following 8-**point test** was developed by Sagraloff-Yap:

Polynomial	Hansen-Sengupta		Newton		Krawczyk	
	# iter	Secs.	# iter	Secs.	# iter	Secs.
cheby20	18837	24.448	15445	19.731	15429	20.064
chrma22	7621	10.986	6557	9.380	6637	9.578
chrmc23	11229	19.637	9893	16.191	9845	15.328
hermite20	2805	3.453	2645	3.278	2645	3.344
kam3-1	2005	0.439	2005	0.438	2005	0.435
kam3-2	2005	0.444	2005	0.432	2005	0.446
laguerre20	1205	1.491	1037	1.294	1021	1.272
laguerre4	205	0.011	229	0.013	213	0.012
laguerre5	197	0.016	197	0.021	197	0.017
laguerre6	245	0.030	245	0.149	245	0.036
$x^{10} - 1$	3253	1.100	2645	0.796	2629	0.809
$x^{20} - 1$	12693	15.729	10053	12.373	10053	12.267
wilk20	1133	1.433	869	1.110	861	1.096

Table 2: Comparison of Hansen-Sengupta, Newton, and Krawczyk

consider 8 compass points $m + 4re^{\mathbf{i}j\pi/4}$ for $j = 0, \ldots, 7$ on the boundary of $D(m, 4r)$ (not $D(m, r)$). We give them the conventional names N, S, E, W, NE, NW, SE, SW, as shown in Figure 4.

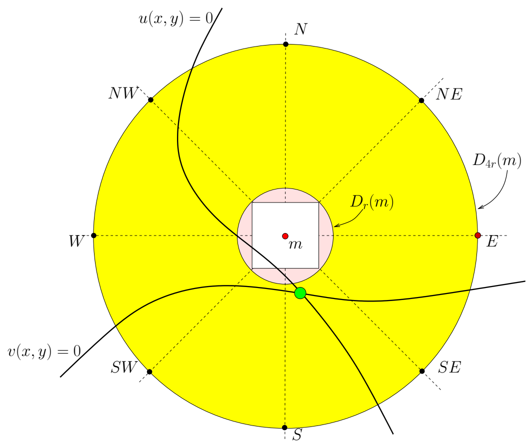


Figure 4: 8 **compass points on** D_{4r}**.**

They define 8 arcs $A_0, \ldots, A_7$ where $A_j = \{u(m+4re^{\mathbf{i}\theta}) : j\pi/4 \leq \theta < (j+1)\pi/4\}$. We say there is an (arcwise) u-**crossing** of A_i if the value of u at the endpoints of A_i changes sign. For instance, the endpoints of A_1 is N and NE, and there is a u-crossing at A_1 if $u(N)u(NE) < 0$. If there are exactly two u-crossings at A_j and A_k, there are exactly two v-crossings at $A_{j'}$ and $A_{k'}$, and these crossing interleave (i.e., either $j < j' < k < k'$ or $j' < j < k' < k$), then we say the 8-**point test passes** for $D(m, 4r)$; otherwise the test fails. The key result is this:

Theorem 1 (Sagraloff-Yap). *Suppose that the predicate $T_6'(m, 4r)$ holds, and the* 8*-point test is applied to the disc $D(m, 4r)$.*

(i) If the test fails, then $D(m, r)$ is not isolating.

(ii) If the test passes, the $D(m, 4r)$ is isolating.

By choosing $m = m(B)$ and r to be the radius of box B, this test allows us to reject B (if the test fails) or to accept a larger disk $D(m, 4r)$. There is an issue of exactness of this test: although the four cardinal points (N, S, E, W) are dyadic, the other four ordinal points (NE, NW, SE, SW) are irrational. It is shown [29] that the 8-point test remains valid when we approximate the ordinal points by other points on the boundary of $D(m, 4r)$, provided their angular deviation from the perfect positions is less than $2.5°$. In particular, we can use Pythagorean triples to provide rational approximations for NE, NW, SE, SW. Such approximate points lie exactly on the boundary of $D(m, 4r)$ and can be chosen arbitrarily close to NE, NW, SE, SW.

The simplest Pythagorean triple that is sufficiently close is $(20, 21, 29)$, with $\arcsin(20/29) \approx 43.60°$. A more accurate triple is $(119, 120, 169)$ with $\arcsin(119/169) = 44.76°$. Using such triples, the 8-point test can be implemented without error using rational arithmetic (not BigFloats).

There are several ways to implement `Ceval`: let m, r denote the midpoint and radius of B. The exclusion predicate is given by $C_{out}(B) \equiv T_1(m, r)$. The inclusion predicate $C_{in}(B)$ can be defined be the conjunction of $T_6'(m, 4r)$, $T_{\sqrt{2}}'(m, 8r)$, and the passing of the 8-point test on $D(m, 4r)$.

By contrast, Yakoubsohn's algorithm is much simpler: it has the same exclusion predicate, but the inclusion predicate is that $r < \varepsilon$ (for some $\varepsilon > 0$). The requirement $T_{\sqrt{2}}'(m, 8r)$ ensures that if two confirmed discs intersect, we can discard any one of them. Thus we output exactly one isolating disk per root.

Let us first look at the raw performance of `Ceval` in Table 3. For each test polynomial, we have two experiments: first to isolate all the roots in the box $[-2, 2] \times [-2, 2] \subseteq \mathbb{C}$, and next to isolate all complex roots. In the latter case, we use the usual Cauchy bound (basically maximum modulus of all the coefficients) to determine a box containing all roots. As expected, the latter takes more iterations and more time. We do not need a head-to-head comparison between `Ceval` with the Newton-type algorithms because `Ceval` is 3 orders of magnitude faster.

We now turn our attention to the comparison of `Ceval` with Yakoubsohn's method. We need to choose a fixed ε: one natural choice is to use the standard root separation bound estimate. But this choice would really slow down Yakoubsohn's algorithm. For our timings, we therefore choose a fixed value of ε depending on the Core Level. At Level 1, we choose $\varepsilon = 0.0001$.

A performance comparison between these two approaches can be found in Table 4. As expected, Yakoubsohn's method is faster since it does not bear the burden of confirming roots. However, in our tests the `Ceval` algorithm always operates on fewer boxes because we can stop subdivision once a box satisfies our inclusion predicate.

Polynomial	$[-2,2]\times[-2,2]$		All Roots	
	Iters	Time(ms)	Iters	Time(ms)
random10	1909	1.741	2615	2.342
nroots10	2037	1.838	2037	1.816
chebyshev20	12805	26.262	18533	39.821
nroots20	7989	16.593	7989	16.444
laguerre20	805	1.747	38253	79.055
hermite20	1685	3.680	17093	35.618
wilk20	581	1.371	40589	97.393
chrma22	4949	10.978	36749	80.626
chrmc23	7101	16.840	43389	107.063
random30	16013	62.602	27413	110.893
random40	27419	178.732	45722	315.381
random50	43160	427.576	71905	743.922
random60	60757	843.580	107746	1495.772
random70	80215	1481.286	108310	2104.210
random80	111795	2685.381	121129	2946.031
random90	139605	4211.166	221837	6789.341

Table 3: `Ceval` algorithm on $B_0 = [-2,2]\times[-2,2]$. Runtimes are in $\underline{\text{milli}}$ seconds.

	Boxes (Yako.)	Iters		Time(ms)	
		`Ceval`	Yako.	`Ceval`	Yako.
random10	76	$\underline{2615}$	3105	$\underline{2.721}$	2.282
nroots10	96	$\underline{2037}$	2581	$\underline{1.773}$	1.802
chebysh20	176	$\underline{18533}$	19509	38.462	$\underline{35.544}$
nroots20	192	$\underline{7989}$	8885	16.481	$\underline{14.526}$
laguerr20	192	$\underline{38253}$	39845	77.991	$\underline{65.489}$
hermite20	160	$\underline{17093}$	18309	35.690	$\underline{29.955}$
wilk20	128	$\underline{40589}$	41909	84.321	$\underline{69.130}$
random20	147	$\underline{8201}$	8985	19.750	$\underline{14.805}$
chrma22	168	$\underline{36749}$	37661	83.140	$\underline{67.142}$
chrmc23	200	$\underline{43389}$	43813	101.816	$\underline{83.464}$
random30	228	$\underline{27413}$	28441	123.294	$\underline{89.068}$
random40	296	$\underline{45722}$	46957	337.769	$\underline{245.090}$
random50	361	$\underline{71905}$	73347	783.600	$\underline{578.127}$
random60	426	$\underline{107746}$	109317	1646.602	$\underline{1191.557}$
random70	511	$\underline{108310}$	110064	2213.269	$\underline{1623.679}$
random80	581	$\underline{121129}$	122990	3021.738	$\underline{2347.435}$
random90	665	$\underline{221837}$	223842	6839.414	$\underline{5385.514}$

Table 4: Comparison of `Ceval` and Yakoubsohn's method with $\varepsilon = 0.0001$. Observe that Yakoubsohn's method is always faster, yet `Ceval` is comparable because it always operates on a fewer number of boxes.

Column 2 in Table 4 shows the number of output boxes from Yakoubsohn's algorithm for each test polynomial. For instance, the first entry is a random polynomial of degree 10, but 76 boxes are output. We expect that at least 66 of these boxes are spurious (assuming no roots lie on the boundary of a box). In our tests, his algorithm tend to produce an average of 8 output boxes around each root. This may be seen in Table 4 by dividing the number of output boxes by the degree of the polynomial.

Finally, we compare `Ceval` with the `MPSolve` package. The latter is based on the Aberth-Erlich simultaneous iteration. The timings for `MPSolve` were generated using the UNIX `time` command and are not as accurate as the timings generated from our code and are provided as a rough indicator of performance. The list of timings can be found in Table 5.

`Ceval` performs about the same as `MPSolve` for polynomials with degree $n < 30$. For higher degree polynomials, their performance starts to diverge with `MPSolve` being con-

	Time(ms)	
	`Ceval`	`MPSolve`
chebyshev20	39.821	$\underline{27}$
laguerre20	$\underline{79.055}$	87
hermite20	$\underline{35.618}$	81
wilk20	97.393	$\underline{49}$
chrma22	80.626	$\underline{47}$
chrmc23	107.063	$\underline{61}$
hermite40	525.80	$\underline{88}$
wilk40	1142.82	$\underline{153}$

Table 5: Comparison of `Ceval` and `MPSolve`.

sistently faster. At $n = 40$ we see that `Ceval` is generally up to five or eight times slower. One of the factors that works against `Ceval` is the estimate of B_0 from the Cauchy bound. For instance, the Wilkinson's degree 40 polynomial has an estimated $B_0 = [-2048, 2048] \times [-2048, 2048]$, which is much larger than the minimal bound of 40. The number of iterations of the Aberth-Erlich iteration that are required to converge to a root is not directly related to the root bounds, or to the distribution of roots. The behavior of this iteration (and the reason for its seemingly wonderful convergence properties) is not well understood; a discussion can be found in [2].

We must keep in mind that the Aberth-Erlich (or indeed the Weierstrass-Durand-Kerner) iteration does not output a list of isolating boxes, rather just approximations of roots. We have no guarantee that the iteration will converge to "reasonable" approximations. Additionally, the strength of subdivision based methods is that they can operate on tight areas of interest while the simultaneous iteration based methods necessarily have to approximate all roots. Consequently, we cannot provide a comparison of the two methods operating on such a preselected area because the `MPSolve` software [3] does not support such an option.

7. EXPERIMENTS AT LEVEL 2

We now provide a comparison of running times of our algorithms at Core Level 2. Basically, machine double are now replaced by `BigFloat` numbers In terms of implementation, this comes almost for free because of `Core Library`'s ability to reuse the same program, just at the cost of re-compilation.

The performance of the three Newton-type operators is presented in Table 6.

Deg	Iters			Time(ms)		
	HS	N	K	HS	N	K
4	325	333	357	0.320	0.334	0.463
6	877	805	829	2.285	1.769	2.164
8	1557	1317	1349	6.972	5.633	6.559
10	2461	2093	2069	19.066	15.730	17.410
12	3445	2965	2965	41.488	35.037	39.390
14	4677	3997	4021	87.262	71.523	77.699

Table 6: Comparison of Newton type operators at Level 2 on $[-2, 2] \times [-2, 2]$

As in the case of Level 1, there is no significant difference between the three operators. At Level 2, however, the interval Newton operator appears to be faster than the Krawczyk operator by a small but consistent margin. Given that arithmetic operations are more expensive at this Level, the method that processes the fewest boxes is bound to be faster. Also, note that the Hansen-Sengupta operator lags further behind the other two as a result of its expensive extended interval arithmetic computations.

These tests are carried out on randomly generated polynomials of the degrees shown. These have been constructed densely, with each coefficient being separately generated.

As in the case of Level 1, `Ceval` continues to be three orders of magnitude faster than the Newton type operators. The results make up Table 7.

The results of this section show that the performance of our algorithms at Level 2 is between 20 and 50 times slower

Deg	Output (Yako.)	Iters		Time(ms)	
		`Ceval`	Yako.	`Ceval`	Yako.
10	72	1965	2381	0.801	0.876
20	128	7909	8565	12.587	12.257
30	224	16413	17381	65.162	53.582
40	248	29293	30405	203.899	163.549

Table 7: Comparison `Ceval` with Yakoubsohn's pure exclusion approach over $[-2, 2] \times [-2, 2]$

than at Level 1. This is an expected consequence of using extended precision types. To improve performance, it might be advantageous to carry out as many operations as possible at machine precision, and to control precision growth in areas that require it.

At the moment, the `Ceval` implementation is incapable of switching Core levels at run time for portions of the working set; it must run entirely at Level 1 or Level 2, a decision made at compile time. This is one aspect of our implementation that we plan to improve.

8. CONCLUSION AND FUTURE WORK

Our experimental results show that the interval arithmetic based approaches suffer from a serious degradation in performance as the degree n of the polynomial increases. Some of this degradation in performance can be attributed to the overestimation of function range by interval extensions, but methods to compensate for it tend to be computationally expensive. Further, these operators suffer from issues due to roots that lie on box boundaries. Overall, this approach appears to suffer from various practical and performance issues, and its use cannot be recommended.

Our experimental results for the `Ceval` algorithm appear quite encouraging. It is efficient and robust, and works well on a large range of polynomials, of various degrees and with densely generated random coefficients. It provides stronger guarantees than Yakoubsohn's exclusion based approach at a comparable speed. Further, both of these approaches perform three orders of magnitude faster than the interval arithmetic based approaches. However, the performance of the predicate T_K breaks down due to the growth of $n!$ for polynomials of degree $n > 90$. We plan to extend the range of the achievable degrees in Level 1 (see below). But even without these extensions, we believe that the algorithm is an efficient and viable choice for isolating roots of complex polynomials of degree $n < 90$. The following plans for future work are related to the development of `Ceval`-like algorithms.

- We noted that the latest version of the `Ceval` paper (to appear in ISSAC 2011) provided a simplified alternative to the 8-point test. Recall that the success of the predicate $T_1(m, r)$ implies that there is no roots in the disk $D(m, r)$ (see Section 6). The simplification is based on the observation that the failure of $T_1(m, r)$ could serve as an inclusion predicate, albeit for a much larger disk: $D(m, 2nr)$. Both versions have the same asymptotic bound of $\widetilde{O}(n^4L^2)$ bit complexity for the benchmark problem. Since their relative performance in practice is unclear, the full version of this paper will compare these two versions.

- As shown above, just switching from Level 1 to Level 2 causes our algorithms to slow down by a factor of up to 20 to 50. We would like to explore a fixed precision BigFloat that behaves closer to a machine type, but with larger number of bits of precision.

- The 8-point-test evaluates the input polynomial at ordinal compass points using the `BigRat` type. Since the `BigRat` type is represented by a pair of `BigInt` instances (a, b) for $\frac{a}{b}$, the exponentiation of these values to a high power will be slow and the representation will be expensive in terms of memory usage as well. Also, since the `BigRat` representation must always contain no common factors, many expensive GCD operations are required as well. One possible change in this area is to use rounded interval arithmetic to estimate the range of f over the interval lower and upper bound of the approximate value of the ordinal points. Clearly, if the interval is either entirely negative or positive, then so is the sign of f at the exact ordinal point.

- We need to explore approaches that allow T_K to remain performant for higher degree polynomials as well. Some of the directions we can look in include the truncated evaluation of the Taylor series and controlled precision growth (with correct rounding) of our calculations.

- Note that our implementation runs either entirely at Level 1 or entirely at Level 2. It would be desirable to implement a wrapper over a machine precision type that can detect underflows and overflows, and switch calculation over to Level 2. We are currently working on the outline of such a type, but a lot of work remains to be done to test it and integrate it with our `Ceval` implementation.

- Recall that Sturm and Descartes' methods [22, 29] are subdivision methods for real roots. It is possible to generalize both to real solutions of bivariate systems. Their performance against evaluation methods should be of interest.

9. REFERENCES

[1] E. Berberich, P. Emeliyanenko, and M. Sagraloff. An elimination method for solving bivariate polynomial systems: Eliminating the usual drawbacks. In Workshop on Algorithm Engineering and Experiments (ALENEX11), 2011. Jan 22, 2011. San Francisco, California.

[2] D. A. Bini. Numerical computation of polynomial zeroes by means of Aberth's method. Numerical Algorithms, 13:179–200, 1996.

[3] D. A. Bini and G. Fiorentino. MPSolve: manual for the MPSolve package. `http://www.dm.unipi.it/cluster-pages/mpsolve/mpsolve.pdf`.

[4] D. A. Bini and G. Fiorentino. Design, analysis, and implementation of a multiprecision , polynomial rootfinder. Numerical Algorithms, 23:127âĂŞ173, 2000.

[5] M. Burr, F. Krahmer, and C. Yap. Continuous amortization: A non-probabilistic adaptive analysis technique. Electronic Colloquium on Computational Complexity (ECCC), TR09(136), December 2009.

[6] M. Burr, V. Sharma, and C. Yap. Evaluation-based root isolation, 2011. In preparation.

[7] J. Cheng, S. Lazard, L. Peñaranda, M. Pouget, F. Rouillier, and E. Tsigaridas. On the topology of planar algebraic curves. In Proc. 25th Symp. on Comp. Geom. (SoCG'09), pages 361–370, 2009.

[8] J.-S. Cheng, X.-S. Gao, and J. Li. Root isolation for bivariate polynomial systems with local generic position method. Mm research preprints, KLMM, Chinese Academy of Sciences, 2008.

[9] Core Library homepage, since 1999. Software download, source, documentation and links: `http://cs.nyu.edu/exact/core/`.

[10] J.-P. Dedieu and J.-C. Yakoubsohn. Localization of an algebraic hypersurface by the exclusion algorithm. Applicable Algebra in Engineering, Communication and Computing, 2:239–256, 1992.

[11] D. Diochnos, I. Emiris, and E. Tsigaridas. On the complexity of real solving bivariate systems. J. Symbolic Computation, 44:818âĂŞ835, 2009.

[12] E. R. Hansen. On solving systems of equations using interval arithmetic. Math. Comp., 22, 1968.

[13] E. R. Hansen. A globally convergent interval method for computing and bounding real roots. BIT, 16:415–424, 1978.

[14] N. Kamath. Subdivision algorithms for complex root isolation: Empirical comparisons. Master's thesis, Oxford University, Oxford Computing Laboratory, Aug. 2010.

[15] R. Krawczyk. Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken. Computing, 4:187–201, 1969.

[16] L. Lin and C. Yap. Adaptive isotopic approximation of nonsingular curves: the parameterizability and nonlocal isotopy approach. Discrete and Comp. Geom., 45(4):760–795, 2011.

[17] D. P. Mitchell. Robust ray intersection with interval arithmetic. In Graphics Interface'90, pages 68–74, 1990.

[18] R. Moore and S. Jones. Safe starting regions for iterative methods. SIAM J. Num.Analysis, 14(6):1051–1065, 1977.

[19] R. E. Moore. Interval Analysis. Prentice Hall, Englewood Cliffs, NJ, 1966.

[20] R. E. Moore. A test for existence of solution to nonlinear systems. SIAM J. Numer. Anal., 14:611–615, 1977.

[21] R. E. Moore and L. Qi. A successive interval test for nonlinear systems. SIAM J. Numer. Anal., 19:845–850, 1982.

[22] B. Mourrain, F. Rouillier, and M.-F. Roy. The Bernstein basis and real root isolation. In J. E. Goodman, J. Pach, and E. Welzl, editors, Combinatorial and Computational Geometry, number 52 in MSRI Publications, pages 459–478. Cambridge University Press, 2005.

[23] A. Neumaier. Interval Methods for Systems of Equations. Cambridge University Press, Cambridge, 1990.

[24] K. Nickel. On the Newton Method in Interval Analysis. Research Memorandum MRC Technical Summary Report #1136, University of Wisconsin, Madison, 1971.

[25] V. Y. Pan. Solving a polynomial equation: some history and recent progress. SIAM Review, 39(2):187–220, 1997.

[26] S. Plantinga and G. Vegter. Isotopic approximation of implicit curves and surfaces. In Proc. Eurographics Symposium on Geometry Processing, pages 245–254, New York, 2004. ACM Press.

[27] H. Ratschek and J. Rokne. Computer Methods for the Range of Functions. Horwood Publishing Limited, Chichester, West Sussex, UK, 1984.

[28] F. Rouillier and P. Zimmermann. Efficient isolation of a polynomial real roots. J. Comp. and Appl. Math., 162:33–50, 2003.

[29] M. Sagraloff and C. K. Yap. A simple but exact and efficient algorithm for complex root isolation. In 36th Int'l Symp.Symbolic and Alge.Comp. (ISSAC 2011), 2011. To Appear. June 8-11, San Jose, California.

[30] A. Schönhage. The fundamental theorem of algebra in terms of computational complexity, 1982. Manuscript , Department of Mathematics, University of Tübingen. Updated 2004.

[31] S. Smale. The fundamental theorem of algebra and complexity theory. Bulletin (N.S.) of the AMS, 4(1):1–36, 1981.

[32] J. M. Snyder. Interval analysis for computer graphics. SIGGRAPH Comput.Graphics, 26(2):121–130, 1992.

[33] V. Stahl. Interval Methods for Bounding the Range of Polynomials and Solving Systems of Nonlinear Equations. Ph.D. thesis, Johannes Kepler University, Linz, 1995.

[34] G. Taubin. Rasterizing algebraic curves and surfaces. IEEE Computer Graphics and Applications, 14(2):14–23, 1994.

[35] The FRISCO Consortium. FRISCO Polynomial test suite. `http://www-sop.inria.fr/saga/POL/`.

[36] J.-C. Yakoubsohn. Numerical analysis of a bisection-exclusion method to find zeros of univariate analytic functions. J. of Complexity, 21:652–690, 2005.

[37] C. K. Yap. Tutorial: Exact numerical computation in algebra and geometry. In Proc. 34th Int'l Symp. Symbolic and Algebraic Comp. (ISSAC'09), pages 387–388, 2009. KIAS, Seoul, Korea, Jul 28-31, 2009.

[38] J. Yu, C. Yap, Z. Du, S. Pion, and H. Bronnimann. Core 2: A library for Exact Numeric Computation in Geometry and Algebra. In 3rd Proc. Int'l Congress on Mathematical Software (ICMS), pages 121–141. Springer, 2010. LNCS No. 6327.

Refining and Verifying the Solution of a Linear System[*]

[Extended Abstract]

Hong Diep Nguyen
INRIA
LIP (CNRS - ENS de Lyon - INRIA - UCBL)
Université de Lyon
ENS de Lyon
46 allée d'Italie
69007 Lyon, France
hong.diep.nguyen@ens-lyon.org

Nathalie Revol
INRIA
LIP (CNRS - ENS de Lyon - INRIA - UCBL)
Université de Lyon
ENS de Lyon
46 allée d'Italie
69007 Lyon, France
nathalie.revol@ens-lyon.fr

ABSTRACT

The problem considered here is to refine an approximate, numerical, solution of a linear system and simultaneously give an enclosure of the error between this approximate solution and the exact one: this is the *verification* step. Desirable properties for an algorithm solving this problem are accuracy of the results, complexity and performance of the actual implementation. A new algorithm is given, which has been designed with these desirable properties in mind. It is based on iterative refinement for accuracy, with well-chosen computing precisions, and uses interval arithmetic for verification.

Categories and Subject Descriptors

G.1.3 [**Numerical Linear Algebra**]: Error analysis - Linear systems (direct and iterative methods); G.4 [**Mathematical Software**]: Verification

General Terms

Reliability, verification.

Keywords

Scientific computing, numerical linear algebra, verified computations, symbolic-numeric, interval arithmetic, floating-point arithmetic, precision, accuracy.

1. INTRODUCTION

The solution of a linear system using floating-point arithmetic entails roundoff errors. One approach to get exactly the solution consists in representing the coefficients as rational numbers and in solving the linear system exactly. Another approach, developed here, consists in mixing iterative refinement – for a better accuracy – and interval arithmetic – for guarantee – to get a tight enclosure of the exact solution. This latter approach can extend to linear systems with interval coefficients of small width.

The algorithm proposed here evolved from an initial version detailed in Section 2, which is very similar to the best available implementation, namely the `verifylss` routine of IntLab [9]. However, this initial version had 3 flaws: failure when the matrix of the system is too ill-conditioned, loss of accuracy and increase of the execution time when it is ill-conditioned. Working on two of these three weak points led us to the algorithm given in Section 3. Further remarks open the way to a better understanding of the respective strengths and weaknesses of exact computations and interval computations, in Section 4.

[*]Supported by the ANR project EVA-Flo.

2. PROBLEM AND INITIAL ALGORITHM

We consider numerical computations, in other words floating-point arithmetic. Let A be an $n \times n$ matrix and b an n-dimensional vector, both with floating-point coefficients: $A \in \mathbb{F}^{n\times n}$ and $b \in \mathbb{F}^n$. Let us denote by x^* the exact solution of the linear system $Ax = b$, and let $\tilde{x}$ be an approximate solution, computed with a so-called numerical routine (LU with partial pivoting in our experiments, such as the `dgetrf` routine of LAPACK). The vector $\tilde{x}$ has floating-point coefficients.

Let us denote by $e = x^* - \tilde{x}$ the error between the exact and the approximate solutions. We look for an enclosure of e, that is, a vector with interval coefficients $\mathbf{e}$ such that $e \in \mathbf{e}$. Let us note here that intervals are denoted with boldface characters. More precisely, the goal is to develop an algorithm which increases the accuracy of $\tilde{x}$ and computes $\mathbf{e}$. We would like this algorithm to satisfy the following four criteria and we will explain our contribution in this direction:

- $\mathbf{e}$ contains the error $e = x^* - \tilde{x}$;
- the accuracy of the result should be close to the best accuracy offered by the floating-point arithmetic, i.e. the relative error should be close to 2^{-53} in IEEE-754 double precision floating-point arithmetic;
- the complexity of the algorithm should be comparable to the complexity of the numerical algorithm, which is $\frac{2}{3}n^3$: the algorithm must exhibit a complexity $\mathcal{O}(n^3)$ and the constant hidden in the $\mathcal{O}$ must be moderate;

- the actual performance of the implementation should be close to the performance of the numerical routine.

The starting point is to use iterative refinement, and to replace floating-point operations by interval operations when it is appropriate to do so. Indeed, iterative refinement is a method that starts from an approximate solution and "contracts" the error, and contractant iterations are methods of choice in interval arithmetic. An algorithm which uses this idea to refine the solution and enclose the error is the `verifylss` function of IntLab [9], a MatLab toolbox that offers interval arithmetic. Our initial algorithm, called `certifylss`, differs from `verifylss` in the computation of $\mathbf{e}$, the enclosure of the error: `verifylss` employs Krawczyk iteration [4] and the Hansen-Bliek-Rohn-Ning-Kearfott formula [5] whereas `certifylss` uses Gauss-Seidel or Jacobi [4, pp. 131 ff.]. (Let us mention here that computing exactly the solution of this interval linear system is not considered: it is known to be NP-hard [8]). The approximate solution $\tilde{x}$ is then corrected and the correction term is the center of $\mathbf{e}$, the enclosure of the error.

A main difference with numerical iterative refinement is that the algorithm premultiplies, using interval arithmetic, the matrix A of the linear system by an approximate, numerical, inverse R: the resulting system has a matrix which should be close to the identity matrix, or rather which should be an H-matrix (see [4, p. 111] for the definition) and thus which behaves well when it comes to solve a linear system.

```
Algorithm certifylss % in MatLab-like syntax
Input: $A \in \mathbb{F}^{n\times n}, b \in \mathbb{F}^n$
$\tilde{x} = A \setminus b$,    $R = inv(A)$,    $\mathbf{K} = [RA]$
while(not converged)
    $\mathbf{r} = [Rb - \mathbf{K}\,\tilde{x}]$        % $RA(x^* - \tilde{x}) \in \mathbf{r}$
    $\mathbf{e} = \mathbf{K} \setminus \mathbf{r}$        % $x^* - \tilde{x} \in \mathbf{e}$
    $\tilde{x} = \tilde{x} + \mathrm{mid}(\mathbf{e})$,    $\mathbf{e} = \mathbf{e} - \mathrm{mid}(\mathbf{e})$
end
Output: $\mathbf{x} = \tilde{x} + \mathbf{e}$
```

Actually, the iteration needs an initial enclosure $\mathbf{e}_0$ of the error, which is then refined at each step. Determining $\mathbf{e}_0$ is based on a heuristic, that relies on $\mathbf{K}$ being an H-matrix, and that can fail. Experiments indicate that failure occurs when A is ill-conditioned: our guess is that R then is a poor approximate of A^{-1} and that $\mathbf{K}$ is not an H-matrix.

The stopping criterion is reached either when the iteration stagnates (no change for $\mathbf{e}$) or when the approximate solution $\tilde{x}$ is approximate to the last bit, in other words when the width of $\mathbf{x}$ corresponds to a relative error of 2^{-52} on $\tilde{x}$, in IEEE-754 double precision.

As already mentioned, three flaws have been identified: loss of accuracy and large execution time for ill-conditioned matrices, failure of the algorithm for extremely ill-conditioned matrices. A new algorithm has been developed to obviate the first two flaws.

3. NEW ALGORITHM TO IMPROVE AND VERIFY THE SOLUTION

3.1 Reducing execution time: relaxation technique

The major contribution to execution time is the operation $\mathbf{e} = \mathbf{K} \setminus \mathbf{r}$. This is performed iteratively, using Gauss-Seidel method which is intrinsically sequential. Furthermore, interval Gauss-Seidel iterations cannot be expressed using LAPACK (floating-point and thus fast) routines. However, they boil down to a floating-point matrix-vector product when Jacobi iterations are used and when the matrix $\mathbf{K}$ is centered around a diagonal matrix. Thus, to reduce execution time, the matrix $\mathbf{K}$ is inflated into a matrix $\widehat{\mathbf{K}}$ that contains $\mathbf{K}$ (otherwise it would no more be guaranteed that $e \in \mathbf{e}$) and that is centered around a diagonal. This is justified by the fact that $\mathbf{K}$ is expected to be close to the identity matrix. Moreover, Jacobi iterations are used instead of Gauss-Seidel ones. The corresponding algorithm is called `certifylss_relaxed`.

3.2 Increasing the accuracy of the result: well-chosen computing precisions

It is well-known [3, ch. 12] (and already done in `certifylss`) that computing the residual $\mathbf{r}$ is subject to cancellation and must be performed using twice the working precision, for accuracy purpose. Following an idea given in [1], we also compute the current approximate solution $\tilde{x}$ using twice the working precision, and we prove in [6] that this yields a fully accurate result, that is, an error of relative width to 1 ulp. Carefully chosen formulas for the computation of $\mathbf{e}$ yield this accuracy without implying extra, useless computations: terms corresponding to 3 times the working precision are not explicitly computed. Details are given in [6] and are not developed below.

3.3 New algorithm: certifylssx

```
Algorithm certifylssx
Input: $A \in \mathbb{F}^{n\times n}, b \in \mathbb{F}^n$
$\tilde{x} = A \setminus b$,    $R = inv(A)$,    $\mathbf{K} = [RA]$
$\widehat{\mathbf{K}} = \mathrm{inflated}(\mathbf{K})$    % $\widehat{\mathbf{K}}$ is centered on a diagonal matrix
while(not converged)
    $\mathbf{r} = [b - \mathbf{A}\,\tilde{x}]$    % $\mathbf{r}$ in twice the working precision
    $\mathbf{r} = R\mathbf{r}]$
    $\mathbf{e} = \widehat{\mathbf{K}} \setminus \mathbf{r}$
    $\tilde{x} = \tilde{x} + \mathrm{mid}(\mathbf{e})$    % $\tilde{x}$ in twice the working precision
    $\mathbf{e} = \mathbf{e} - \mathrm{mid}(\mathbf{e})$
end
Output: $\mathbf{x} = \tilde{x} + \mathbf{e}$
```

The performance of this new algorithm, in terms of accuracy (maximal componentwise relative width of the final $\mathbf{e}$) and execution time, is shown on Figure 1, each dot being the average on 10 matrices of dimensions 1000×1000 generated using the `gallery (randsvd, ...)` command of MatLab. These curves illustrate that we succeeded in obviating the two target problems: we obtained a solution accurate to the last bit, and we verified this accuracy, for the whole range of validity of our algorithm. We succeeded in doing so with acceptable execution time: the theoretical overhead factor is 6 and the practical factor is 15. Finally, let us also note that when the algorithm fails, it does so quickly, without wasting computational time.

4. CONCLUSION, FUTURE WORK, COMPLEXITY COMPARISONS

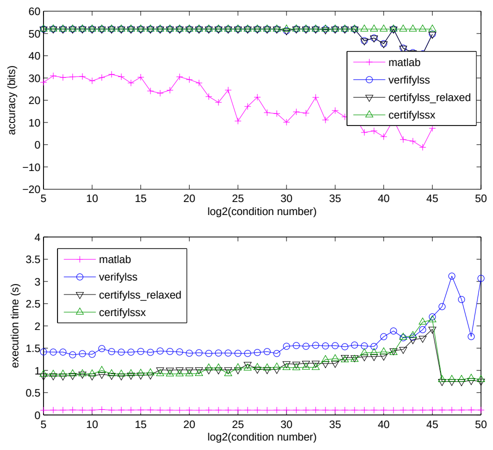


Figure 1: Performances of the `certifylssx` algorithm.

The algorithms presented in Sections 2 and 3 succeeded in providing an accurate solution $\tilde{x}$ of a linear system, with a guarantee on this accuracy being given by an enclosure $\mathbf{e}$ of the error. However, these algorithms fail when the condition number of the matrix becomes too large. This is due to the fact that the approximate inverse R of A becomes poor, and thus the interval matrix $\mathbf{K}$ becomes wide. A solution would thus be to compute a more accurate R, using extended precision, as proposed in [10, 7]. Preliminary experiments tend to show that the increase in execution time is drastic. To keep it reasonable, again a solution seems to trade off execution time against accuracy in well-chosen parts of the code.

Desirable properties for such an algorithm were mentioned in Section 1: accuracy, complexity, execution time. Accuracy and execution time have been shown on figures and results are thoroughly proven in [6]. Let us mention the complexity of algorithms for solving linear systems:

- the numerical, non-verified algorithm (based on LU factorization with partial pivoting) has a complexity of $\frac{4}{3}n^3 + \mathcal{O}(n^2)$ flops (*flops* stands for *f*loating-point *o*perations *p*er *s*econd);
- the numerical, verified algorithm of Section 3 has a complexity of $8n^3 + 2n^2 \log_2(p/(p - \log_2 \kappa(A))) + 8n^2 + \mathcal{O}(n)$ flops, where p is the precision (53 for the IEEE-754 double precision) and $\kappa(A)$ is the condition number of A. For the algorithm to terminate, one needs $\log_2 \kappa(A) < p$, in our experiments we need $\log_2 \kappa(A) < 45$. Because the refinement iterations converge quadratically, as well as the iterations of $\mathbf{e}$ (Krawczyk iterations are known to converge quadratically and Gauss-Seidel iterations are even faster [4]), one can show that the complexity of the iterative refinement part is $2n^2 \log_2(p/(p - \log_2 \kappa(A)))$. These complexity results are new.
- the complexity of solving exactly the linear system [2], where the coefficients of A and b are considered as rational numbers, is $\mathcal{O}^{\tilde{}}(pn^3)$ where again p is the precision, and $\mathcal{O}^{\tilde{}}$ means that (poly-)logarithmic terms are not explicited.

In other words, this means that exact methods seem superior to get exact (and thus verified) results. However, only methods based on interval arithmetic can extend to data given with uncertainties, i.e. with coefficients which are (tight) intervals. Indeed, the algorithms given here apply, as long as $\mathbf{K}$ remains an H-matrix.

5. REFERENCES

[1] J. Demmel, Y. Hida, W. Kahan, X. S. Li, S. Mukherjee, and E. J. Riedy. Error bounds from extra-precise iterative refinements. *ACM Trans. on Mathematical Software*, 32(2):325–351, 2006.

[2] J. D. Dixon. Exact solution of linear equations using *p*-adic expansions. *Numerische Mathematik*, 40:137–141, 1982.

[3] N. J. Higham. *Accuracy and Stability of Numerical Algorithms (2nd ed.)*. SIAM, 2002.

[4] A. Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, 1990.

[5] A. Neumaier. A simple derivation of the Hansen-Bliek-Rohn-Ning-Kearfott enclosure for linear interval equations. *Reliable Computing*, 5:131–136, 1999 (+ Erratum: Reliable Computing 6:227, 2000).

[6] H. D. Nguyen. *Efficient algorithms for verified scientific computing: numerical linear algebra using interval arithmetic*. PhD Thesis, École Normale Supérieure de Lyon, Jan. 2011. `http://perso.ens-lyon.fr/hong.diep.nguyen/`

[7] K. Ozaki, T. Ogita, and S. Oishi. An algorithm for automatically selecting a suitable verification method for linear systems. *Numerical Algorithms*, 56(3):363–382, 2011.

[8] J. Rohn. Enclosing solutions of linear interval equations is NP-hard. *Computing*, 53:365–368, 1994.

[9] S. M. Rump. INTLAB - INTerval LABoratory. In *Developments in Reliable Computing (Tibor Csendes ed.)*, pages 77–104. Kluwer Academic Publishers, 1999.

[10] S. M. Rump. Inversion of extremely ill-conditioned matrices in floating-point. *Japan Journal of Industrial and Applied Mathematics*, 26:249–277, 2009.

An Effective Implementation of a Symbolic-Numeric Cylindrical Algebraic Decomposition for Optimization Problems

Hidenao Iwane
Fujitsu Laboratories Ltd
4-1-1 Kamikodanaka,
Nakahara-ku, Kawasaki
211-8588, Japan
iwane@jp.fujitsu.com

Hitoshi Yanami
Fujitsu Laboratories Ltd
4-1-1 Kamikodanaka,
Nakahara-ku, Kawasaki
211-8588, Japan
yanami@labs.fujitsu.com

Hirokazu Anai
Fujitsu Laboratories Ltd /
Kyushu University
4-1-1 Kamikodanaka,
Nakahara-ku, Kawasaki
211-8588, Japan
anai@jp.fujitsu.com

ABSTRACT

With many applications in engineering and in scientific fields, quantifier elimination (QE) has been attracting more attention these days. Cylindrical algebraic decomposition (CAD) is used as a basis for a general QE algorithm. We propose an effective symbolic-numeric cylindrical algebraic decomposition (SNCAD) algorithm for solving polynomial optimization problems. The main ideas are a *bounded CAD construction* approach and *utilization of sign information*. The bounded CAD constructs CAD only in restricted admissible regions to remove redundant projection factors and avoid lifting cells where truth values are constant over the region. By utilization of sign information we can avoid symbolic computation in the lifting phase. Techniques for implementation are also presented. These techniques help reduce the computing time. We have examined our implementation by solving many example problems. Experimental results show that our implementation significantly improves efficiency compared to our previous work.

Categories and Subject Descriptors

G.4 [**Mathematics of computing**]: Mathematical Software; I.1 [**Symbolic and Algebraic Manipulation**]: Algorithms

General Terms

Algorithm, Experimentation

Keywords

cylindrical algebraic decomposition, quantifier elimination, symbolic-numeric computation, global optimization, certified numerical computation

1. INTRODUCTION

Cylindrical algebraic decomposition (CAD) is a general-purpose symbolic method aiming for quantifier elimination, which is an effective tool for solving real algebraic constraints (in particular parametric and non-convex case) arising in many engineering and industrial problems. However, QE based on CAD is not considered to be practical on computers, since CAD usually consists of many purely symbolic computations and has a bad computational complexity.

To circumvent the inherent computational complexity of a QE algorithm based on CAD, several researchers have focused on QE algorithms specialized to particular types of input formulas; see [44, 22, 29, 45, 19]. This direction is quite promising in practice since a number of important problems in engineering have been successfully reduced to such particular input formulas and resolved by using the specialized QE algorithms. See concrete successful applications in [46, 43, 13, 18, 1, 3].

However, there still remain many significant problems in engineering that cannot be recast as such particular formulas. Therefore, it is strongly desired to develop an efficient algorithm for CAD. A CAD algorithm consists of three phases; the projection phase, the base phase, and the lifting phase. The main improvement of CAD has been focused on improving the projection operator in the projection phase, e.g., [8, 21, 31], because it is the most effective way to reduce the computational time. Computational difficulties in the lifting phase stem from symbolic computation over towers of algebraic extensions and combinatorial explosion in CAD construction. An effective way for efficient CAD construction is to utilize numerical computation, instead of symbolic treatment, with derived numerical information on algebraic numbers as far as possible without violating correctness of the results. So far there have been some attempts to introduce numerical computation into CAD construction, for example [23, 39, 10, 36, 4, 41, 25]. Additionally many improvements of a CAD algorithm tailored for QE problems have been proposed. Partial CAD [9] avoid lifting cells by evaluation of truth values and utilization of quantifier information of input formulas. It significantly reduces the computing time in the lifting phase. Also, many approaches for special types of input formulas have been proposed, for example [30, 32, 33, 40, 5].

Meanwhile, several attempts to develop symbolic-numeric algorithms for solving polynomial optimization problems (POPs) have been done. In [2] an efficient algorithm based on CAD to solve semidefinite programming exactly is studied.

Another promising direction is to use generalized critical values for global optimization, which originates with [38] in the scheme of sum of squares. Then within the framework of the critical point method, generalized critical values are first used in [16].

As for sums of squares approach, we note that it provides algebraic certificates to compute lower bounds on global infimum (see [35]). In [26] computation of "symbolic" algebraic certificates (instead of "numerical" ones) is proposed. Moreover, the problem of obtaining theorems on the existence of algebraic certificates based on sums of squares is due to [20].

In this paper, we introduce an efficient approach for solving POPs by using SNCAD. Our approach is based on the SNCAD scheme, which employs validated numeric using interval arithmetic, proposed in [25] and we further exploit the following structures of POPs:

1. Variables sometimes have interval constraints.
2. To solve a POP by a QE algorithm the associated QE problem has equational constraints.

We solved optimization problems derived from manufacturing design by using QE. In this case the property 1 is often held, because each decision variable means a length or a position. By using the property 1 we can avoid the CAD construction over outside of the admissible regions of the variables. The property 2 enables us to skip lifting procedures by utilizing sign information.

The rest of this paper is organized as follows. The scheme for a standard CAD algorithm and QE based on CAD are reviewed in Section 2. The explanation of POPs and our symbolic approach to POPs are shown in Section 3. We describe our algorithm in Section 4 and Section 5. Experimental results are given in Section 6. Discussion and some concluding remarks are made in Section 7.

2. STANDARD CAD ALGORITHMS

We briefly sketch the basic ideas of cylindrical algebraic decomposition (CAD); see [8] for details. Here we denote the rational number field and the real number field by $\mathbb{Q}$ and $\mathbb{R}$, respectively. Assume that we are given a prenex formula φ with q free variables $x_1, \ldots, x_q$ and $(r-q)$ quantified variables $x_{q+1}, \ldots, x_r$:

$$\varphi(x_1, \ldots, x_q) \equiv \mathsf{Q}_{q+1}x_{q+1} \ldots \mathsf{Q}_r x_r \ \psi(x_1, \ldots, x_r),$$

where $\mathsf{Q}_j \in \{\exists, \forall\}$ and ψ is a quantifier-free formula. We can assume, by transposing terms if necessary, each atomic formula in ψ is represented in the form $f \,\rho\, 0$, where f is a polynomial with rational coefficients on $x_1, \ldots, x_r$ and $\rho \in \{\leq, <, =, \neq\}$. Let $F \subseteq \mathbb{Q}[x_1, \ldots, x_r]$ be the set of polynomials appearing in ψ as the left hand sides of atomic formulas. A subset $C \subseteq \mathbb{R}^r$ is said to be *sign-invariant* for F if every polynomial in F has a constant sign on all points in C. Then $\psi(S)$ is either "true" or "false" for all $S \in C$.

Suppose we have a finite sequence $\mathcal{D}_1, \ldots, \mathcal{D}_r$ for F which has the following properties:

1. Each $\mathcal{D}_i$ is a partition of $\mathbb{R}^i$ into finitely many connected semi-algebraic sets called *cells*.
2. $\mathcal{D}_{i-1}$, $1 < i \leq r$, consists exactly of the projections of all cells in $\mathcal{D}_i$ along the coordinate of the i-th variable in $(x_1, \ldots, x_r)$. For each cell $C \in \mathcal{D}_{i-1}$ we can determine its preimage $P(C) \subseteq \mathcal{D}_i$ under the projection.
3. Each cell $C \in \mathcal{D}_r$ is sign-invariant for F. Moreover for each cell $C \in \mathcal{D}_r$ we are given a *sample point* $S \in C$ in such a form that we can determine the sign of $f(S)$ for each $f \in F$ and thus evaluate $\varphi(S)$.

Then the partition $\mathcal{D}_r$ of $\mathbb{R}^r$ for F is called an *F-invariant cylindrical algebraic decomposition* of $\mathbb{R}^r$. A CAD algorithm computes such a sequence $\mathcal{D}_1, \ldots, \mathcal{D}_r$ and it consists of three phases; the *projection phase*, the *base phase*, and the *lifting phase*.

Projection phase: We first construct from $F \subseteq \mathbb{Q}[x_1,\ldots, x_r]$ a new finite set $F' \subseteq \mathbb{Q}[x_1,\ldots,x_{r-1}]$ that satisfies a special condition called "delineability", where the order of the real roots of all polynomials in F as univariate polynomials in x_r does not change above each connected cell in $\mathcal{D}_{r-1}$.

The step constructing F' from F is called a *projection* and denoted by $F' := \textsc{Proj}(F, x_r)$. We call polynomials in F' *projection polynomials* and their irreducible factors *projection factors*. Iterative application of the $\textsc{Proj}$ operator leads to a finite sequence

$$F_r, \ldots, F_1, \quad \text{where} \quad F_r := F,\ F_i := \textsc{Proj}(F_{i+1}, x_{i+1})$$

for $1 \leq i < r$. The $\textsc{Proj}$ operator, in general, computes certain coefficients, discriminants, and resultants derived from polynomials in F_{i+1} and their higher derivatives by regarding those as univariate polynomials in x_{i+1}. The final set F_1 consists of univariate polynomials in x_1.

Base phase: Then we construct a partition $\mathcal{D}_1$ of the real line $\mathbb{R}^1$ into finitely many intervals that are sign-invariant for F_1. This step is called the *base phase*, achieved by isolating the real zeros of the univariate polynomials in F_1.

Lifting phase: The partitions $\mathcal{D}_i$ of $\mathbb{R}^i$ for $2 \leq i \leq r$ are computed recursively: The roots of all polynomials in F_i as univariate polynomials in x_i are delineated above each connected cell $C \in \mathcal{D}_{i-1}$. Thus we can cut the *cylinder* above C into finitely many connected semi-algebraic sets (cells). This is done by real root isolation of the univariate polynomials derived through specializing the polynomials in F_i by a sample point of C. Then $\mathcal{D}_i$ is a collection of all such cells obtained from all cylinders above the cells of $\mathcal{D}_{i-1}$.

A finite sequence $\mathcal{D}_1, \ldots, \mathcal{D}_r$ for F has a tree structure: The first level of nodes under the root of the tree corresponds to the cells in $\mathcal{D}_1$. The second level of nodes stands for the cells in $\mathcal{D}_2$, i.e., the cylinders over the cells of $\mathbb{R}^1$. The leaves represent the cells of $\mathcal{D}_r$, i.e., a CAD of $\mathbb{R}^r$. A sample point of each cell is stored in its corresponding node or leaf. At each level of the tree there are a number of projection polynomials F_i whose signs define a cell when evaluated over a sample point.

3. OPTIMIZATION PROBLEMS

An optimization problem is the problem of finding the minimum value or maximum value of a function under some constraints. Since the maximum value of a function is equivalent to the minimum value of the negative of the function, we can focus on minimization without loss of generality. Let $\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ be a vector of variables, $\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x}))$ a vector of $\mathbb{R}$-valued functions, and $g_j(\boldsymbol{x})$ a $\mathbb{R}$-valued function for $j = 1, \dots, k$. An optimization problem is formulated as follows:

$$\begin{array}{ll} \text{Minimize} & \boldsymbol{f}(\boldsymbol{x}) \\ \text{subject to} & g_1(\boldsymbol{x}) \le 0, \dots, g_k(\boldsymbol{x}) \le 0. \end{array}$$

This is called a *multi-objective optimization problem* when it has multiple objective functions. The variables x_i are called *decision variables*, and the functions f_i are called *objective functions*. The following set is called a *feasible region*:

$$\psi_c(\boldsymbol{x}) = \{\boldsymbol{x} \in \mathbb{R}^n \mid g_i(\boldsymbol{x}) \le 0 \text{ for } i = 1, \dots, k\}.$$

The image of the feasible region is called a *feasible objective region.*

Next we define an order in the objective space to state what minimize means in the problem above.

Let $\boldsymbol{a} = (a_1, \dots, a_m)$ and $\boldsymbol{b} = (b_1, \dots, b_m)$ be points in $\mathbb{R}^m$. We denote $\boldsymbol{a} \preceq \boldsymbol{b}$ to represent $a_i \le b_i$ for $i = 1, \dots, m$. Since $\mathbb{R}^m$ is a *partially ordered set* with respect to $\preceq$. In general there does not exist an optimal solution in a multi-objective optimization problem, because there is a trade-off between objective functions.

In a multi-objective optimization problem, we need to consider a solution as a set instead of a single point.

Definition 1. Let $\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \dots, f_m(\boldsymbol{x}))$ be a vector of objective functions, $\psi_c \subseteq \mathbb{R}^n$ a semialgebraic set, and consider the optimization problem above. $\boldsymbol{x}_0 \in \psi_c$ is called a *Pareto optimal solution* if there does not exists $\boldsymbol{x} \in \psi_c$ such that $\boldsymbol{f}(\boldsymbol{x}) \ne \boldsymbol{f}(\boldsymbol{x_0})$ and $\boldsymbol{f}(\boldsymbol{x}) \preceq \boldsymbol{f}(\boldsymbol{x}_0)$. Such points form a set, called a *Pareto optimal set.* And the image of the Pareto optimal set is called the *Pareto optimal front*, which is a subset of the feasible objective region.

In this paper we consider special optimization problems whose objective functions and constraints are all described by multivariate polynomials. We call such a problem a *polynomial optimization problem* (POP).

3.1 Symbolic Approach to POP

Here we give our symbolic approach to a polynomial optimization problem. We can construct a first-order formula with free variables $\boldsymbol{y} = (y_1, \dots, y_m)$ that is true at $\boldsymbol{y} = \boldsymbol{y}_0$ if and only if $\boldsymbol{y}_0$ is in the feasible objective region:

$$\exists \boldsymbol{x} \ (\boldsymbol{y} \equiv \boldsymbol{f}(\boldsymbol{x}) \wedge \psi_c(\boldsymbol{x})), \tag{1}$$

where we denote $(y_1 = f_1(\boldsymbol{x}) \wedge \cdots \wedge y_m = f_m(\boldsymbol{x}))$ by $\boldsymbol{y} \equiv \boldsymbol{f}(\boldsymbol{x})$. By eliminating $\boldsymbol{x}$ from (1) we obtain a quantifier-free formula ψ_{FO} with respect to $\boldsymbol{y}$.

Next we can express the Pareto optimal front by the following first-order formula with free variables $\boldsymbol{y}_p \in \mathbb{R}^m$:

$$\psi_{FO}(\boldsymbol{y}_p) \wedge \neg\exists \boldsymbol{y} \ (\psi_{FO}(\boldsymbol{y}) \wedge \boldsymbol{y} \ne \boldsymbol{y}_p \wedge \boldsymbol{y} \preceq \boldsymbol{y}_p). \tag{2}$$

Performing QE on (2), we obtain an equivalent quantifier-free formula ψ_P with respect to $\boldsymbol{y}_p$.

Finally we consider the following first-order formula:

$$\exists \boldsymbol{y} \ (\boldsymbol{y} \equiv \boldsymbol{f}(\boldsymbol{x}) \wedge \psi_c(\boldsymbol{x}) \wedge \psi_p(\boldsymbol{y})). \tag{3}$$

Performing QE on (3), we obtain an equivalent quantifier-free formula that is true at $\boldsymbol{x} = \boldsymbol{x}_{opt}$ if and only if $\boldsymbol{x}_{opt}$ in the Pareto optimal set.

The feasible objective region ψ_{FO} includes the information on the Pareto optimal front and is easier to compute than ψ_P. By using the symbolic approach, we obtain an exact solution of a POP even if it is non-convex. But from the computational point of view QE is indeed very hard. Then we propose an effective approach of a CAD algorithm for POPs.

4. BOUNDED CAD

In this section we show an effective approach for CAD constructed in a given restricted region. We will call it *bounded CAD* construction. The bounded CAD construction uses a smaller set of projection factors and skips lifting more cells. It significantly reduces the computing time.

We are going to demonstrate by means of a small example how we find unnecessary projection factors for bounded CAD construction. We consider the following QE problem:

$$\varphi = \exists y \ (x^2 + y^2 < 1 \wedge x \ge 0),$$

where x is bounded. We obtain the following projection factors by a conventional projection operator:

$$F_1 = \{x + 1, x, x - 1\}.$$

Since the formula φ is obviously false in $x < 0$, we do not need to construct CAD in $x < 0$. Thus, the projection factor $x + 1$ is unnecessary to solve the QE problem φ.

Here we propose an effective approach for a CAD algorithm which constructs CAD only in the regions where truth values of an input QE formula are not trivial.

4.1 A New Projection for Bounded CAD

Since a projection operator that produces small sets makes CAD computation efficient, there have been proposed many projection operators, e.g., [8, 21, 31]. In addition some special projection operators which depend on input formulas have been proposed, for example the restricted equational projection operator [32]. In this subsection we propose an effective projection operator for bounded CAD construction.

The output of a CAD algorithm is a partition of a variable space, where all input polynomials are sign-invariant within each cell. Therefore the following lemma holds.

LEMMA 1. *Let $F \subseteq \mathbb{Q}[x_1, \dots, x_r]$ be a finite set of polynomials, $g(x_1, \dots, x_r)$ a polynomial which does not vanish for all $(x_1, \dots, x_r) \in \mathbb{R}^r$, $\mathcal{D}_r$ an output of a CAD algorithm for F. Then each cell $C \in \mathcal{D}_r$ is sign-invariant in $F \cup g$.*

Lemma 1 states that a sign-definite polynomial is not needed for CAD construction. The following lemma shows that we might reduce projection factors for bounded CAD construction.

LEMMA 2. *Let $F \subseteq \mathbb{Q}[x_1, \dots, x_r]$ be a finite set of polynomials, $\mathcal{U}_r \subseteq \mathbb{R}^r$ a region, $g(x_1, \dots, x_r)$ a polynomial which does not vanish for all $(x_1, \dots, x_r) \in \mathcal{U}_r$, $\mathcal{D}_r$ an output of a CAD algorithm for F.*

If $C \in \mathcal{D}_r$ is a subset of $\mathcal{U}_r$, then C is sign-invariant in $F \cup g$.

By using lemma 2, we propose a new projection operator that refines any other projection operator PROJ by removing unnecessary projection factors in a restricted area before applying PROJ.

```
BCADPROJECTION($F_k$, $\mathcal{U}_k$, $x_k$)
Input:   polynomials $F_k \subseteq \mathbb{Q}[x_1, \dots, x_k]$  $(k > 1)$,
         region for bounded CAD construction $\mathcal{U}_k \subseteq \mathbb{R}^k$,
         main variable $x_k$
Output:  projection factors $F_{k-1} \subseteq \mathbb{Q}[x_1, \dots, x_{k-1}]$

    $G \leftarrow \{\}$
    for $f(x_1, \dots, x_k) \in F_k$ do
(*)     if $\exists x_1 \cdots \exists x_k((x_1, \dots, x_k) \in \mathcal{U}_k \wedge f(x_1, \dots, x_k) = 0)$
            $G \leftarrow G \cup f(x_1, \dots, x_k)$
    return Proj($G$, $x_k$)
```

In this projection operator we have to solve a lot of sub-QE problems (*). However, if we can solve them efficiently, it might drastically decrease the computational time of CAD construction since we can utilize a smaller set of projection factors.

Remark 1. Since the input formula of a QE problem might have redundant conditions, we check whether input polynomials of BCADProjection are sign-definite in $\mathcal{U}_r$. Unnecessary projection factors of F_1 are naturally removed in the base phase.

Incidentally, the output of most of projection operators contains coefficients of input polynomials to hold degree-invariant, e.g., [31]. Thus, we might reduce the number of projection factors for bounded CAD by the following algorithm.

```
CoefPartOfProjection($f$, $\mathcal{U}_{k-1}$, $x_k$)
Input:   polynomial $f \in \mathbb{Q}[x_1, \dots, x_k]$  $(k > 1)$,
         region $\mathcal{U}_{k-1} \subseteq \mathbb{R}^{k-1}$,
         main variable $x_k$
Output:  part of projection factor $F_{k-1} \subseteq \mathbb{Q}[x_1, \dots, x_{k-1}]$

    $G \leftarrow \{\}$
    for $i$ from degree of $f$ in $x_k$ to 1 by -1 do
        $c \leftarrow$ coefficient of $x_k^i$ in $f$
        if $\exists x_1 \cdots \exists x_{k-1}((x_1, \dots, x_{k-1}) \in \mathcal{U}_{k-1} \wedge$
                            $c(x_1, \dots, x_{k-1}) = 0)$
            $G \leftarrow G \cup c$
        else break
    return $G$
```

4.2 Truth Value Evaluation

Partial CAD, presented by G. E. Collins and H. Hong [9], is an efficient approach for the lifting phase. Trial evaluation is one of the ways of avoiding lifting cells in partial CAD construction.

The following simple example illustrates the trial evaluation. Let $\varphi(x_1, x_2) = \exists x_2(\psi_1(x_1) \wedge \psi_2(x_1, x_2))$ be a formula where ψ_1 and ψ_2 are quantifier-free. Now, we consider to lift a cell $C_1 \in \mathcal{D}_1$. We know the signs of the projection factors F_1, because C_1 is sign-invariant for F_1. Therefore we can evaluate the formula ψ_1. If ψ_1 is false in the cell C_1, then the formula φ is false for all x_2. Thus we do not need to lift C_1.

Here we explain a trial evaluation for bounded CAD construction. Let $\varphi(x_1, \dots, x_r)$ be an input formula of a QE problem, and $\mathcal{D}_i$ a partition of $\mathbb{R}^i$ for $i = 1, \dots, r$.

The trial evaluation in the partial CAD utilizes the following lemma from [9].

Lemma 3. *Let C_k be a cell in $\mathcal{D}_k$ of $\mathbb{R}^k$, $1 \leq k < r$, and let $S = (s_1, \dots, s_k)$ be a sample point for C_k. If $\varphi(s_1, \dots, s_k, x_{k+1}, \dots, x_r)$ has a constant truth value then we do not need to lift C_k.*

Let $f(x_1, \dots, x_r) \in \mathbb{Q}[x_1, \dots, x_r]$ be a non-constant r-variable polynomial. We define $n(f)$ as the largest integer k, such that $\deg_k(f) > 0$, where $\deg_k(f)$ is the degree of f with respect to x_k. The trial evaluation deals with only the polynomial f such that $n(f) \leq k$. Since we construct CAD only in $\mathcal{D}_r$, the following lemma is immediate.

Lemma 4. *Let C_k be a cell in $\mathcal{D}_k$, $1 \leq k < r$, $\mathcal{U}_r$ a region, and let $S = (s_1, \dots, s_k)$ be a sample point for C_k. If $\varphi(s_1, \dots, s_k, x_{k+1}, \dots, x_r)$ has a constant truth value in $\mathcal{U}_r$, then we do not need to lift C_k for bounded CAD construction in $\mathcal{U}_r$.*

In our approach, we try to evaluate all the polynomials occurring in input formulas. Hence we can avoid lifting more cells, and the computation time might decrease.

4.3 Procedures

In this subsection we explain three procedures for bounded CAD construction.

(A) Region $\mathcal{U}_r$: As a preprocessing of bounded CAD, we have to compute a region $\mathcal{U}_r$ where truth value of an input formula φ is not trivial. We denote a quantifier-free part of φ by ψ. We will call a region where truth value of a formula is true and false a *true region* and a *false region*, respectively.

In general computing a true region and false region exactly is difficult, but it is often the case that the truth value of a certain subset is easily decided from an atomic formula or two appearing in φ. We refer to such points that are directly detected true as a *white region*. Similarly we use the term a *black region* for a 'trivially' false subset. Note that detecting white/black regions are important because we possibly remove some of projection factors and avoid lifting cells in those regions, which reduces total computing time.

First we compute a white region and a black region by CompTF($\psi \wedge (0 = 0)$, $(-\infty, \infty)$, $\emptyset$). Its outputs are $\mathcal{T}_r$ and $\mathcal{F}_r$ expressed by Cartesian products of unions of open intervals.

```
CompTF($\psi$, $I_T$, $I_F$)
Input:   quantifier-free formula $\psi(x_1, \dots, x_r)$,
         regions $I_T$, $I_F \subseteq \mathbb{R}$
Output:  white region and black region

    if $\psi$ is atomic formula then
        return CompAtom($\psi$, $I_T$, $I_F$)
    $\psi \leftarrow$ equivalent conjunction & disjunction formula to $\psi$
    $r \leftarrow$ false
    if $\psi$ is conjunction then
        $T_0 \leftarrow (-\infty, +\infty)$
        $F_0 \leftarrow \emptyset$
        for $p$ in element of conjunction do
            $T$, $F \leftarrow$ CompTF($p$, $(-\infty, \infty)$, $\emptyset$)
            if error then continue
            $T_0 \leftarrow T_0 \cap T$
            $F_0 \leftarrow F_0 \cup F$
            $r \leftarrow$ true
    else
        $T_0 \leftarrow \emptyset$
        $F_0 \leftarrow (-\infty, +\infty)$
        for $p$ in element of disjunction do
            $T$, $F \leftarrow$ CompTF($p$, $\emptyset$, $(-\infty, \infty)$)
            if error then continue
            $T_0 \leftarrow T_0 \cup T$
            $F_0 \leftarrow F_0 \cap F$
        $r \leftarrow$ true
    if $r =$ false then
        return error
    return $T_0$, $F_0$
```

The sub-procedure CompAtom outputs a white region and a black region of a given atomic formula.

```
CompAtom($\psi$, $I_T$, $I_F$)
Input:   atomic formula $\psi = f(x_1, \dots, x_r)\rho\ 0$,
             where $\rho \in \{<, \leq, =, \neq\}$,
         regions $I_T$, $I_F \subseteq \mathbb{R}$
Output:  white region and black region

    $T_j \leftarrow I_T$ for $j = 1, \dots, r$
    $F_j \leftarrow I_F$ for $j = 1, \dots, r$
    if $f$ is univariate polynomial in $x_i$ then
        $L_i \leftarrow$ a union of isolating intervals of $f$
        $J_i \leftarrow \mathbb{R} \setminus L_i$ (union of open intervals)
        $T_i \leftarrow \{s_i \in J_i \mid f(s_i)\ \rho\ 0\}$
        $F_i \leftarrow \{s_i \in J_i \mid s_i \notin T_i\}$
    else
        for $i$ from 1 to $r$ do
            if all coefficients of $x_i$ are rational number
                    except constant term then
                $F \leftarrow$ substitute $(-\infty, \infty)$ for $x_j$ $(j \neq i)$
                transpose $F\ \rho\ 0$ to $g(x_i)\ \rho\ [I^{(l)}, I^{(u)}]$
                update $T_i$, $F_i$ from $g(x_i)\ \rho\ [I^{(l)}, I^{(u)}]$
                    by same manner as univariate polynomial
        if $T_i$, $F_i$ do not update then
            return error
    return $T_1 \times \cdots \times T_r$, $F_1 \times \cdots \times F_r$
```

Let $\mathcal{T}_r, \mathcal{F}_r \subseteq \mathbb{R}^r$ be a white region and a black region of an input formula φ. By using $\mathcal{T}_r$ and $\mathcal{F}_r$, we can easily obtain $\mathcal{U}_r = \mathbb{R}^r \setminus (\mathcal{T}_r \cup \mathcal{F}_r)$.

In our implementation $\mathcal{U}_r$ is expressed by a Cartesian product of closed intervals, that is $\mathcal{U}_r = [I_1^{(l)}, I_1^{(u)}] \times \cdots \times [I_r^{(l)}, I_r^{(u)}]$. It is the reason why it makes easy to achieve evaluations in the following procedures (B) and (C).

(B) Evaluation of sub-QE problem: Since we express $\mathcal{U}_r$ by a Cartesian product of intervals as (A), it is easy to evaluate the sub-QE problem (*) in the algorithm BCAD-Projection by using numerical computation. We just substitute each interval $[I_i^{(l)}, I_i^{(u)}]$ for x_i, and compute a range of values by using interval arithmetic. If the range contains zero, we deal with the QE problem (*) as true.

For necessary projection factor this approach returns always true. Therefore we do not lose the correctness as a projection operator.

(C) Evaluation of truth value in a region $\mathcal{U}_r$: Here we use the same notation as lemma 4. In a similar way as (B), to evaluate the formula φ we substitute isolating intervals of s_i for x_i $(i = 1, \dots, k)$ and intervals $[I_j^{(l)}, I_j^{(u)}]$ for x_j $(j = k+1, \dots, r)$ in each polynomial appearing in φ, and compute a range of values by using interval arithmetic without losing correctness. If φ has a constant truth value in C_k over $\mathcal{U}_r$, then we avoid lifting C_k.

Note that we already obtain the sign information of the polynomial f such that $n(f) \leq k$, and if s_k is not contained the interval $[I_k^{(l)}, I_k^{(u)}]$ then a sample point S belongs to $\mathcal{T}_k$ or $\mathcal{F}_k$.

5. UTILIZATION OF COMPUTATIONAL RESULTS

In this section we show how to utilize computational results in the lifting phase. Using those information enables us to avoid symbolic computation. The approaches appearing in this section except Section 5.3.1 are a general approach for symbolic-numeric CAD (SNCAD).

The approaches depend on a projection operator. Here we explain our approach by using McCallum's projection operator ProjMC from [31].

```
ProjMC($F_{k+1}$, $x_{k+1}$)
Input:   polynomials $F_{k+1} \subseteq \mathbb{Q}[x_1, \dots, x_{k+1}]$ $(k \geq 1)$,
         main variable $x_{k+1}$
Output:  projection factors $F_k \subseteq \mathbb{Q}[x_1, \dots, x_k]$

    $G \leftarrow \{\}$
    for $f \in F_{k+1}$ do
        $G \leftarrow G \cup$ Coeffs$(f, x_{k+1})$
        $G \leftarrow G \cup$ Discriminant$(f, x_{k+1})$
    for $f, g \in F_{k+1}$ do    $(f \neq g)$
        $G \leftarrow G \cup$ Resultant$(f, g, x_{k+1})$
    return irreducible factors of $G$
```

The sub-procedures Coeffs(f, x), Discriminant(f, x), and Resultant(f, g, x) return the set of all the coefficients of f w.r.t. x, the discriminant of f w.r.t. x, and the resultant of f and g w.r.t. x, respectively.

Let $f_{k+1}(x_1, \dots, x_k, x_{k+1}) \in F_{k+1}$ be a projection factor. We think of $f_{k+1} = \sum_{i=0}^{d} c_i(x_1, \dots, x_k)x_{k+1}^i$ as an element of $\mathbb{Q}(x_1, \dots, x_k)[x_{k+1}]$. A sample point $S = (s_1, \dots, s_k)$ in a cell $C_k \subseteq R^k$ is a k-tuple of algebraic numbers. Each component of S is inductively defined as follows: When $k = 1$, $s_1 \in \mathbb{R}$ is expressed by a univariate polynomial $f_1 \in \mathbb{Q}[x_1]$ and an isolating interval I_1, where s_1 is a unique real root of f_1 in I_1. Then, for $k \geq 2$, s_k is also expressed by a pair of a polynomial $f_k \in \mathbb{Q}[x_1, \dots, x_k]$ and an isolating interval I_k, where s_k is a unique real root of $f_k(s_1, \dots, s_{k-1}, x)$ in I_k. From now on we identify $s_i \in \mathbb{R}$ with a pair (f_i, I_i) and write s_i for (f_i, I_i) simply.

Since C_k is sign-invariant for F_k, we obtain the sign information of F_k at S. Now F_k is a set of projection factors derived from F_{k+1} by ProjMC. Hence we know the signs of the coefficients of f, the discriminant of f, and the resultant of f and g at S, for all $f, g \in F_{k+1}$. By using sign information of projection factors we can avoid a lot of symbolic computation in the lifting phase.

Remark 2. When we use ProjMC as a projection operator of BCADProjection, the sign of a removed projection factor is defined as its sign over a region $\mathcal{U}_k$.

5.1 Sign Information of Coefficients

In our implementation of SNCAD, presented in [25], to find the real roots in $f(x_1, \dots, x_k, x_{k+1})$ at $S = (s_1, \dots, s_k)$, instead of substituting (symbolically) algebraic number s_i for x_i, $i = 1, \dots, k$, we substitute I_i, the isolating interval of s_i, for x_i in f. Since extension of intervals is generally-restrained, each coefficient $I = [I^{(l)}, I^{(u)}]$ of an input interval polynomial is sign-definite (i.e., $I^{(l)} \cdot I^{(u)} > 0$ or $I^{(l)} = I^{(u)} = 0$).

```
SubstSamplePointCoef($c$, $S$)
Input:   polynomial $c \in \mathbb{Q}[x_1, \dots, x_k]$,
         sample point $S = (s_1, \ldots, s_k) \in \mathbb{R}^k$
Output:  interval $I$ which substitute $s_j$ for $x_j$ in $c$
         where $I^{(l)} \cdot I^{(u)} > 0$ or $I^{(u)} = I^{(l)} = 0$

  $I \leftarrow$ substitute each isolating interval of $s_j$ for $x_j$ in $c$
  $(I \leftarrow c(S))$
  if $I^{(l)} \le 0$ and $I^{(u)} \ge 0$ then
      if SymbolicZeroChk$(c, S)$ then
          return [0,0]

  while $I^{(l)} \le 0$ and $I^{(u)} \ge 0$
      $S \leftarrow$ IncreasePrecision$(S)$
      $I \leftarrow c(S)$
  return $I$
```

In this algorithm we need to switch to the symbolic computation SymbolicZeroChk(c,S) to check whether c is exactly zero at S, when the interval I contains zero. If c is not equal to zero at S, we improve the precision of S while I contains zero.

Now the coefficients c of f are projection factors, we can avoid the symbolic computation by using sign information.

We note that some of the projection operators (e.g., [32, 6]) produce not all the coefficients of input polynomials.

5.2 Sign Information of Discriminant

By using sign information of discriminants we can skip the unnecessary symbolic square-free computation. Moreover, we can avoid more symbolic computation for the polynomials of degree two.

Since we use dynamic evaluation [14, 15] for representing algebraic extensions in our scheme, we can avoid heavy computation of algebraic factorization of defining polynomials and we only need square-free computation of defining polynomials.

The following algorithm presented in [25] isolates real roots of a polynomial at a sample point.

```
IntvRealRootIsol($f$, $S$)
Input :  polynomial $f \in \mathbb{Q}[x_1, \ldots, x_k, x_{k+1}]$,
         sample point $S = (s_1, \ldots, s_k) \in \mathbb{R}^k$
Output:  isolating intervals for $f(s_1, \ldots, s_k, x_{k+1})$,
         square-free part of $f$

  $r, f \leftarrow$ IntvRealRootIsolSqfr$(f, S)$
  if error then
      $f \leftarrow$ SymbolicSqfr$(f)$
      while
          $r, f \leftarrow$ IntvRealRootIsolSqfr$(f, S)$
          if not error then
              break
          $S \leftarrow$ IncreasePrecision$(S)$
  return $r$, $f$
```

The function IntvRealRootIsolSqfr based on Krawczyk's method [27] isolates real roots for a univariate polynomial $f(s_1, \ldots, s_k, x_{k+1})$. The Krawczyk method fails in isolating real roots for non-squarefree polynomials. From this property we call symbolic square-free SymbolicSqfr only when IntvRealRootIsolSqfr fails for non-squarefree case.

Still, we might execute the unnecessary square-free computation, if the precision of the isolating intervals of the sample point is not sufficient. However, we can avoid the symbolic square-free computation by using the sign of the discriminant of f at S. Thus when the discriminant of f is not equal to zero at S, we do not need to call symbolic square-free computation.

Now we know that the signs of the coefficients and the discriminant of $f \in \mathbb{Q}[x_1, \ldots, x_k, x_{k+1}]$ at $S = (s_1, \ldots, s_k) \in \mathbb{R}^k$. When the leading coefficient of $f(s_1, \ldots, s_k, x_{k+1})$ of degree two does not vanish and its discriminant is equal to zero, the following formula must hold:

$$c_2 x_{k+1}^2 + c_1 x_{k+1} + c_0 = w(2c_2 x_{k+1} + c_1)^2, \qquad (4)$$

where w is a constant value such that $c_2 \cdot w > 0$. Since we are interested only in the sign of f in a CAD computation, we replace the symbolic square-free computation with (4) where $|w| = 1$. We can obtain the sign of c_2 from the sign information of projection factors. Hence we can avoid the heavy symbolic computation in an algebraic extension field. This method has effect in the CAD computation, when there exist many projection factors of degree two.

From another point of view, when its degree is two and its discriminant is negative at S, $f(s_1, \ldots, s_k, x_{k+1})$ is sign definite. We can apply this fact in the truth value evaluation.

5.3 Sign Information of Resultant

Let $f, g \in F_{k+1}$ be projection factors, $C_k \in \mathcal{D}_k$ a cell, and $S = (s_1, \ldots, s_k) \in \mathbb{R}^k$ a sample point for C_k. We denote the real root isolating intervals of f and g at S by L_f and L_g, respectively. In our SNCAD, if $I_f \in L_f$ intersects $I_g \in L_g$ then we check whether two isolating intervals stand for the same algebraic number in a symbolic sense by using their defining polynomials. Here we can use the sign information of their resultant to avoid the symbolic computation.

If the resultant of f and g w.r.t. x_{k+1} is not equal to zero at S, they have no common root. Thus in this case we can isolate the intervals by improving the precision of isolating intervals of S.

In the case that the resultant of f and g is equal to zero when the following three conditions hold, I_f and I_g are express the same algebraic number:

1. The leading coefficients of f and g in x_{k+1} are not equal to zero at S.

2. Either f or g has only real roots at S.

3. The number of overlapping isolating intervals is exactly one.

Remark 3. By computing all the complex roots of projection factors, we might avoid more symbolic computations.

5.3.1 Multiple Equational Constraints

If an atomic formula is an equation it is called an *equational constraint*, and the polynomial in the equation is called an *equational constraint polynomial.* QE problems sometimes have equational constraints. In fact in our symbolic approach to POPs, the associated QE problems always have equational constraints for handling objective functions. Some projection operators have been proposed for equational constraints, e.g., [32]. Here we consider QE problems with multiple equational constraints. In other words we consider that QE problems become true only when multiple polynomial is zero simultaneously.

The following lemma is obvious:

LEMMA 5. *Let $f_1, f_2 \in F_{k+1}$ be equational constraint polynomials, $C_k \in \mathcal{D}_k$ a cell, $S \in \mathbb{R}^k$ a sample point for C_k. If the resultant of f_1 and f_2 is different from zero at S then truth value of C_k is false.*

Since we already obtain the sign information of their resultant, we can avoid lifting the cells by this condition without heavy symbolic computation.

6. COMPUTATIONAL RESULTS

In this section we show our experimental results on several benchmark problems including some optimization problems to demonstrate the effectiveness and efficiency of our approach.

The problems we deal with here are shown in the sequel. Four example problems in the first block of the tables are taken from [41] concerning with stability study and control design. The second block consists of typical control design problems which are very obstinate for semi-definite programming [34]. Two well-known examples from QE-related papers are in the third block. The examples except of the last block are shown in [25]. The last block except Table 3 adds QE problems for optimization problems. Here we considered QE problems which express feasible objective regions, i.e., solving (1).

portfolio [37]:

$$\begin{array}{ll} \text{Minimize} & 45x_3^2 - 30x_2x_3 + 10x_1x_3 + 3x_2^2 - 40x_1x_2 + 8x_1^2 \\ \text{subject to} & x_1 + x_2 + x_3 \le 10000, \\ & 5x_1 - 4x_2 + 15x_3 \ge 100000, \\ & x_1 \ge 0, x_2 \ge 0, x_3 \ge 0. \end{array}$$

The associated QE problem is given as:

$$\begin{array}{l} \exists x_1 \exists x_2 \exists x_3 \quad (x_1 \ge 0 \ \wedge \ x_2 \ge 0 \ \wedge \ x_3 \ge 0 \ \wedge \\ \quad x_1 + x_2 + x_3 \le 10000 \ \wedge \ 5x_1 - 4x_2 + 15x_3 \ge 100000 \ \wedge \\ \quad y = 45x_3^2 - 30x_2x_3 + 10x_1x_3 + 3x_2^2 - 40x_1x_2 + 8x_1^2). \end{array}$$

port-nox3: portfolio problem removed redundant constraint [37]:

$$\begin{array}{ll} \text{Minimize} & 45x_3^2 - 30x_2x_3 + 10x_1x_3 + 3x_2^2 - 40x_1x_2 + 8x_1^2 \\ \text{subject to} & x_1 + x_2 + x_3 \le 10000, \\ & 5x_1 - 4x_2 + 15x_3 \ge 100000, \\ & x_1 \ge 0, x_2 \ge 0. \end{array}$$

The associated QE problem is given as:

$$\begin{array}{l} \exists x_1 \exists x_2 \exists x_3 \quad (x_1 \ge 0 \ \wedge \ x_2 \ge 0 \ \wedge \\ \quad x_1 + x_2 + x_3 \le 10000 \ \wedge \ 5x_1 - 4x_2 + 15x_3 \ge 100000 \ \wedge \\ \quad y = 45x_3^2 - 30x_2x_3 + 10x_1x_3 + 3x_2^2 - 40x_1x_2 + 8x_1^2). \end{array}$$

port-para: parametric optimization problem of [37]:

$$\begin{array}{ll} \text{Minimize} & 45x_3^2 - 30x_2x_3 + 10x_1x_3 + 3x_2^2 - 40x_1x_2 + 8x_1^2 \\ \text{subject to} & x_1 + x_2 + x_3 = t, \\ & 5x_1 - 4x_2 + 15x_3 \ge 100000, \\ & x_1 \ge 0, x_2 \ge 0, x_3 \ge 0. \end{array}$$

The associated QE problem is given as:

$$\begin{array}{l} \exists x_1 \exists x_2 \exists x_3 \quad (x_1 \ge 0 \ \wedge \ x_2 \ge 0 \ \wedge \ x_3 \ge 0 \ \wedge \\ \qquad x_1 + x_2 + x_3 = t \ \wedge \ 5x_1 - 4x_2 + 15x_3 \ge 100000 \ \wedge \\ \qquad y = 45x_3^2 - 30x_2x_3 + 10x_1x_3 + 3x_2^2 - 40x_1x_2 + 8x_1^2). \end{array}$$

By eliminating x_3 using *virtual substitution* [29] we obtain the following QE problem:

$$\begin{array}{l} \exists x_1 \exists x_2 \quad (x_1 \ge 0 \ \wedge \ x_2 \ge 0 \ \wedge \\ \qquad t \ge x_1 + x_2 \ \wedge \ 15t - 10x_1 - 19x_2 \ge 100000 \ \wedge \\ \qquad y = 45t^2 + 80tx_1 + 120tx_2 - 43x_1^2 - 70x_1x_2 - 78x_2^2). \end{array}$$

mooea: extended problem of example 1 in [12, p. 11]:

$$\begin{array}{ll} \text{Minimize} & x_1^2 + x_2^2 + x_3 \text{ and} \\ \text{minimize} & (x_1 - 1)^2 + x_2^2 + x_3 \\ \text{subject to} & -2 \le x_1 \le 2, -2 \le x_2 \le 2, -1 \le 10x_3 \le 1. \end{array}$$

The associated QE problem is given as:

$$\begin{array}{l} \exists x_1 \exists x_2 \exists x_3 \quad (y_1 = x_1^2 + x_2^2 + x_3 \ \wedge \\ \qquad y_2 = (x_1 - 1)^2 + x_2^2 + x_3 \ \wedge \\ \qquad -2 \le x_1 \le 2 \ \wedge \ -2 \le x_2 \le 2 \ \wedge \ -1 \le 10x_3 \le 1). \end{array}$$

wilson [47]:

$$\begin{array}{ll} \text{Minimize} & (x_1 - 2)^2 + (x_2 - 1)^2 \text{ and} \\ \text{minimize} & x_1^2 + (x_2 - 6)^2 \\ \text{subject to} & 2/5 \le x_1 \le 8/5, 2 \le x_2 \le 5. \end{array}$$

The associated QE problem is given as:

$$\begin{array}{l} \exists x_1 \exists x_2 \exists x_3 \quad (\\ \qquad y_1 = (x_1 - 2)^2 + (x_2 - 1)^2 \ \wedge \ y_2 = x_1^2 + (x_2 - 6)^2 \ \wedge \\ \qquad 2/5 \le x_1 \le 8/5 \ \wedge \ 2 \le x_2 \le 5). \end{array}$$

lampinen [28, p. 6]:

$$\begin{array}{ll} \text{Minimize} & x_1^2 + x_2 \text{ and} \\ \text{minimize} & x_1 + x_2^2 \\ \text{subject to} & -10 \le x_1 \le 10, -10 \le x_2 \le 10. \end{array}$$

The associated QE problem is given as:

$$\begin{array}{l} \exists x_1 \exists x_2 \exists x_3 \quad (\\ \qquad y_1 = x_1^2 + x_2 \ \wedge \ y_2 = x_1 + x_2^2 \ \wedge \\ \qquad -10 \le x_1 \le 10 \ \wedge \ -10 \le x_2 \le 10). \end{array}$$

We have implemented all the algorithms presented in the previous Sections 4 and 5 in the C language with a Maple library as a part of SyNRAC [24]. We use the Maple library as a symbolic solver over the rational number field.

Comparisons of computing time with some QE implementations based on CAD are shown in Table 1. Symbols SyN2011, QEP, and MATH stand for SyNRAC, QEPCAD B 1.58 [7], and Mathematica 8.0, respectively and SyN2009 means SyNRAC implementation presented in SNC 2009 [25] which is implemented in the Maple user language and does not apply the approaches in Sections 4 and 5. All the computation was executed on a PC with Intel(R) Core(TM) i3 CPU U330 1.20 GHz and 2.92 GByte memory. All timing data are given in second (CPU-time). QEPCAD B was executed with options "+N40000000 +L100000." In the column of QEP, "-" implies that the program terminated with an error.

Table 2 shows the number of cells produced in CAD for each problem. In the first four examples the numbers of cells produced by Mathematica were taken from [41]. The other data for Mathematica were not obtained (marked "*"). In the column of SyN2009 and QEP, "-" implies that the program terminated with an error or did not terminate in an hour.

Tables 3 and 4 tell us the numbers of occurrences of important events in our algorithms SyN2009 and SyN2011, respectively. The first column "prec" shows "the number of times raising the precision." The second one "zero check" represents

problem	SyN2011	SyN2009	QEP	MATH
adam1	0.20	0.01	0.95	0.88
adam2-1	2.64	13.09	326	2.75
adam2-2	4.77	63.82	806	4.15
adam3	5.99	58.63	>3600	2.31
pl01	0.45	0.51	0.08	0.19
lass	0.36	2.49	0.10	0.43
candj	0.43	1.91	0.11	0.32
xaxis	0.58	8.75	0.18	0.29
portfolio	1.33	>3600	524	272
port-nox3	5.97	>3600	9.62	52.55
port-para	10.92	>3600	128	103
mooea	32.31	>3600	-	9.74
wilson	19.63	>3600	-	28.92
lampinen	11.18	>3600	198	13.94

Table 1: Computing time (sec)

problem	SyN2011	SyN2009	QEP	MATH
adam1	21	13	58	81
adam2-1	2922	3027	6835	4853
adam2-2	9677	9799	11653	5053
adam3	10588	7661	-	32606
pl01	133	123	225	*
lass	332	899	1328	*
candj	613	567	685	*
xaxis	2855	2791	3029	*
portfolio	5413	-	763190	*
port-nox3	20069	-	76470	*
port-para	46848	-	520953	*
mooea	496277	-	-	*
wilson	211612	-	-	*
lampinen	73289	-	396818	*

Table 2: Number of cells

problem	prec	zero check	sqrfree
adam1	0	1/1	0/0/7
adam2-1	7	155/155	68/70/1640
adam2-2	1696	756/902	346/355/8340
adam3	49	1040/1054	160/170/5678
pl01	0	10/10	0/0/92
lass	0	50/50	2/4/678
candj	0	48/48	0/1/398
xaxis	1	358/359	15/15/1723

Table 3: Numbers of occurrences (SyNRAC 2009)

problem	prec	zero check	sqrfree
adam1	0	0/0	0/0/0/0
adam2-1	0	0/0	2/5/5/171
adam2-2	0	25/27	34/167/167/911
adam3	3	448/480	21/99/99/782
pl01	0	0/0	0/3/3/12
lass	0	6/6	0/2/2/25
candj	0	6/6	0/1/1/66
xaxis	0	32/32	14/14/14/302
portfolio	0	11/11	27/27/27/623
port-nox3	0	128/128	580/595/595/7354
port-para	0	27/27	491/662/662/8458
mooea	0	47/47	584/596/596/70655
wilson	0	0/0	297/297/297/35266
lampinen	0	522/522	72/179/179/21253

Table 4: Numbers of occurrences (SyNRAC 2011)

problem	SyN2011	SyN2009	QEP
adam1	8	8	8
adam2-1	68	75	75
adam2-2	83	93	83
adam3	43	43	43
pl01	10	12	12
lass	21	21	21
candj	17	19	19
xaxis	30	30	30
portfolio	68	183	183
port-nox3	76	76	76
port-para	124	124	124
mooea	255	255	255
wilson	310	310	310
lampinen	73	73	73

Table 5: Numbers of projection factors

"true cases / the number of zero checks executed." The last column "sqrfree" shows "the number of square-free computations by (4) / the number of worked square-free computations / the number of executed square-free computations / the number of lifting cells." In Table 3 there is not the number of square-free computations by (4).

Table 5 shows the number of projection factors. We could not know at all the numbers of projection factors produced by Mathematica, omitted in the table.

Effect of our projection operator: Table 5 shows that our projection operator produces a smaller set of projection factors than the other implementations. We focus only optimization problems, but we can see that our approach works for some of other QE problems.

The problem portfolio has a redundant constraint $x_3 \geq 0$. Interestingly our projection operator in the problem portfolio produces a smaller set than in the problem port-nox3 which is removed the redundant constraint from portfolio, and computation time is shorter. The other implementations have the opposite result.

Effect of utilization of sign information: The significant effect of utilization of sign information can be seen by comparing Tables 3 and 4. The second and third columns in Tables 3 and 4 show that SyN2011 avoids a lot of symbolic computation of zero check and square-free. Moreover, the third column in Table 4 shows that unnecessary square-free computation does not execute and most of square-free computation is applied for polynomials of degree two.

Number of cells: Table 2 shows that SyN2011 produces a smaller number of cells than the others by virtue of the techniques presented in Sections 4 and 5.3.1. Avoiding many computation reduces the number of times raising the precision of numeric computation.

7. CONCLUSION

In this paper we have proposed the effective approach for bounded CAD construction and utilization of sign information. The bounded CAD construction approach makes smaller sets of projection factors and avoids lifting the more cells. The utilization of sign information avoid a lot of symbolic zero check and square-free computation. Our approaches originally aim at speeding up of solving polynomial optimization problems. However, it seems that the properties we exploit in this paper are actually common in our benchmark problems. Thus we can expect our approaches might be effective for relatively wide range of problems.

Furthermore, we think that our idea of bounded CAD construction is applicable to the *virtual substitution* algorithm [29] as positive QE approach [42].

8. ACKNOWLEDGMENTS

This work has been partially supported by Global COE Program "Mathematics-for-Industry" of Kyushu University. We would like to express our gratitude to Dr. A. Strzeboński and Mr. K. Ouchi for giving useful information on Mathematica implementation. We would like to thank anonymous referees for useful comments.

9. REFERENCES

[1] H. Anai and S. Hara. Fixed-structure robust controller synthesis based on sign definite condition by a special quantifier elimination. In *Proceedings of American Control Conference, 2000*, volume 2, pages 1312–1316, 2000.

[2] H. Anai and P. Parrilo. Convex quantifier elimination for semidefinite programming. In *Proceedings of the International Workshop on Computer Algebra in Scientific Computing (CASC 2003)*, pages 3–11, Sept. 2003.

[3] H. Anai, H. Yanami, S. Hara, and K. Sakabe. Fixed-structure robust controller synthesis based on symbolic-numeric computation: design algorithms with a CACSD toolbox. In *Proceedings of CCA/ISIC/CACSD 2004 (Taipei, Taiwan)*, volume 2, pages 1540–1545, 9 2004.

[4] H. Anai and K. Yokoyama. Cylindrical algebraic decomposition via numerical computation with validated symbolic reconstruction. In A. Dolzman, A. Seidl, and T. Sturm, editors, *Algorithmic Algebra and Logic*, pages 25–30, 2005.

[5] C. W. Brown. Improved projection for CAD's of $\mathbb{R}^3$. In *Proceedings of the 2000 international symposium on Symbolic and algebraic computation*, ISSAC '00, pages 48–53, New York, NY, USA, 2000. ACM.

[6] C. W. Brown. Improved projection for cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 32(5):447–465, 2001.

[7] C. W. Brown. QEPCAD B: A program for computing with semi-algebraic sets using CADs. *SIGSAM BULLETIN*, 37:97–108, 2003.

[8] G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *LNCS*, volume 32. Springer Verla, 1975.

[9] G. E. Collins and H. Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12(3):299–328, 1991.

[10] G. E. Collins, J. R. Johnson, and W. Krandick. Interval arithmetic in cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 34(2):145–157, 2002.

[11] J. H. Davenport and J. Heintz. Real quantifier elimination is doubly exponential. *Journal of Symbolic Computation*, 5(1/2):29–35, 1988.

[12] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley-Interscience Series in Systems and Optimization. John Wiley & Sons, Chichester, 2001.

[13] A. Dolzmann, T. Sturm, and V. Weispfenning. Real quantifier elimination in practice. In *Algorithmic Algebra and Number Theory*, pages 221–247. Springer, 1998.

[14] J. D. Dora, C. Dicrescenzo, and D. Duval. About a new method for computing in algebraic number fields. In *Research Contributions from the European Conference on Computer Algebra-Volume 2*, EUROCAL '85, pages 289–290, London, UK, 1985. Springer-Verlag.

[15] D. Duval. Algebraic numbers: An example of dynamic evaluation. *Journal of Symbolic Computation*, 18(5):429–445, 1994.

[16] M. S. El Din. Computing the global optimum of a multivariate polynomial over the reals. In *Proceedings of the twenty-first international symposium on Symbolic and algebraic computation*, ISSAC '08, pages 71–78, New York, NY, USA, 2008. ACM.

[17] J.-C. Faugère, G. Moroz, F. Rouillier, and M. S. El Din. Classification of the perspective-three-point problem, discriminant variety and real solving polynomial systems of inequalities. In *Proceedings of the twenty-first international symposium on Symbolic and algebraic computation*, ISSAC '08, pages 79–86, New York, NY, USA, 2008. ACM.

[18] L. González-Vega. Applying quantifier elimination to the Birkhoff interpolation problem. *Journal of Symbolic Computation*, 22:83–103, July 1996.

[19] L. González-Vega. *A combinatorial algorithm solving some quantifier elimination problems*, pages 365–375. Texts and Monographs in Symbolic Computation. Springer, 1998.

[20] F. Guo, M. S. El Din, and L. Zhi. Global optimization of polynomials using generalized critical values and sums of squares. In *Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation*, ISSAC '10, pages 107–114, New York, NY, USA, 2010. ACM.

[21] H. Hong. An improvement of the projection operator in cylindrical algebraic decomposition. In *Proceedings of the international symposium on Symbolic and algebraic computation*, pages 261–264, New York, NY, USA, 1990. ACM.

[22] H. Hong. Quantifier elimination for formulas constrained by quadratic equations. In *Proceedings of*

the 1993 international symposium on Symbolic and algebraic computation, ISSAC '93, pages 264–274, New York, NY, USA, 1993. ACM.

[23] H. Hong. An efficient method for analyzing the topology of plane real algebraic curves. In *Selected papers presented at the international IMACS symposium on Symbolic computation, new trends and developments*, pages 571–582, Amsterdam, The Netherlands, The Netherlands, 1996. Elsevier Science Publishers B. V.

[24] N. Hyodo, M. Hong, H. Yanami, S. Hara, and H. Anai. Solving and visualizing nonlinear parametric constraints in control based on quantifier elimination: A matlab toolbox for parametric control system design. *Appl. Algebra Eng., Commun. Comput.*, 18:497–512, November 2007.

[25] H. Iwane, H. Yanami, H. Anai, and K. Yokoyama. An effective implementation of a symbolic-numeric cylindrical algebraic decomposition for quantifier elimination. In *Proceedings of the 2009 International Workshop on Symbolic-Numeric Computation*, volume 1, pages 55–64, 2009.

[26] E. Kaltofen, B. Li, Z. Yang, and L. Zhi. Exact certification of global optimality of approximate factorizations via rationalizing sums-of-squares with floating point scalars. In *Proceedings of the twenty-first international symposium on Symbolic and algebraic computation*, ISSAC '08, pages 155–164, New York, NY, USA, 2008. ACM.

[27] R. Krawczyk. Newton-algorithmen zur bestimmung von nullstellen mit fehlerschranken. *Computing*, pages 187–201, 1969.

[28] J. Lampinen. *Multiobjective Nonlinear Pareto-Optimization*. Laboratory of Information Processing, 2000.

[29] R. Loos and V. Weispfenning. Applying linear quantifier elimination. *The Computer Journal*, 36(5):450–462, 1993.

[30] S. McCallum. Solving polynomial strict inequalities using cylindrical algebraic decomposition. *The Computer Journal*, pages 432–438, 1993.

[31] S. McCallum. An improved projection operator for cylindrical algebraic decomposition. In B. F. Caviness and J. R. Johnson, editors, *Quantifier elimination and cylindrical algebraic decomposition*, Texts and Monographs in Symbolic Computation, pages 242–268. Springer, 1998.

[32] S. McCallum. On projection in CAD-based quantifier elimination with equational constraint. In *Proceedings of the 1999 international symposium on Symbolic and algebraic computation*, ISSAC '99, pages 145–149, New York, NY, USA, 1999. ACM.

[33] S. McCallum. On propagation of equational constraints in CAD-based quantifier elimination. In *Proceedings of the 2001 international symposium on Symbolic and algebraic computation*, ISSAC '01, pages 223–231, New York, NY, USA, 2001. ACM.

[34] P. A. Parrilo and S. Lall. Semidefinite programming relaxations and algebraic optimization in control, 2006. Mini-course on Polynomial Equations and Inequalities I and II.

[35] P. A. Parrilo and B. Sturmfels. Minimizing polynomial functions. In *Algorithmic and Quantitative Aspects of Real Algebraic Geometry in Mathematics and Computer Science*, volume 60, pages 83–99. ACM, 2001.

[36] S. Ratschan. Approximate quantified constraint solving by cylindrical box decomposition. *Reliable Computing*, 8(1):21–42, 2002.

[37] J. Schattman. Portfolio optimization under nonconvex transaction costs with the global optimization toolbox. `http://www.maplesoft.com/applications/view.aspx?SID=1401&view=html`.

[38] M. Schweighofer. Global optimization of polynomials using gradient tentacles and sums of squares. *SIAM J. on Optimization*, 17:920–942, September 2006.

[39] A. W. Strzeboński. A real polynomial decision algorithm using arbitrary-precision floating point arithmetic. *Reliable Computing*, 5(3):337–346, 1999.

[40] A. W. Strzeboński. Solving systems of strict polynomial inequalities. *Journal of Symbolic Computation*, 29:471–480, March 2000.

[41] A. W. Strzeboński. Cylindrical algebraic decomposition using validated numerics. *Journal of Symbolic Computation*, 41(9):1021–1038, 2006.

[42] T. Sturm and A. Weber. Investigating generic methods to solve hopf bifurcation problems in algebraic biology. In K. Horimoto, G. Regensburger, M. Rosenkranz, and H. Yoshida, editors, *Algebraic Biology*, volume 5147 of *Lecture Notes in Computer Science*, chapter 15, pages 200–215. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2008.

[43] T. Sturm and V. Weispfenning. Rounding and blending of solids by a real elimination method. In *Proceedings of the 15th IMACS World Congress on Scientific Computation, Modelling, and Applied Mathematics (IMACS 97)*, pages 727–732, 1997.

[44] V. Weispfenning. The complexity of linear problems in fields. *Journal of Symbolic Computation*, 5:3–27, February 1988.

[45] V. Weispfenning. Quantifier elimination for real algebra - the quadratic case and beyond. *AAECC*, 8:85–101, 1993.

[46] V. Weispfenning. Simulation and optimization by quantifier elimination. *Journal of Symbolic Computation*, 24:189–208, August 1997.

[47] B. Wilson, D. Cappelleri, T. W. Simpson, and M. Frecker. Efficient Pareto frontier exploration using surrogate approximations. *Optimization and Engineering*, 2:31–50, 2001.

Challenge to Fast and Stable Computation of Approximate Univariate GCD, Based on Displacement Structures

Masaru Sanuki
Center for Research on International Cooperation in Educational Development (CRICED)
University of Tsukuba
Tsukuba-Shi, Ibaraki 305-8572, Japan
sanuki@criced.tsukuba.ac.jp

ABSTRACT

Given polynomials with floating-point number coefficients, one can now compute the approximate GCD stably, except in ill-conditioned cases where the GCD has small or large leading coefficient/constant term. The cost is $O(m^2)$, where m is the maximum of degrees of given polynomials. On the other hand, for polynomial with integer coefficients, one can compute the polynomial GCD faster by using the half-GCD method with the cost less than $O(m^2)$. In this paper, we challenge to compute the approximate GCD faster, with the cost less than $O(m^2)$. Our idea is to use the displacement technique and the half-GCD method.

Categories and Subject Descriptors

I.1.2 [**Computing Methodologies**]: Symbolic and Algebraic Manipulation—*Algorithms*; G.1.2 [**Mathematics of Computing**]: Numerical Analysis—*Approximation*

General Terms

Algorithms, Experimentation

Keywords

approximate GCD, displacement technique, half-GCD method, symbolic-numeric computation

1. INTRODUCTION

Let $\mathbb{F}[x]$ be the ring of polynomials over $\mathbb{F}$, the set of floating-point numbers with the machine epsilon ε_{M}, in main variable x. Let the given polynomial $f(x) \in \mathbb{F}[x]$ be expressed as $f(x) = f_m x^m + \cdots + f_0,\ f_m \neq 0$. By $\deg(f)$, $\mathrm{lc}(f)$ and $\mathrm{rest}(f)$, we denote the degree, the leading coefficient and the rest terms of f w.r.t. the main variable x, respectively; $\mathrm{rest}(f) = f - \mathrm{lc}(f)x^{\deg(f)}$. By $||f||$, we denote the norm of f; we use the infinity norm in this paper, $||f|| = \max\{|f_m|, \ldots, |f_0|\}$. For two polynomials $f(x)$ and $g(x)$ with $m = \deg(f), n = \deg(n)$, by $\mathrm{quo}(f, g)$ and $\mathrm{rem}(f, g)$, we denote the quotient, the remainder, respectively, of f by g. By $\gcd(f, g)$ and $\mathrm{appgcd}(f, g; \varepsilon)$, we denote the greatest common divisor (GCD) and the approximate GCD (appGCD) of tolerance ε, of f and g, respectively. The appGCD is not unique, but depends on the tolerance.

Computing the appGCD has been studied by many authors, and many methods were proposed so far: methods for univariate polynomials are quasi-GCD method [25], PRS (polynomial remainder sequence) [28] and stabilized PRS methods [32], SVD (singular-value decomposition) methods [6, 36, 17], ϵ-GCD method [9], QRGCD method [18, 38, 7], Padé-GCD method [19], methods based on structured-matrix [37, 3, 14], STLN (structured total least norm) methods [13, 34], method using Barnett's theorem [8], subspace method [21], ERES (Extended-Row-Equivalence and Shifting operations) method [15], and so on [2]. The PRS method is unstable when the leading coefficient is small, however, the stabilized PRS method is able to compute the appGCD stably even if the leading coefficient is small, and it is efficient; the cost is $O(m^2)$ usually. The QRGCD method computes the appGCD stably and directly by using the QR-decomposition of the Sylvester matrix of given polynomials. This method is similar to the stabilized PRS method, the difference is how thoroughly the pivoting is made, and the cost is $O(m^3)$. Zhi [37] and Bini-Boito [3] improved the cost from $O((m+n)^3)$ to $O(5(m+n)^2)$ and $O(2m^2)$, respectively. Bini-Boito's method is based on the GKO (Gohberg-Kailath-Olshevsky) method, which performs LU-decomposition of the Cauchy-like matrix with partial pivoting [10], and it computes appGCD stably and efficiently in most cases [27, 26]. On the other hand, Zhi's method is based on the fast QR-decomposition without pivoting. Although both methods utilize the displacement technique, Zhi's method often becomes unstable, because it does not make pivoting.

The half-GCD method with integer coefficients is known as a fast method for computing exact GCD and its cofactors [16]. However, since this method is based on the Euclidean elimination, the computation of appGCD becomes unstable. Therefore, it is impossible to apply the half-GCD method to our case without any improvement.

The main purpose of this paper is to compute the appGCD of univariate polynomials stably and faster than other methods. In **2**, we explain the stabilized PRS method [32] and improve the method to stabilize further. In **3**, we explain Zhi's method briefly and investigate the relationship between the method and the improved PRS method. In **4**, we propose a modified half-GCD method for polynomials with floating-point number coefficients, so as to compute the appGCD stably and efficiently. In **5**, we survey the GKO method and explain that our new technique cannot be incorporated with the GKO method. In **6**, we show some results of experiments. In **7**, we give conclusion and remarks.

1.1 Estimating cancellation errors

In order to estimate cancellation errors, we unitize *effective floating-point number*, or *efloat number*, in short. This number was proposed by Kako-Sasaki [12], and it allows us to estimate the cancellation error neglecting the rounding error. The efloat number is expressed as #E$[f, e]$; f is called the value-part and it is the same as conventional floating-point number, e is called error-part, showing the cancellation error. In our computational environment, we set $e = |f| \cdot \varepsilon_{\mathrm{M}}$ initially. Arithmetics of efloat numbers is as follows.

$$\begin{aligned}
\#\mathrm{E}[f_1, e_2] + \#\mathrm{E}[f_2, e_2] &= \#\mathrm{E}[f_1 + f_2, \max\{e_1, e_2\}],\\
\#\mathrm{E}[f_1, e_2] - \#\mathrm{E}[f_2, e_2] &= \#\mathrm{E}[f_1 - f_2, \max\{e_1, e_2\}],\\
\#\mathrm{E}[f_1, e_2] \times \#\mathrm{E}[f_2, e_2] &= \#\mathrm{E}[f_1 \times f_2, \max\{e_1|f_2|, e_2|f_1|\}],\\
\#\mathrm{E}[f_1, e_2] \div \#\mathrm{E}[f_2, e_2] &= \#\mathrm{E}[f_1 \div f_2, \max\{e_1\left|\frac{f_2}{{f_1}^2}\right|, \frac{e_2}{|f_1|}\}].
\end{aligned}$$

If $|f| < e$ then we regard #E$[f, e]$ as 0. The complex number $g = a + b\hat{\imath}$, where $a, b \in \mathbb{R}$ and $\hat{\imath}$ is the imaginary unit, is converted to #E$[a, e_a]$ + #E$[b, e_b]\,\hat{\imath}$; this conversion is different from the original one in [12].

2. IMPROVING PRS METHODS

2.1 Stabilized PRS method

We first explain the stabilized method. Let $P_1(x)$ and $P_2(x)$ be polynomials in $\mathbb{F}[x]$, expressed as follows ($f_m \neq 0, g_n \neq 0$ and $m \geq n$).

$$P_1(x) = f(x) = f_m x^m + \cdots + f_0, \qquad (2.1)$$

$$P_2(x) = g(x) = g_n x^n + \cdots + g_0. \qquad (2.2)$$

Below, every given polynomial f is normalized as $||f|| = 1$.

The PRS method is the oldest method for computing polynomial GCD. However, it is unstable when the leading coefficient of P_i is small, where P_i is an element of PRS ($P_1, P_2, \ldots, P_i = \mathrm{rem}(P_{i-2}, P_{i-1}), \ldots$). The instability is caused by the term cancellation. The cancellation mechanism in the PRS method with floating-point number coefficients may be explained as follows [32, 30].

If $\delta = |g_n|/|f_m| \ll 1$ then the remainder $h = \mathrm{rem}(f, g)$ is nearly a multiple of $\mathrm{rest}(g)$ (in [32], h is called a *dummy* of g and denoted by $\mathrm{dummy}(g)$). Hence, cancellations of $O(1/\delta^{d+1})$ occur in the computation of $\mathrm{rem}(g, h)$, where $d = m - n \geq 0$. We call the reduction of g by h *self-reduction.* In other words, the self-reduction is the reduction of g by $\mathrm{dummy}(g)$. In order to compute the appGCD stably using the PRS, we have to avoid the self-reduction. Then, we are led to the stabilized PRS method [32]. In this method, we perform the reduction of P_{i-1} and P_i ($\deg(P_{i-1}) < \deg(P_i)$) for $i \geq 2$, as follows.

- If $|\mathrm{lc}(P_i)|/|\mathrm{lc}(P_{i-1})| < \eta \leq 1$, we compute P_{i+1} as
$$P_{i+1} = (x^{d_i} - a)P_i - \frac{\mathrm{lc}(P_i)}{\mathrm{lc}(P_{i-1})}P_{i-1}, \qquad (2.3)$$
where $d_i = \deg(P_{i-1}) - \deg(P_i)$ and a is a polynomial such that $\deg(a) < d_i$ and $||a|| \approx 1$. Then, we reduce P_{i-1} by P_{i+1}.
- Otherwise, we compute $P_{i+1} = \mathrm{rem}(P_{i-1}, P_i) = P_{i-1} - \mathrm{quo}(P_{i-1}, P_i)P_i$.

The stabilized PRS method generates cofactor sequence $((A_1, B_1), (A_2, B_2), \ldots, (A_k, B_k))$, satisfying

$$A_i P_1 + B_i P_2 = P_i \ (i = 1, 2, \ldots). \qquad (2.4)$$

It should be noted that in the stabilized PRS method, the conventional degree conditions $\deg(A_i) < \deg(P_2) - \deg(P_i)$ and $\deg(B_i) < \deg(P_1) - \deg(P_i)$ are not satisfied [32].

REMARK 1 ([32, 22]). We had set $\eta = 0.2$ in [32] and $\eta = 1.0$ in [22]. If we use the stabilized PRS method with small η, the computation sometimes becomes unstable for polynomials of large degrees. The stabilized PRS method with $\eta = 1$ is close to the QRGCD method. When we use the stabilized PRS method to polynomials of large degree, we are necessary to set the precision higher.

2.2 Double-side reduction

In Zhi's method, not only the leading coefficient but also the constant term are eliminated. This leads us to introduce the following reduction, which we call *double-side reduction.*

DEFINITION 1 (DOUBLE-SIDE REDUCTION). *Let f and g be polynomials with $f_m \cdot g_n \neq 0$ and $f_0 \cdot g_0 \neq 0$. A double-side reduction of f and g is defined as*

$$\begin{pmatrix} f' \\ g' \end{pmatrix} := \text{d-red}\begin{pmatrix} f \\ g \end{pmatrix} = \begin{pmatrix} 1 & -\gamma x^d \\ -\omega/x & 1/x \end{pmatrix}\begin{pmatrix} f \\ g \end{pmatrix}, \qquad (2.5)$$

where

$$\gamma = f_m/g_n, \quad \omega = g_0/f_0 \qquad (2.6)$$

and $d = m - n \geq 0$. By this operation, we can reduce the leading coefficient of f and the constant term of g simultaneously. When $f_0 \cdot g_0 = 0$, we first transform (f, g) to (f', g') as

$$\begin{pmatrix} f' \\ g' \end{pmatrix} = \begin{pmatrix} 1 & a \\ \pm a & 1 \end{pmatrix}\begin{pmatrix} f \\ g \end{pmatrix}, \qquad (2.7)$$

where a is a number such that $|a| \approx 1$, then we apply double-side reduction to f' and g'. Then, we have

$$\deg(f'), \deg(g') < \max\{\deg(f), \deg(g)\}.$$

We note that the double-side reduction does not assure $\gcd(f, g) = \gcd(f', g')$. For example, we perform the double-side reduction of $f_1 = x^5 + 1$ and $g_1 = x^3 + 1$. Then,

$$\begin{pmatrix} f_1' \\ g_1' \end{pmatrix} = \begin{pmatrix} 1 & -x^2 \\ -1/x & 1/x \end{pmatrix}\begin{pmatrix} f_1 \\ g_1 \end{pmatrix} = \begin{pmatrix} x^2 - 1 \\ -x^2(x^2 - 1) \end{pmatrix}.$$

Thus, $x + 1 = \gcd(f_1, g_1) \neq \gcd(f_1', g_1') = x^2 - 1$. Another example is $f_2 = x^5 + 1$ and $g_2 = x^3 - 1$:

$$\begin{pmatrix} f_2' \\ g_2' \end{pmatrix} = \begin{pmatrix} 1 & -x^2 \\ -1/x & 1/x \end{pmatrix}\begin{pmatrix} f_2 \\ g_2 \end{pmatrix} = \begin{pmatrix} x^2 + 1 \\ -x^2(x^2 + 1) \end{pmatrix}.$$

We see $\gcd(f_2', g_2') = x^2 + 1 \neq \gcd(f_2, g_2) = 1$.

Every element $P_i(x)$ in the dual-PRS is represented as

$$P_i = \frac{A_i}{x^{\alpha_i}}f + \frac{B_i}{x^{\alpha_i}}g \in \mathbb{F}[x].$$

Hence $\gcd(f, g)$ divides $x^{\alpha_i}P_i$ for all i. If $\gcd(f, P_i) = 1$ or $\gcd(g, P_i) = 1$ then $\gcd(f, g) = 1$. On the other hand, since

$$\gcd(f', g') = \gcd(\gcd(\mathrm{rest}(f) - \gamma x^d \mathrm{rest}(g)), -w\tilde{f} + \tilde{g}),$$

$\gcd(f', g')$ may be divided by $\gcd(\mathrm{GCD}_{fg}, \mathrm{GCD}_{\tilde{f}\tilde{g}})$, where $\tilde{f} = (f - f_0)/x$, $\tilde{g} = (g - g_0)/x$, $\mathrm{GCD}_{fg} = \gcd(\mathrm{rest}(f), \mathrm{rest}(g))$ and $\mathrm{GCD}_{\tilde{f}\tilde{g}} = \gcd(\tilde{f}, \tilde{g})$. Note that $\tilde{f}$ and $\tilde{g}$ are polynomials, and that $\mathrm{rest}(f), \mathrm{rest}(g), \tilde{f}$ and $\tilde{g}$ are not elements of the dual-PRS. Actually, f and g are expressed by cofactors as

$$\begin{aligned} f_1 &= (x-1)(x^4 + x^3 + x^2 + x + 1) \\ &= (x-1) \times (\underline{(x+1)(x^3+x)} + 1) \\ &= (x-1) \times (x^4 + \underline{(x+1)(x^2+1)}), \\ g_1 &= (x-1) \times (\underline{(x+1)x} + 1) \\ &= (x-1) \times (x^2 + \underline{(x+1)}), \end{aligned}$$

so $\gcd(f_1', g_1') = (x-1) \times (x+1)$. On other other hand, f and g are split into two parts, as follows.

$$\begin{aligned} f &= f^{(1)} + ef^{(2)} = ef^{(3)}x + f^{(4)}, \\ g &= g^{(1)} + eg^{(2)} = eg^{(3)}x + g^{(4)}. \end{aligned}$$

Thus, $f' = g^{(1)}f - f^{(1)}g = e(g^{(1)}f^{(2)} - f^{(1)}g^{(2)})$ and $g' = (g^{(4)}f - f^{(4)}g)/x = e(g^{(4)}f^{(3)} - f^{(4)}g^{(3)})$, hence we have $\gcd(f', g') = e$. Similarly, f_2 and g_2 are expressed as

$$\begin{aligned} f_2 &= (x-1)(\underline{(x^2+1)(x^2+x)} + 1) \\ &= (x-1)(x^4 + \underline{(x^2+1)(x+1)}), \\ g_2 &= (x+1)(\underline{x^2} - x\underline{+1}). \end{aligned}$$

We see $\mathrm{rest}(x^2 - x + 1) \neq x^2 + 1$, $\gcd(f_1', g_2') = x^2 + 1$. Therefore, we must check whether or not polynomial obtained is the GCD.

Remark 2 (occurence of self-reduction).
The case $\gcd(f, g) \neq \gcd(f', g')$ is also caused by large errors due to the self-reduction. We consider the following three cases.

1. Case 1: $|\gamma| \approx |\omega|$.
   If $|\gamma| \approx |\omega| \approx 1$ then no self-reduction occurs and we can perform the reduction of f and g stably. Furthermore, if $|\gamma| \approx |\omega| \gg 1$, since $f' = f - \gamma x^d g = \mathrm{dummy}(f)$ and $g' = (-\omega f + g)/x = \mathrm{dummy}(g)$, no cancellation error occurs in the succeeding reduction of f' and g'. Similarly, if $|\gamma| \approx |\omega| \ll 1$, no cancellation error occurs in the succeeding reduction of $f' = \mathrm{dummy}(f)$ and $g' = \mathrm{dummy}(g)$.
2. Case 2: $|\gamma| \ll 1 \ll |\omega|$.
   In this case, f' and g' are nearly multiples of f. Then, self-reductions of $O(|\omega/\gamma|)$ occur in the succeeding double-side reduction of f' and g'.
3. Case 3: $|\gamma| \gg 1 \gg |\omega|$.
   In this case, f' and g' are nearly a multiples of g. Then, self-reductions of $O(|\gamma/\omega|)$ occur in the succeeding double-side reduction of f' and g'.

The reason why we introduced the double-side reduction in that we can perform pivoting in the reduction. Pivoting is performed in Cases 2 and 3: we transform (f, g) to $(\hat{f}, \hat{g})$, as follows.

$$\begin{pmatrix} \hat{f} \\ \hat{g} \end{pmatrix} = \begin{pmatrix} 1 & a \\ b & 1 \end{pmatrix} \begin{pmatrix} f \\ g \end{pmatrix}, \quad a, b \in \mathbb{F} \text{ such that } |a|, |b| \approx 1.$$

Then, $(\hat{f}, \hat{g})$ satisfies $|\mathrm{lc}(\hat{f})|/|\mathrm{lc}(\hat{g})| \approx 1$ and $|\hat{g}_0|/|\hat{f}_0| \approx 1$, hance this operation is pivoting. In particular, if we set $b = -a$ and $b = a$ then the above transformed matrix becomes the Givens rotation matrix and the hyperbolic rotation matrix, respectively.

We call PRS method using double-side reduction with pivoting *dual-PRS method*, which is described in the following algorithm. The cost of dual-PRS method is $O(m^2)$. Below, pd-red denote the double-side reduction with pivoting.

Algorithm 1 (Dual-PRS method).

```
 Input:  $f = P_1(x), g = P_2(x)$ and tolerance $\varepsilon$.
Output:  $\gcd(f, g)$
         $i := 2$;
         $(P_3 := f, P_4 := g)$
    LP:  if $\deg(P_{2i-1}) = 0$ or $\deg(P_{2i}) = 0$
           then $P_{2i-1}, P_{2i}$ are candidates of appGCD.
         else if $||P_{2i+1}|| < O(\varepsilon)$ or $||P_{2i+2}|| < O(\varepsilon)$
           then $P_{2i-1}, P_{2i}$ are candidates of appGCD.
           (if $f, g \in \{P_{2i-1}, P_{2i}\}$, then GCD)

         otherwise $(P_{2i+1}, P_{2i+2}) = $ pd-red$(P_{2i-1}, P_{2i})$.
         $i := i + 1$;
         goto LP;
```

Proposition 1. *The dual-PRS method terminates and outputs the GCD.*

Proof. Assuming that P_k is a candidate of $\gcd(f, g)$, we compute $\gcd(f, P_k)$ or $\gcd(g, P_k)$ by applying the dual-PRS method. When $\gcd(f, P_k) = 1$ or $\gcd(g, P_k) = 1$, f and g are relatively prime, and when $\gcd(f, P_i) = \tilde{P_{k'}}$ with $\deg(\tilde{P_{k'}}) < \deg(P_k)$, we compute $\gcd(f, \tilde{P_{k'}})$ or $\gcd(f, \tilde{P_{k'}})$. Since $\deg(P_k) < \max\{\deg(f), \deg(g)\}$, algorithm terminates and the appGCD is obtained $\square$

The dual-PRS method cannot generate polynomial cofactors A_i and B_i, which satisfy (2.4). Usually, A_i and B_i are in $\mathbb{F}[x, x^{-1}]$.

Remark 3 (reduction and transformation).
We can carry out double-side reduction and transformation with pivoting simultaneously, as follows (case of $d = 0$).

$$\begin{aligned} &\begin{pmatrix} 1 & \hat{\gamma} \\ \hat{w}/x & 1/x \end{pmatrix} \begin{pmatrix} 1 & a \\ b & 1 \end{pmatrix} \\ = &\begin{pmatrix} 1 + \hat{\gamma}b & a + \hat{\gamma} \\ \dfrac{(\hat{w} + b)}{x} & \dfrac{(\hat{w}a + 1)}{x} \end{pmatrix} \\ = &\begin{pmatrix} \dfrac{t(b)\mathrm{lc}(f) + t(ab)\mathrm{lc}(g)}{b \cdot \mathrm{lc}(f) + \mathrm{lc}(g)} & \dfrac{t(ab)\mathrm{lc}(f) + t(a)\mathrm{lc}(g)}{b \cdot \mathrm{lc}(f) + \mathrm{lc}(g)} \\ \Big(\dfrac{t(b)f_0 + t(ab)g_0}{f_0 + a \cdot g_0}\Big)\Big/x & \Big(\dfrac{t(ab)f_0 + t(a)g_0}{f_0 + a \cdot g_0}\Big)\Big/x \end{pmatrix}, \end{aligned} \tag{2.8}$$

where the function $t(r)$ adds 1 to r, i.e., $t(r) = r + 1$ and

$$\hat{\gamma} = \frac{\mathrm{lc}(f) + a \cdot \mathrm{lc}(g)}{b \cdot \mathrm{lc}(f) + \mathrm{lc}(g)}, \ \hat{w} = \frac{b \cdot f_0 + g_0}{f_0 + a \cdot g_0}.$$

Here, we choose numbers $a, b \in \mathbb{F}$ as follows.

- If $|\gamma| \gg |w|$ then $t(b) \neq 0$ and $t(ab) \neq 0$.
- If $|\gamma| \ll |w|$ then $t(a) \neq 0$ and $t(ab) \neq 0$.

Then, the norm of each coefficient in (2.8) is about 1. Therefore, self-reduction does not occur in the reduction of f and g. In this paper, we choose numbers a, b to satisfy $0.9 \le |a|, |b| \le 1.1$ randomly. □

2.3 Ill-conditioned polynomials and conversion to well-conditioned ones

Although operation pd-red allows us to avoid the self-reduction, there are ill-conditioned polynomials for which the dual-PRS method (and most other methods, too) fails to compute appGCD.

Example 1. (well-conditioned case)

$$\begin{aligned} f(x) &= (x^2+1)(x^3+2x^2-1), \\ g(x) &= (x^2+1)(0.001x^3+x^2+1). \end{aligned}$$

Applying the dual-PRS method with pivoting, we obtain the following two polynomials as candidates of appGCD (we set at $\varepsilon_{\mathrm{M}} = 10^{-15}$ initially in this example).

$$-\#\mathrm{E}[0.528\cdots, 1.0\,\mathrm{e}{-15}]x^2 - \#\mathrm{E}[0.528\cdots, 1.0\,\mathrm{e}{-15}],$$
$$-\#\mathrm{E}[0.386\cdots, 1.0\,\mathrm{e}{-15}]x^2 - \#\mathrm{E}[0.386\cdots, 1.0\,\mathrm{e}{-15}].$$

In fact, each polynomial is an appGCD of f and g with tolerance $O(10^{-15+1}) = O(10^{-14})$. □

Example 2. (ill-conditioned case)

$$\begin{aligned} f(x) &= (0.001x^2+x+0.01)(x^3+3x^2-1), \\ g(x) &= (0.001x^2+x+0.01)(x^3+2x^2-x+1). \end{aligned}$$

The common factor $0.001x^2+x+0.01$ has small leading-coefficient and small constant term. Applying the dual-PRS method with pivoting, we obtain the following two appGCDs.

$$\begin{aligned} &\#\mathrm{E}[0.297\cdots\,\mathrm{e}{-8}, 1.0\,\mathrm{e}{-15}]x^2 \\ &+\#\mathrm{E}[0.297\cdots\,\mathrm{e}{-5}, 0.3\,\mathrm{e}{-14}]x \\ &+\#\mathrm{E}[0.297\cdots\,\mathrm{e}{-7}, 0.2\,\mathrm{e}{-16}] \end{aligned}$$

and

$$\begin{aligned} &\#\mathrm{E}[0.314\cdots\,\mathrm{e}{-6}, 0.3\,\mathrm{e}{-15}]x^2 \\ &+\#\mathrm{E}[0.314\cdots\,\mathrm{e}{-3}, 0.7\,\mathrm{e}{-15}]x \\ &+\#\mathrm{E}[0.314\cdots\,\mathrm{e}{-5}, 1.0\,\mathrm{e}{-15}]. \end{aligned}$$

We see big cancellation errors of $O(10^9)$ and $O(10^{12})$ occurred in the above appGCDs, respectively. In this example, we were able to compute appGCD with the sacrifice of big cancellation errors. In the ill-conditioned case, big cancellation occurs each time the leading term is eliminated. Hence, for polynomials of high degrees, dual-PRS method fails to give any significant result. □

Fortunately, we can convert ill-conditioned polynomials to well-conditioned ones, as follows.

We assume that every root of $f(x) = 0$ is bigger than 1 (if f does not satisfy this condition, we apply Greaffe root-squaring technique [7]). Then, we have $|f_m| \le |f_{m-1}| \le \cdots \le |f_0|$. Put $\sigma_f = \max_i\{(m-i)|f_i|/|f_m|\}$, then we have

$$|\mathrm{lc}(f(\sigma_f x))|, |f_0| \approx ||f(\sigma_f x)||.$$

For given polynomials f and g, put $\sigma = \max\{\sigma_f, \sigma_g\}$, and we convert f and g into $\hat{f}$ and $\hat{g}$ as $\hat{f} = f(\sigma x)$ and $\hat{g} = g(\sigma x)$, respectively. Then, the resulting polynomials are required ones.

3. RELATIONSHIP BETWEEN DUAL-PRS METHOD AND ZHI'S METHOD

In this section, we consider the relationship between the dual-PRS method and Zhi's method [37]. Zhi's method is based on a fast QR-decomposition of structured matrix and the generalized Schur decomposition [11, 5] of M_5 in $\mathbb{F}^{2N\times 2N}$, where $N = m+n$, the sum of degrees of given polynomials [11]; for M_5, see **3.1**. The cost of Zhi's method is $O(5N^2)$. In **3.1**, we explain a fast QR-decomposition of the Sylvester matrix briefly, that is used in Zhi's method. In **3.2**, we show the relationship between two methods.

3.1 Fast QR-decomposition of the Sylvester matrix

Let M_5 be a symmetric Toeplitz matrix, expressed as

$$M_5 = \begin{pmatrix} S^TS & S^T \\ S & \boldsymbol{O} \end{pmatrix} \in \mathbb{F}^{2N\times 2N}.$$

Here, $S = S(f,g)$ is the Sylvester matrix of $f(x)$ and $g(x)$, expressed as

$$S(f,g) = \begin{pmatrix} f_m & & & g_n & & \\ f_{m-1} & f_m & & g_{n-1} & g_n & \\ \vdots & f_{m-1} & \ddots & \vdots & g_{n-1} & \ddots \\ & \vdots & \ddots & & \vdots & \ddots \end{pmatrix} \in \mathbb{F}^{N\times N}.$$

Let F be the following lower shift square matrix:

$$F = \begin{pmatrix} 0 & & & \\ 1 & 0 & & \\ & \ddots & \ddots & \\ & & 1 & 0 \end{pmatrix}.$$

The displacement of M_5 w.r.t. F is denoted by $\nabla_F(M_5)$, and defined to be $\nabla_F(M_5) = M_5 - FM_5F^T = GJG^T$, where G is a $2N\times 5$ matrix and J is a signature matrix of the form

$$G = \begin{pmatrix} \boldsymbol{f} & \boldsymbol{g} & \tilde{\boldsymbol{f}} & \tilde{\boldsymbol{g}} & \boldsymbol{0} \\ \boldsymbol{e}_1 & \boldsymbol{e}_{n+1} & \boldsymbol{e}_{n+1} & \boldsymbol{0} & \boldsymbol{e}_1 \end{pmatrix}, \quad J = \begin{pmatrix} I_2 & \\ & -I_3 \end{pmatrix}.$$

Here,
$\boldsymbol{f} = (f_m, \cdots, f_0, 0, \cdots, 0)^T$, $\tilde{\boldsymbol{f}} = (0, \cdots, 0, f_m, \cdots, f_1)^T$,
$\boldsymbol{g} = (g_n, \cdots, g_0, 0, \cdots, 0)^T, \tilde{\boldsymbol{g}} = (0, \cdots, 0, g_n, \cdots, g_1)^T$,
and each $\boldsymbol{e}_i = (0, \ldots, \overset{i}{1}, 0, \ldots)$ is the standard basis in $\mathbb{F}^N$. Let $S = QR$ be the QR-decomposition of S. A fast QR-decomposition of S is done as follows, where Θ_i below is a Givens matrix.

Algorithm 2 (Zhi's method [37]).

```
Input:   $f$ and $g$
Output:  Matrix $L_N = \binom{R^T}{Q} \in \mathbb{F}^{2N\times N}$ and appgcd$(f,g)$.
Step 0:  Put $i := 1$ and $G_1 := G$.
Step 1:  Convert $G_i$ to *proper form* w.r.t. the first column,
         and put $\bar{G}_i = G_i\Theta_i$ [5].
Step 2:  Put $\bar{\boldsymbol{\ell}}_i = \bar{G}_i\binom{1}{\boldsymbol{0}} \in \mathbb{F}^{2N-i+1}$.
         Then, the $i$-th column of matrix $L_N$, denoted by
         $\boldsymbol{\ell}_i$, is $\boldsymbol{\ell}_i = \binom{\boldsymbol{0}}{\bar{\boldsymbol{\ell}}_i} \in \mathbb{F}^{2N}$.
Step 3:  Let $i := i + 1$.
         If $i < N$ then perform the conversion
         $\begin{pmatrix} \boldsymbol{0} \\ G_i \end{pmatrix} := \begin{pmatrix} F\boldsymbol{\ell}_{i-1} & \bar{G}_{i-1}\begin{pmatrix} 0 & \\ & I \end{pmatrix} \end{pmatrix}$,
         using Givens and hyperbolic rotations, and House-
         holder transformation,
         goto Step 1.
         If $i = N$, we have the appGCD of $f(x)$ and $g(x)$.
```

3.2 Rewriting Zhi's method using double-side reductions

We note that the above vectors $\boldsymbol{f}$ and $\boldsymbol{g}$ in $\mathbb{F}^N$ correspond to polynomials $x^{n-1}f$ and $x^{m-1}g$ in $\mathbb{F}[x]$, respectively. Put $p_1 = x^{n-1}f$ and $p_2 = x^{m-1}g$, then $p_1, p_2 \in \mathbb{F}[x]$ and $\deg(p_1) = \deg(p_2) = m + n - 1$.

For $i = 1, \ldots, n-1$ in Algorithm 2

Steps 1, 2, and 3 as a whole perform the transformation from (p_{2i-1}, p_{2i}) to (p_{2i+1}, p_{2i+2}) satisfying $\deg(p_{2i-j}) < \deg(p_{2(i+1)-j})$ for $j = 1, 2$.

Let $\Theta_i^{(0)}(x) \in \mathbb{F}[x, x^{-1}]^{2\times 2}$ be a matrix, obtained by applying the Givens rotation $\Theta_i^{(0)} \in \mathbb{F}^{2\times 2}$ in Algorithm 2. By this operation, p_{2i+1} and p_{2i+2} are obtained by eliminating the leading coefficient of p_{2i-1} and the constant term of p_{2i}, simultaneously, as follows.

$$\begin{aligned}\begin{pmatrix} p_{2i+1} \\ p_{2i+2} \end{pmatrix} &= \frac{1}{x}\Theta_i^{(0)}(x) \begin{pmatrix} p_{2i-1} \\ p_{2i} \end{pmatrix} \\ &= \frac{1}{\sqrt{1+|\gamma_i^{(0)}|^2}} \begin{pmatrix} 1 & -\gamma_i^{(0)} \\ \gamma_i^{(0)}/x & 1/x \end{pmatrix} \begin{pmatrix} p_{2i-1} \\ p_{2i} \end{pmatrix},\end{aligned}$$

where $\gamma_i^{(0)} = \mathrm{lc}(p_{2i})/\mathrm{lc}(p_{2i-1}) \in \mathbb{F}$. This operation is the same as the double-side reduction. After i rotations by $\Theta_1^{(0)}(x), \ldots, \Theta_i^{(0)}(x)$, we obtain (p_{2i+1}, p_{2i+2}) expressed as follows.

$$\begin{aligned}\begin{pmatrix} p_{2i+1} \\ p_{2i+2} \end{pmatrix} &= \frac{1}{x^i}\Theta_i^{(0)}(x) \cdots \Theta_1^{(0)}(x) \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} \\ &= \frac{1}{x^i} \begin{pmatrix} a_{2i+1} & b_{2i+1} \\ a_{2i+2} & b_{2i+2} \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix},\end{aligned}$$

where a_{2i+1}, a_{2i+2} and b_{2i+1}, b_{2i+2} are polynomials satisfying $\deg(a_{2i+j}), \deg(b_{2i+j}) \le i$ $(j = 1, 2)$. Note that

$$\begin{aligned} a_{2i+2}(x) &= -x^{\deg(b_{2i+1}+1)} b_{2i+1}(1/x), \\ b_{2i+2}(x) &= x^{\deg(a_{2i+1}+1)} a_{2i+1}(1/x). \end{aligned}$$

Here, p_i/x is a polynomial in $\mathbb{F}[x]$ for $1 \le i \le n$, however, p_i/x is not a polynomial for $i > n$.

For $i > n$ in Algorithm 2

Put $P_1 = p_{2n-1}$ and $P_2 = p_{2n}$, then we have $\deg(P_{2i-1}) = \deg(P_{2i})$ and $\gcd(P_{2i-1}, P_{2i}) = \gcd(f, g)$. Suppose that we computed (P_{2i-1}, P_{2i}) by Algorithm 2, then we generate (P_{2i+1}, P_{2i+2}) satisfying $\deg(P_{2i+1}) = \deg(P_{2i+2}) < \deg(P_{2i})$ and $\gcd(P_{2i+1}, P_{2i+2}) = \gcd(f, g)$, as follows. First we compute $(\breve{P}_{2i+1}, \breve{P}_{2i+2})^T = \Theta_i^{(1)}(P_{2i-1}, P_{2i})^T$. We set $P_{2i+2} = \breve{P}_{2i+2}$, then $\gcd(f, g)$ divides $\breve{P}_{2i+2}$. However, $\deg(P_{2i+2}) < \deg(\breve{P}_{2i+1})$.

Polynomials $\tilde{f} = (f - f_0)/x$ and $\tilde{g} = (g - g_0)/x$ correspond to vectors $\tilde{\boldsymbol{f}} = F^n\boldsymbol{f}$ and $\tilde{\boldsymbol{g}} = F^m\boldsymbol{g}$, respectively. Let $\psi : \mathbb{F}[x] \to \mathbb{F}[x]$ be a shift mapping such that

$$\breve{P}(x) \mapsto \frac{\breve{P}(x) - \breve{P}(0)}{x}.$$

Then $\gcd(f, g)$ does not divide $\psi(\breve{P}_{2i+1})$. So, we transform $\psi(\breve{P}_{2i+1})$ to P_{2i+1} satisfying $\gcd(f, g) | P_{2i+1}$ as follows; note that $\tilde{f}_1 = \tilde{f}$ and $\tilde{g}_1 = \tilde{g}$.

1. Reduce $\tilde{f}_i$ by $\tilde{g}_i$ using the Givens rotation matrix $\Theta_i^{(2)} \in \mathbb{F}^{2\times 2}$.

$$\begin{aligned}\Theta_i^{(2)} \begin{pmatrix} \tilde{f}_i \\ \tilde{g}_i \end{pmatrix} &= \frac{1}{\sqrt{1+|\gamma_i^{(2)}|}} \begin{pmatrix} -\gamma_i^{(2)} & 1 \\ 1 & \gamma_i^{(2)} \end{pmatrix} \begin{pmatrix} \tilde{f}_i \\ \tilde{g}_i \end{pmatrix} \\ &= \begin{pmatrix} \breve{f}_{i+1} \\ \breve{g}_{i+1} \end{pmatrix} = \begin{pmatrix} \tilde{f}_{i+1} \\ \breve{g}_{i+1} \end{pmatrix},\end{aligned}$$

where $\gamma_1^{(2)} = \mathrm{lc}(\tilde{g}_i)/\mathrm{lc}(\tilde{f}_i)$.

2. Reduce $\breve{g}_{i+1}$ by $\psi(\breve{P}_{2i+1})$ using the hyperbolic rotation matrix $\Phi_i \in \mathbb{F}^{2\times 2}$.

$$\begin{aligned}&\Phi_i \begin{pmatrix} \psi(\breve{P}_{2i+1}) \\ \breve{g}_{i+1} \end{pmatrix} \\ &= \frac{1}{\sqrt{1-|\rho_i|^2}} \begin{pmatrix} 1 & -\rho_i \\ -\rho_i & 1 \end{pmatrix} \begin{pmatrix} \psi(\breve{P}_{2i+1}) \\ \breve{g}_{i+1} \end{pmatrix} \\ &= \begin{pmatrix} P_{2i+1} \\ \tilde{g}_{i+1} \end{pmatrix},\end{aligned}$$

where $\rho_i = \mathrm{lc}(\breve{g}_{i+1})/\mathrm{lc}(\psi(\breve{P}_{2i+1}))$. By [37] or [11, 5], we set that P_{2i+1} satisfies either $\gcd(f, g)|P_{2i+1}$ or $P_{2i+1} = 0$.

Thus, we obtain (P_{2i+1}, P_{2i+2}) with $\deg(P_{2i+1}) < \deg(P_{2i-1})$ and $\deg(P_{2i+2}) < \deg(P_{2i})$. The operations mentioned above can be expressed as

$$\begin{pmatrix} P_{2i+1} \\ P_{2i+2} \\ \tilde{f}_{i+1} \\ \tilde{g}_{i+1} \\ 1 \end{pmatrix} = Q_{\mathrm{Zhi}}^{(i)} \cdot \begin{pmatrix} P_{2i-1} \\ P_{2i} \\ \tilde{f}_i \\ \tilde{g}_i \\ 1 \end{pmatrix}.$$

Here, a matrix $Q_{\mathrm{Zhi}}^{(i)}$, in $\mathbb{F}[x, x^{-1}]^{5\times 5}$, is represented as

$$c_{i+1,i} \left(\begin{array}{cccc|c} \frac{1}{x} & \frac{\gamma_i^{(1)}}{x} & \rho_i & \rho_i\gamma_i^{(2)} & -\frac{\breve{p}_{2i+1,0}}{x} \\ -\gamma_i^{(1)} & 1 & & & \\ & & -\gamma_i^{(2)} & 1 & \\ -\frac{\rho_i}{x} & -\frac{\rho_i\gamma_i^{(1)}}{x} & 1 & \gamma_i^{(2)} & \frac{\breve{p}_{2i+1,0}\,\rho_i}{x} \\ \hline & & & & 1 \end{array}\right) \tag{3.1}$$

where $c_{i+1,i} \in \mathbb{F}\backslash\{0\}$. Therefore, formulating Zhi's method as above, it becomes similar to the dual-PRS method. However, it is hard to treat the above matrix since the size is large and every element is in $\mathbb{F}[x^{-1}]$.

4. HALF-GCD METHOD WITH DOUBLE-SIDE REDUCTION

The half-GCD method is a fast method using the Euclidean elimination, the cost is $O(\log^2(m) \cdot m)$ [16]. However, the Euclidean elimination with floating-point numbers is unstable [32]. In order to stabilize the half-GCD method with floating-point numbers we employ the double-side reduction. In the Euclidean method, the remainder sequence is computed to obtain GCD. On the other hand, in the half-GCD method, GCD is computed by Schönhage's 2×2 quotient matrix [16]. For polynomials with integer coefficients, the quotient matrix $Q_{\rm int}$ is defined to satisfy

$$\begin{pmatrix} g \\ r \end{pmatrix} = Q_{\rm int} \begin{pmatrix} f \\ g \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -q(x) \end{pmatrix} \begin{pmatrix} f \\ g \end{pmatrix}, \qquad (4.1)$$

where $q(x) = \mathrm{quo}(f, g)$ is a polynomial. To incorporate the half-GCD method for polynomials with floating-point number coefficients, we construct the quotient matrix as follows.

$$Q_{\rm fp} = \frac{1}{x} \begin{pmatrix} x & -\gamma\, x^{d+1} \\ -\omega & 1 \end{pmatrix} \in \mathbb{F}[x, x^{-1}]^{2\times 2}. \qquad (4.2)$$

That is, we compute the quotient matrix by the double-side reduction, so as to stabilize the GCD computation. This is our key idea.

4.1 Divide-and-conquer technique

We can split polynomial f as follows.

$$\begin{aligned} f &= f^{(1)}x^{k_1} + f^{(0)}, \quad k_1 = \lceil \deg(f)/2 \rceil \\ &= (f^{(11)}x^{k_2} + f^{(10)})x^{k_1} + (f^{(01)}x^{k_2'} + f^{(00)}) \\ &= [(f^{(111)}x^{k_3} + f^{(110)})x^{k_2} + f^{(10)}]x^{k_1} \\ &\quad + [f^{(01)}x^{k_2'} + (f^{(001)}x^{k_3'} + f^{(000)})] \\ &= \cdots. \end{aligned}$$

In computing the quotient matrix $Q_{\rm fp}$, we compute numbers γ and w in (4.2) first by $f^{(11)}$, $g^{(11)}$, $f^{(00)}$ and $g^{(00)}$, then by $f^{(111)}$, $g^{(111)}$, $f^{(000)}$ and $g^{(000)}$, and so on. Thus, we perform the half-GCD method with floating-point number coefficients as follows. Below, we assume that f and g are split as $f = f_h x^{k_1} + f_\ell$ and $g = g_h x^{k_1} + g_\ell$, respectively.

Algorithm 3 (Half-GCD method; HGCD).

```
Input:  $f_h, f_\ell, g_h, g_\ell$
        with $\deg(f_h) = \deg(g_h) \le \deg(f_\ell) = \deg(g_\ell)$.
Steps:  if $\deg(f_h) = 0$ then return a matrix in (4.2)
        else
            split $f_h$ and $f_\ell$ as
            $f_h = f_{h,1}x^k + f_{h,0}$, $f_\ell = f_{\ell,1}x^k + f_{\ell,0}$,
            and split $g_h$ and $g_\ell$ as
            $g_h = g_{h,1}x^k + g_{h,0}$, $g_\ell = g_{\ell,1}x^k + g_{\ell,0}$,
            where $k = \lceil \deg(f_h)/2 \rceil$.
            $R_1 = \mathrm{HGCD}(f_{h,1}, g_{h,1}, f_{\ell,0}, g_{\ell,0})$;
            $(q_1, q_2)^T = R_1(f_h, g_h)$ and $(r_1, r_2)^T = R_1(f_{\ell,0}, g_{\ell,0})$
            $R_2 = \mathrm{HGCD}(q_1, q_2, r_1, r_2)$;
            return $R_2 \cdot R_1$;
        end;;
```

Example 3. (Computing a half-GCD)

$$\begin{aligned} f(x) &= (x^2+1)(x^5+2x^3-1) \\ &= x^7 + \#\mathrm{E}[3.0, 3.0\,\mathrm{e}{-15}]x^5 + \#\mathrm{E}[2.0, 2.0\,\mathrm{e}{-15}]x^3 \\ &\quad - x^2 - 1, \\ g(x) &= (x^2+1)(0.001x^5 + x^2 + 1) \\ &= \#\mathrm{E}[0.001, 1.0\,\mathrm{e}{-18}]x^7 + \#\mathrm{E}[0.001, 1.0\,\mathrm{e}{-18}]x^5 \\ &\quad + x^4 + \#\mathrm{E}[2.0, 2.0\,\mathrm{e}{-15}]x^2 + 1. \end{aligned}$$

Applying the half-GCD method, we obtained a matrix $(m_{i,j})/x^3$, $1 \le i, j \le 2$, where

$$\begin{aligned} m_{11} &= x^4 - \#\mathrm{E}[1.99\cdots, 2.99\,\mathrm{e}{-15}]x^2 \\ &\quad + \#\mathrm{E}[999.\cdots, 9.99\,\mathrm{e}{-13}]x, \\ m_{12} &= \#\mathrm{E}[1000.0, 1.0\,\mathrm{e}{-12}]x^4 \\ &\quad - \#\mathrm{E}[1.99\cdots, 2.99\,\mathrm{e}{-15}]x^2 \\ &\quad + \#\mathrm{E}[999.\cdots, 9.99\,\mathrm{e}{-13}]x, \\ m_{21} &= x^3 - \#\mathrm{E}[1.99\cdots, 2.99\,\mathrm{e}{-15}]x, \\ m_{22} &= \#\mathrm{E}[1000.0, 1.0\,\mathrm{e}{-12}]x^3 \\ &\quad - \#\mathrm{E}[1.99\cdots, 2.99\,\mathrm{e}{-15}]x. \end{aligned}$$

Remark 4 (Applicability of half-GCD mehotd). The computation of quotient matrix $Q_{\rm fp}$ cannot be done efficiently for the stabilized PRS method [32], its improved version [22] and Zhi's method [37]. The reason is as follows.

- Original stabilized PRS method
  Suppose we have computed P_3 according to formula (2.3). After this, in order to avoid the self-reduction, we check the smallness of $|\mathrm{lc}(P_3)/\mathrm{lc}(P_1)|$ again.

  If $|\mathrm{lc}(P_3)/\mathrm{lc}(P_1)|$ is small, we compute P_4 from P_1 and P_3 as in **2.1**. Therefore, we cannot determine the quotient $q(x)$ for $\mathrm{rem}(P_2, P_3)$ because we do not compute the remainder. That is, we cannot apply half-GCD technique to stabilized PRS method directly.

- Its improved version
  This method is close to the QRGCD method but it is still close to the stabilized PRS method. Hence, half-GCD technique is not directly applicable to the improved version.

- Zhi's method
  Since the size of quotient matrix is 5×5, as expressed in (3.1), half-GCD technique is not directly applicable to Zhi's method.

4.2 Complexity

The complexity of half-GCD method for polynomials in $\mathbb{F}[x]$ is estimated as follows.

Lemma 1 (Complexity of half-GCD mehtod). *The cost $T(m)$ of half-GCD satisfies the inequality*

$$T(m) \le 2 \cdot T(\frac{m}{2}) + cM(m). \qquad (4.3)$$

Here, $m = \max\{\deg(f), \deg(g)\}$, $M(n)$ is the cost of multiplication of polynomials with degree m, and c is a constant.

Proof. In our method, we carry out two recursive calls of HGCD for polynomials of degree $m/2$ and several multiplication of 2×2 matrices whose entries are polynomials of degree m. Hence, the lemma holds. □

From lemma 1, we obtain the cost of HGCD as follows.

Theorem 2 (Complexity). *The cost $T(m)$ is bounded as follows.*

$$T(m) \le cM(m)\log(m). \qquad (4.4)$$

5. FAST LU-DECOMPOSITION USING GKO METHOD

The GKO (Gohberg-Kailath-Olshevsky) method is a fast LU-decomposition of the Cauchy-like matrix, and the cost is $O(n^2)$ for the $n \times n$ matrix. In this section, we explain the LU-decomposition of the Cauchy-like matrix and of the Toeplitz-like matrix [3], respectively. Bini-Boito proposed to compute the appGCD by utilizing the GKO method [3]. We explain the GKO method first.

Let C_1 be the Cauchy-like matrix of order n, expressed as

$$C_1 = (c_{i,j}^{(1)})_{i,j} = \Big(\frac{\boldsymbol{g}_i^{(1)} \cdot \boldsymbol{b}_j^{(1)}}{f_i - a_j}\Big)_{i,j}.$$

Here, $\boldsymbol{g}_i^{(1)} = (g_{i,1}^{(1)}, g_{i,2}^{(1)}) \in \mathbb{F}^{1\times 2}$ and $\boldsymbol{b}_j^{(0)} = (b_{1,j}^{(1)}, b_{2,j}^{(1)})^T \in \mathbb{F}^{2\times 1}$ for $1 \le i,j \le n$. Put $\nabla_{F_1,A_1}(C_1) = F_1C_1 - CA_1$, where $F_1 = \mathrm{diag}(f_1,\dots,f_n)$, and $A_1 = \mathrm{diag}(a_1,\dots,a_n)$, $\nabla_{F_1,A_1}(C_1)$ can be decomposed as $\nabla_{F_1,A_1}(C_1) = G_1B_1$, where $G_1 = (\boldsymbol{g}_1,\dots,\boldsymbol{g}_n)^T \in \mathbb{F}^{n\times 2}$ and $B_1 = (\boldsymbol{b}_1,\dots,\boldsymbol{b}_n) \in \mathbb{F}^{2\times n}$. Applying the GKO method, the LU-decomposition of C_1, i.e., $C_1 = L_1U_1$, is done as follows.

1. Construct $G_2 = ({\boldsymbol{g}_1^{(2)}}^T,\dots,{\boldsymbol{g}_{n-1}^{(2)}}^T)^T \in \mathbb{F}^{(n-1)\times 2}$ and $B_2 = ({\boldsymbol{b}_1^{(2)}}^T,\dots,{\boldsymbol{b}_{n-1}^{(2)}}^T) \in \mathbb{F}^{2\times(n-1)}$ as

$$\begin{pmatrix} 0 \\ G_2 \end{pmatrix} = G_1 - \frac{1}{c_{1,1}^{(1)}} \begin{pmatrix} c_{1,1}^{(1)} \\ \vdots \\ c_{n,1}^{(1)} \end{pmatrix} \boldsymbol{g}_1^{(1)}, \qquad (5.1)$$

$$\begin{pmatrix} 0 & B_2 \end{pmatrix} = B_1 - \frac{1}{c_{1,1}^{(1)}} \boldsymbol{b}_1^{(1)} \big(c_{1,1}^{(1)},\cdots,c_{1,n}^{(1)}\big). \qquad (5.2)$$

   Then, the 1st row of L_1 is $1/c_{1,1}^{(1)}(c_{1,1}^{(1)},\dots,c_{n,1}^{(1)})^T$ and the 1st column of U_1 is $\big(c_{1,1}^{(1)},\cdots,c_{1,n}^{(1)}\big)$. The (G_2,B_2) is a generator-pair of the Schur complement C_2 of C_1; $\nabla_{F_2,A_2}(C_2) = F_2C_2 - C_2A_2 = G_2B_2$, where $F_2 = \mathrm{diag}(a_2,\dots,a_n)$ and $A_2 = \mathrm{diag}(a_2,\dots,a_n)$.

2. Continue the computation of generator-pair (G_i, B_i) recursively, until L and U are constructed. □

Note that the pivoting does not destroy the structure of the Cauchy-like matrix. Therefore, using the GKO method, we can perform the LU-decomposition with partial pivoting, and the cost is $O(n^2)$.

Next, we explain Bini-Boito's method. The Sylvester matrix $S(f,g)$ is not the Cauchy-like matrix but the Toeplitz-like matrix [3]. We can transform the Toeplitz-like matrix to the Cauchy-like matrix, as follows. We set matrices $\tilde{G}_1$ and $\tilde{B}_1$ as follows.

$$\tilde{G}_1 = \begin{pmatrix} \mathbf{0}_{m-1}^T & \mathbf{0}_{m+n-1}^T \\ 1 & \\ \mathbf{0}_n & 1 \end{pmatrix},$$

$$\tilde{B}_1 = \begin{pmatrix} -f_{m-1}, & \dots, & -f_1, -f_0+g_n, & g_{n-1}, & \dots, & g_1, & g_0+f_m \\ -g_{n-1}, & \dots, & -g_1, -g_0+f_m, & f_{m-1}, & \dots, & f_1, & f_0+g_n \end{pmatrix}.$$

Put $\mathcal{F} = \dfrac{1}{\sqrt{n}}\Big(e^{2\pi\hat{1}/2\times(i-1)(j-1)}\Big)_{i,j}$ and

$$\begin{aligned} D_0 &= \mathrm{diag}(1, e^{\frac{2\pi\hat{1}}{n}},\cdots,e^{\frac{2(n-1)\pi\hat{1}}{n}}),\\ D_1 &= \mathrm{diag}(e^{\frac{\pi\hat{1}}{n}}, e^{\frac{3\pi\hat{1}}{n}},\cdots,e^{\frac{(2n-1)\pi\hat{1}}{2n}}),\\ D_{-1} &= \mathrm{diag}(1, e^{\frac{\pi\hat{1}}{n}},\cdots,e^{\frac{(2n-1)\pi\hat{1}}{2n}}). \end{aligned}$$

We transform $\tilde{G}_1$ and $\tilde{B}_1$ to $G_1 = \mathcal{F}\tilde{G}_1$ and $B_1^H = \mathcal{F}D_0\tilde{B}_1^H$, then the product G_1B_1 is the displacement of the Cauchy-like matrix. The cost of transformation is $O(n^2)$. Therefore, we can employ the GKO method to compute the appGCD. The cost is $O(m^2)$.

Algorithm 4 (Bini-Boito's method [3]).

1. Determine the degree, let is be k, of the approximate GCD; k can be determined by the rank of the Sylvester matrix of given polynomials f and g.
2. Compute the GCD and the cofactors f/c and g/c of f and g. Actually, they are computed as follows.
   - Method 1: use the Sylvester matrix.
     Compute the null-space of subresultant matrix; cofactor corresponds to the null-space of subresultant matrix.
   - Method 2: use the Bezout matrix.
     the GCD is computed by utilizing Barnett's theorem. See [3] in details.
3. Refinement (optional) [36, 3]

Unfortunately, we cannot incorporate the half-GCD method with the GKO method. The reason is so follows. In the GKO method, pivoting iscrutically important and the pivoting is made by scannnig the whole matrix.

6. EXPERIMENTS

We have implemented the dual-PRS method and half-GCD method for floating-point number coefficients in Maple. The following timing data (in milli-seconds) were obtained on Core2Duo running at 2.4GHz under WindowsXp. Core2Duo is a dual-core CPU of Pentium, but actually we tested on single-core CPU only, which is equivalent to Pentium running at 2.4GHz.

We have tested appgcd(f_i, g_i), where f_i and g_i $(i = 1,\dots,6)$ are polynomials with coefficients of fixed-precisions floating-point numbers of $\varepsilon_{\mathrm{M}} = 10^{-10}$ and 10^{-32}, respectively. The computation with floating-point numbers depends on which kind of floating-point numbers are used. So, we have carried out the computation by both the hardware arithmetics when $\varepsilon_{\mathrm{M}} = 10^{-10}$ (default precision in Maple) and the software arithmetics. We generated polynomials f_i, g_i by Maple as follows.

```
> d[1] :=  10;  d[2] :=  20; d[3] :=  50;
  d[4] := 100;  d[5] := 200; d[6] :=1000;

> c:=x^2 + x + 1:                    # fix the GCD
> for i from 1 to 6 do
    for j from 1 to 10 do
      ff[i,j]:=randpoly(x,dense,degree=d[i]-2):
      gg[i,j]:=randpoly(x,dense,degree=d[i]-2):
```

```
      f[i,j]:=expand(c*ff[i]):    # given poly. f
      g[i,j]:=expand(c*gg[i]):    # given poly. g
    end do:
  end do:
```

- $i = 1$: $\deg(f) = \deg(g) = 10$
- $i = 2$: $\deg(f) = \deg(g) = 20$
- $i = 3$: $\deg(f) = \deg(g) = 50$
- $i = 4$: $\deg(f) = \deg(g) = 100$
- $i = 5$: $\deg(f) = \deg(g) = 200$
- $i = 6$: $\deg(f) = \deg(g) = 1000$

Tables below show the average times and the accuracies for each method; the accuracy is estimated by using efloat numbers.

6.1 Comparison of dual-PRS with QRGCD methods

We first show test of dual-PRS method by computing it with QRGCD method.

Table 1 shows the result of experiment on hardware arithmetics with `Digits=10` ($\varepsilon_{\mathrm{M}} = 10^{-10}$).

	Dual-PRS		QRGCD	
i	Ave.CPU	Error	Ave.CPU	Error
1	0.125	e-10	0.036	e-10
2	0.191	e-10	0.040	e-10
3	0.434	e-9	0.106	e-10
4	1.022	e-8	0.347	e-7
5	3.548	e-7	1.266	e-6

Table 1: hardware arithmetics: `Digits=10`

Although, the timing data on the dual-PRS method show $O(m^2)$ behavior, those on the QRGCD method do not show $O(m^3)$ behavior but show $O(m^2)$ behavior; the QRGCD method is abnormally fast at this precision [32].

Table 2 shows the result of experiment on software arithmetics with `Digits=32`.

	Dual-PRS		QRGCD	
i	Ave.CPU	Error	Ave.CPU	Error
1	0.105	e-31	0.100	e-31
2	0.237	e-31	0.443	e-31
3	0.451	e-30	6.006	e-30
4	1.234	e-29	about 80 sec.	e-26
5	3.734	e-28	over 600 sec.	—
6	103.365	e-20	—	—

Table 2: software arithmetics: `Digits=32`

We see that the dual-PRS method is much faster than the QRGCD method. Furthermore, it is stabler than the QRGCD method. If we use 32-digits precision, the dual-PRS method can compute appGCD pretty safely for polynomials of degrees $\lesssim 200 \sim 1000$, so long as we use higher precision. On the other hand, applying the stabilized PRS method, we cannot compute the appGCD for polynomials of degrees $\gtrsim 50$ [32].

6.2 Comparison of dual-PRS method with half-GCD method

Next, we show a test of half-GCD method. In the half-GCD method with floating-point numbers, we use the FFT-based multiplication; in order to attain higher efficiency in the divide-and-conquer methods, fast multiplication is necessary.

The result is shown in Table 3, where the fast multiplication was used only for $[i = 6]$ case, i.e., for polynomials of degree 1000. For other cases, old-fashioned multiplication seems to be not bad.

	Dual-PRS		HGCD	
i	Ave.CPU	Error	Ave.CPU	Error
1	0.105	e-31	0.195	e-31
2	0.237	e-31	0.403	e-31
3	0.451	e-30	1.417	e-30
4	1.234	e-29	3.091	e-29
5	3.734	e-28	7.961	e-28
6	103.365	e-20	87.423	e-22

Table 3: software arithmetics: `Digits=32`

Table 3 shows that the dual-PRS method is faster than the half-GCD method, except for degree 1000.

7. CONCLUSION

In this paper, we proposed two methods.

One method is the dual-PRS method, devised by considering Zhi's method. In [32, 18], authors pointed out that there is a relationship between the PRS method and the QR-decomposition of Sylvester matrix, i.e., the QRGCD method. This means that the PRS method can be rewritten by QRGCD method. In this paper, we showed a relationship between the dual-PRS method and a fast QR-decomposition of the Sylvester matrix, i.e., Zhi's method. For univariate polynomials, the dual-PRS method works efficiently and stably for polynomials of degree $\lesssim 200 \sim 1000$ so long as we employ higher precisions.

Another method is the half-GCD method. In order to stabilize the divide-and-conquer computation, we utilize the dual-PRS technique in this method, too.

Both methods are fast and stable, because the reduction and the pivoting are carried out simultaneously. However, our current versions are very crude, and substantial improvement are required.

Acknowledgments

The author thanks very much Prof. Tateaki Sasaki for supervising this work and various advices.

8. REFERENCES

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman. The design and analysis of computer algorithms. Addison-Wesley, 1974.

[2] S. Belhaj. Block factorization of Hankel matrices and Euclidean algorithm. *Math. Medel. Nat. Phenom.* **5(7)**, 2010, 48–54.

[3] D. Bini and P. Boito. Structured matrix-based methods for polynomial ϵ-gcd: analysis and

comparisons. *Proc. of ISSAC'07*, ACM Press, 2007, 9–16.

[4] B. Beckermann and G. Labahn. A fast and numerically stable Euclidean-like algorithm for detecting relatively prime numerical polynomials. *J. Symb. Comput.*, **26** (1998), 691–714.

[5] S. Chandrasekaran and A. H. Sayed. A fast stable for nonsymmetric Toeplitz and quasi-Toeplitz systems of linear equations. *SIAM J. Matrix Anal. Appl.*, **19 (1)** (1998), 107–139.

[6] R. Corless, P. Gianni, B. Trager and S. Watt. The singular value decomposition for polynomial systems. *Proc. of ISSAC'95*, ACM Press, 1995, 195–207.

[7] R. Corless, S. Watt and L. Zhi. *QR* factoring to compute the GCD of univariate approximate polynomials. *IEEE Trans. Signal Proces.*, **52(12)** (2004), 3394–3402.

[8] G. M. Diaz-Toca and L. Gonzalez-Vega. Computing greatest common divisors and squarefree decompositions through matrix methods: The parametric and approximate cases, *Linear Algebra Appl.*, **412(2-3)**, (2006), 222–246.

[9] I. Emiris, A. Galligo and H. Lombardi. Certified approximate univariate GCDs. *J. Pure and Applied Alge.*, **117&118** (1997), 229–251.

[10] I. Gohberg, T. Kailath and Y. Olshevsky. Fast Gaussian elimination with partial pivoting for matrices with displacement structure. *Math. Comp.*, **64(212)**, 1996, 1557–1576.

[11] T. Kailath and A. H. Sayed. Displacement structute: theory and applications. *SIAM Review*, **37 (3)**, 1995, 297–386.

[12] F. Kako and T. Sasaki. Proposal of "effective floating-point number" for approximate algebraic computation. *Preprint of Tsukuba Univ.*, 1997.

[13] E. Kaltofen, Z. Yang and L. Zhi. Structured low rank approximation of a Sylvester matrix. *International Workshop on SNC'05*, D. Wang & L. Zhi (Eds.), 2005, 188–201; full paper appear in *Symbolic-Numeric Computation (Trends in Mathematics)*, D. Wang & L. Zhi (Eds.), Birkhäuser Verlag, 2007, 69–83.

[14] Z. Li, Z. Yang, and L. Zhi. Blind image deconvolution via fast approximate GCD *Proc. of ISSAC'10*, ACM Press, 2010, 155–162.

[15] M. Mitrouli and N. Karcanias. Computation of the gcd of polynomials using Gaussian transformation and shifting, Int. Journ. Control, **58** 1993, 211–228.

[16] R. T. Moenck. Fast computation of GCDs. *Proc. 5th ACM Symp. Thory of Comput.*, 1973, 142–151.

[17] K. Nagasaka. Ruppert matrix as subresultant mapping, *Proc. of CASC 2007*, Springer Berlin. 2007, 316–327.

[18] H. Ohsako, H. Sugiura and T. Torii. A stable extended algorithm for generating polynomial remainder sequence (in Japanese). *Trans. of JSIAM (Japan Society for Indus. Appl. Math.)* **7** (1997), 227–255.

[19] V. Pan. Univariate polynomials: nearly optimal algorithms for factorization and rootfinding. *Proc. of ISSAC'01*, ACM Press, 2001, 253–267.

[20] V. Pan and X. Yang. *Acceleration of Euclidean algorithm and extensions.* Proc. of ISSAC'02, ACM Press, 2002, 207–213.

[21] W. Qiub, Y. Huaa, and K. Abed-Meraima. A subspace method for the computation of the GCD of polynomials. Automatica, **33(4)** 1997, 741–743.

[22] M. Sanuki. Improvement of stabulized polynomial remainder sequence algorithm, submitted, 2009.

[23] T. Sasaki. A study of approximate polynomials, I — representation and arithmetic —. *Japan J. Indust. Appl. Math.* **12** (1995), 137–161.

[24] T. Sasaki. A practical method for floating-point Gröbner basis computation. *Proc. of ASCM 2009*, Fukuoka, Japan, Math-for-industry series, **22** 2009, 167–176.

[25] A. Schönhage. Quasi-GCD. *J. Complexity*, **1**, 1985, 118–147.

[26] M. Stewart. Stable pivoting for the fast factorization of Cauchy-like matrix. Preprint, 15 page, 1997.

[27] D. R. Sweet and R. P. Brent. Error analysis of a fast partial pivoting method for structured matrices. *Adv. Signal Proc. Algorithms, Proc. of SPIE*, T. Luk, ed., 2363 (1995), 266–280.

[28] T. Sasaki and M-T. Noda. Approximate square-free decomposition and root-finding of ill-conditioned algebraic equations. *J. Inform. Proces.*, **12** (1989), 159–168.

[29] T. Sasaki and A. Furukawa. Secondary polynomial remainder sequence and an extension of subresultant theory. *J. Inform. Proces.*, **7** (1984), 175–184.

[30] T. Sasaki and F. Kako. Computing floating-point Gröbner bases stably. Proc. of SNC'07, ACM Press, 2007, 180–189,

[31] T. Suzuki and T. Sasaki. On Sturm sequence with floating-point number coefficients. *RIMS Kokyuroku*, **685** (1989), 1–14.

[32] M. Sanuki and T. Sasaki. Computing approximate GCD in ill-conditioned cases. *Proc. of SNC'07*, ACM Press, 2007, 170–179.

[33] T. Sasaki and M. Sasaki. Polynomial remainder sequence and approximate GCD. *SIGSAM Buletin* **31**, 1997, 4–10.

[34] A. Terui. An iterative method for calculating approximate GCD of univariate polynomials. *Proc. of ISSAC'09*, ACM Press, 2009, 351–358.

[35] K. Thull and C. K. Yap. A unified approach to HGCD algorithms for polynomials and integers. Manuscript, Available from `http://cs.nyu.edu/cs/faculty/yap/papers/`.

[36] Z. Zeng. The approximate GCD of inexact polynomials part I: a univariate algorithm. *Preprint*, 2004.

[37] L. Zhi. Displacement structure in computing the approximate GCD of univariate polynomials. *Proc. of ASCM2003 (Asian Symposium on Computer Mathematics)*, World Scientific, 2003, 288–298.

[38] C. J. Zarowski, X. Ma and F. W. Fairman. QR-factorization method for computing the greatest common divisor of polynomials with inexact coefficients. *IEEE Trans. Signal Proces.*, **48(11)** (2000), 3042–3051.

Author Index