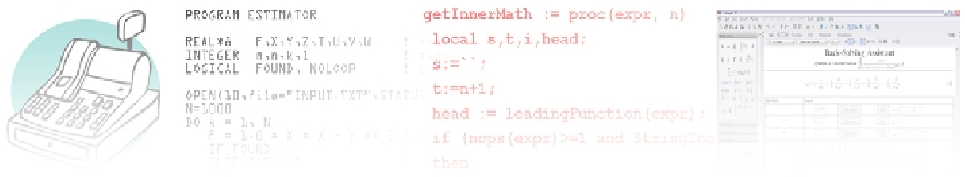


Milestones in Computer Algebra

MICA 2008:

A Conference in Honour of Keith Geddes' 60th Birthday

Stonehaven Bay, Trinidad and Tobago, 1-3 May 2008

http://www.orcca.on.ca/conferences/mica2008

In cooperation with ACM SIGSAM

Proceedings of MICA 2008: Milestones in Computer Algebra 2008
A Conference in Honour of Keith Geddes' 60th Birthday
Editors: Marc Moreno Maza and Stephen Watt
©MICA 2008
ISBN 978-0-7714-2682-7

Foreword to the MICA 2008 Conference

This conference honours the scientific career of Keith Geddes. Colleagues, students and friends will celebrate Professor Geddes' achievements in many areas: in fundamental research, in technology transfer, and in the training of the next generation of scientists, mathematicians and engineers.

Keith received his PhD in 1973 from the University of Toronto under the supervision of John C. Mason. Since that time, as a professor at the University of Waterloo, his research has spanned the areas of numerical approximation, algebraic algorithms for symbolic computation, hybrid symbolic-numeric computation and the design and implementation of computer algebra systems. Keith has actively supported our community through the ACM Special Interest Group on Symbolic and Algebraic Manipulation (SIGSAM), which he chaired from 1991 to 1993, and numerous conference program committees. He is perhaps best known as co-founder of the Maple computer algebra system. Through his teaching, research, service and software, the work of Keith Geddes has touched literally millions of individuals.

We are at a point that marks many numerically significant milestones: Keith was born just over 60 years ago in Saskatchewan, he began his research career just under 40 years ago as a graduate student at the University of Toronto, he co-founded Maplesoft 20 years ago in Waterloo and now he has chosen to start his retirement at the end of the current year. This is clearly an occasion that calls for celebration!

Almost four dozen scientific colleagues will come together at MICA 2008 to pay tribute to Keith. This includes eight distinguished invited speakers, some two dozen colleagues who have contributed scientific papers and posters, and others who come to pay their respects. In addition, a great many colleagues have sent their wishes, but cannot attend in person.

Many people have contributed time and effort to make this conference a success: The authors and speakers have prepared a host of high-quality contributions, the program committee has spent considerable effort in reviewing the submissions in a brisk time frame, the members of the organizing committee have all taken on additional responsibilities, and many student volunteers have helped with practical aspects. We thank Maplesoft for major funding for this meeting. Without this support the meeting could not take place in its present form. We thank the University of Waterloo for additional financial contribution. We also thank ACM SIGSAM, the Ontario Research Centre for Computer Algebra (ORCCA), the University of Western Ontario and the University of the West Indies for their support.

Everyone who knows Keith and has had the privilege to work with him will attest to his qualities as a scholar and person. On behalf of all who are participating in this conference, and all those who send their best wishes, we thank Keith for his many contributions, and wish him a rich and active retirement.

Stephen Watt	Mark Giesbrecht	Marc Moreno Maza
General Chair	Program Committee Chair	Proceedings Co-editor

Contents

Contributed posters

Macsyma: A Personal History

Joel Moses
Institute Professor
Professor of Computer Science and Engineering
Professor of Engineering Systems
MIT

Abstract
The Macsyma system arose out of research on mathematical software in the AI group at MIT in the 1960's. Algorithm development in symbolic integration and simplification arose out of the interest of people, such as the author, who were also mathematics students. The later development of algorithms for the GCD of sparse polynomials, for example, arose out of the needs of our user community. During various times in the 1970's the computer on which Macsyma ran was one of the most popular nodes on the ARPANET. We discuss the attempts in the late 70's and the 80's to develop Macsyma systems that ran on popular computer architectures. Finally, we discuss the impact of the fundamental ideas in Macsyma on current research on large scale engineering systems.

I entered MIT as a doctoral student in mathematics in 1963. My goal was to redesign the symbolic integration program by James Slagle that was done under the supervision of Marvin Minsky in 1961[1]. Minsky is one of the founders of the field of Artificial Intelligence. Slagle wrote his program, SAINT, Symbolic Automatic INTegrator, in LISP. While I initially wanted to use an assembler, I quickly became enamored of LISP due to its simplicity and its mathematical elegance. I did not realize then that my group and other groups would spend the next two decades improving LISP's speed and memory cost so that it rivaled that of popular languages, once declarations were made to program variables.

Actually, Minsky was unwilling to supervise another thesis in integration. He wanted his students to work on new applications of artificial intelligence, rather than improve old ones. My initial work thus was on proving that integration was undecidable. My idea was to use the recent result that proved the undecidability of Hilbert's tenth problem on polynomials with integer coefficients [2]. I believed that this result could be extended to integration problems in the calculus. After making significant progress on this problem I found out that Daniel Richardson had recently followed the same approach and proved the theorem, although he relied on the absolute value function toward the end of the proof, a step that it made the proof somewhat controversial [3]. Thus, in 1965 I was able to get Minsky to agree to my original goal. My program, SIN (a deliberate pun on Slagle's SAINT), was completed in 1967 [4]. In retrospect, I think I made several contributions in the thesis. A key one was in AI. I introduced, at about the same time as Stanford's Edward Feigenbaum [5], what later came to be known as the Knowledge Based Systems approach to AI. The original approaches to AI usually relied on searches of tree structures in order to solve problems. I argued that such searches could take enormous time as problems grew harder to solve. Instead, I felt that one should endow computers with knowledge of the problem domain so that searches could be eliminated, at least much of the time. The need for knowledge was accepted relatively quickly by leading AI researchers, such as Herb Simon. I assumed implicitly that the knowledge in the systems would be highly structured to make access easy. This turned out not be an easy sell, and it took me many years to figure out why. This issue is still a mainstay of my current work. An alternative to my approach that actually emphasized the use of unstructured knowledge was Feigenbaum's Rule-Based Expert Systems approach [6]. Much is to be learned from the initial successes of rule based expert systems in the 1970's and their later failures in the 1980's that led to the "AI winter."

A second contribution of my thesis was the overall structure of SIN which is composed of three stages. I used a heuristic, called the "derivative divides" heuristic, in the first stage of the SIN. This heuristic was to look for a component of the integrand whose derivative divides into the rest of the integrand leaving only a constant. If such a component existed, then a table look-up based on the form of the component resulted in the integral. Consider integrating x $\sin(x^2)$ with respect to x. The x^2 in $\sin(x^2)$ has derivative 2 x which divides the rest of the integrand (namely x) leaving only a constant 1/2. Looking up sin (y) in a small table in SIN yields – cos (y). Hence the integral is $-1/2 \cos(x^2)$. One could argue that in practice this heuristic solved about 80% of the problems that are posed.

The second stage of SIN uses a dozen or so methods specialized to the type of integrand. For example, rational functions of exponentials are handled by a method that attempts to integrate them by substituting a new variable, say y, for an exponential, often resulting in a rational function in y. I assumed that 80% of the remaining problems could be solved using the various algorithms in this second stage.

The final stage was based on my reading of the existing literature on integration, largely from Ritt's book on integration [7]. I originally developed a method, called the EDGE heuristic for EDucated GuEss. This approach assumed that the integral could be expressed as a sum of non-constant multiples of components in the integrand. The idea was to differentiate such a form and attempt to solve for the multiples. A few years later, when Risch's paper [8] was sent to me, I replaced this stage with Risch's algorithm in that paper, which is effective except for certain integrands that involved algebraic functions. Algebraic functions were known to be the sticking points in indefinite integration for a century. One can say that algebraic geometry was developed by Riemann and others in order to solve integration problems. The difficulty in solving such problems led to the conjecture by Hardy circa 1905 [9] that determining whether an integral can be expressed in terms of the usual functions of the calculus could not be decided in finite time and space. The irony, given Gödel's results, is that Hardy, who was courageous in going against Hilbert's view that all such decision problems were soluble, was essentially proved wrong.

This three stage approach to problem solving can be seen in other contexts these days. That is, a first stage that is relatively low cost, yet solves a high percentage of the problems; a second stage that requires identification of cases and possesses recipes for solving each case; and a third stage that involves much additional machinery. The book Re-engineering the Corporation [10] uses this approach, and it may not be an accident that both authors are associated with MIT. I believe that we will increasingly see such a three stage approach in health care. For example, the Minute Clinics that are becoming popular in the US can be considered such a first stage in a health care system.

The thesis had some, albeit limited impact on mathematical education. Thomas's famous calculus text had nearly a page describing it in some of the book's versions. It is interesting that Thomas's book has modules on Maple and Mathematica these days.

Finally, SIN was indeed faster and more powerful than SAINT as I had initially intended. In part, its power arose from the fact that I used the MATHLAB system for integrating rational functions. MATHLAB development was led by Carl Engelman of the MITRE Corporation. It too was written in LISP, and it used 19th century algorithms for factorization of polynomials that appeared in van der Waerden's books on abstract algebra [11].

A key idea in Risch's integration algorithm is the notion of field extensions. We assume that the base or ground field is the field of rational functions in x. Then e^x is an extension of the ground field which contains rational functions in e^x whose coefficients are rational functions in x. Similarly for log (x + 1). The function log (e^x + 1) can be placed in a field that involves two extensions of the rationals. The integration algorithm begins by expressing the integrand in some extension field of the rational functions and reduces the number of extensions at each step until it gets to the base field of rational functions. We generally get a set of linear equations which if solvable permit one to generate the integral. Otherwise, the problem cannot be integrated in terms of the usual functions of the calculus. The notion of field extensions is basic to modern pure mathematics in areas such as algebraic geometry. This notion played a key role in my thinking over the years. It is related to the notion of levels of abstraction in Computer Science.

In the years immediately following my thesis research I worked on a companion problem of simplification. The Edge heuristic as well as Risch's algorithm both emphasized the point that integration, when the integrand is carefully expressed, is the inverse of differentiation. My 1971 simplification paper [12] defined three theoretical approaches to simplification algorithms. *Zero-equivalence* algorithms guaranteed that expressions equivalent to 0 are recognized. Thus $\sin^2(x) + \cos^2(x) - 1$ would simplify to 0 using such algorithms. *Canonica*l algorithms would take an expression and reduce it to a canonical form. Thus equivalent expressions would result in the same form. Such an approach is not always ideal. For example, $(x+1)^{100}$ would result in a polynomial with 101 terms in most canonical polynomial systems, whereas it might be desirable to keep it in factored form in some situations. Risch's algorithm, which uses field extensions, produces what I called a *regular* simplification algorithm. The field extensions are algebraically independent. That is, they possess no relationship expressible in polynomial terms. For example, e^x and log(x+1) are algebraically independent. Regular simplifiers guarantee zero-equivalence but are not necessarily canonical since, for example, the order in which field extensions are chosen can yield somewhat different results.

Another student of Minsky in the 1963-1967 time frame was William Martin. Bill was trying to develop an interactive system that an engineer could use in solving a symbolic problem one step at a time. He developed a nice way to display expressions on a screen, as well as an interpreter for step-by-step symbolic solutions. The expression display used a separate machine, called the Kludge, which used a bit map display, and thus allowed Bill to generate two dimensional graphics of mathematical formulas. Bill finished his thesis a few months before I did in 1967[13]. We both stayed on at MIT after finishing our theses.

There were at that time several other groups working on symbolic systems and algorithms. They were brought together by Jean Sammet of IBM in a conference, called SYMSAM that she organized in 1966. Jean had also formed SICSAM, the Special Interest Committee on Symbolic and Algebraic Manipulation, which later became SIGSAM. Jean had led the development of FORMAC, which IBM made into a product [14]. I worked for Jean in the summer of 1964, unsuccessfully attempting to convince her group to add a pattern matching procedure to FORMAC. It helped the emerging community of symbolic systems builders that IBM had a product, especially one that had somewhat limited capabilities.

Other attendees at this SYMSAM conference included Tony Hearn, then at Stanford, who had been working on REDUCE, with an emphasis on solving problems in physics, especially Feynman diagrams which required special integration routines [15]. Little did we realize then that calculation of Feynman diagrams would lead to the 1999 Nobel physics prize for Martinus Veltman and his student for work done about six years later using his system SCHOONSCHIP [16]. Bell Labs had Stan Brown whose ALPAK system made calculations with rational functions, which were of value to researchers at the Labs [17]. His system was more powerful than MATHLAB. IBM Research had George Collins whose work already showed his mastery of algebraic algorithms [18]. IBM Research also had Jim Griesmer and Dick Jenks who started the development of SCRATCHPAD, a system with broad symbolic capabilities, in 1965[19].

An interesting question is why there was so much interest in symbolic systems and algorithms at that time. I think one reason is that numerical algorithms were not yet seen as powerful then as they are seen now, and this meant that there was continued effort to use the classic symbolic mathematical approaches to the solution of problems, especially in physics and engineering. For me, it was fun to be able to implement symbolic algorithms that appeared in my mathematics courses, and that also seemed to have practical value.

I went on the road in 1967 giving what were effectively job talks in those days. In contrast to today where one has to reply to advertisements regarding a search, things were much looser in those days. In CMU I had a long question-and-answer session with Alan Newell regarding my approach to AI. Bob Caviness was in the audience that day. Alan Perlis, the department chair, had a special interest in symbolic computing and his students created a symbolic system [20]. According to Caviness, one student was supposed to create an integration system, but unfortunately he died before he could finish it. At Bell Labs I was accompanied by Bill Martin. Several of the researchers we met that day have been friends of mine since, especially Elwyn Berlekamp whose factorization algorithm over primes was used later in Macsyma [21]. Tony Hearn was my host at Stanford and helped me during the lecture when I got confused in my description of an algorithm.

Back at MIT, Martin began deliberations regarding a new symbolic mathematics system that would combine all our work (Bill's, Carl Engelman's and mine), and would use the latest algorithms that we heard about at the 1966 conference. The system, which I later named Macsyma, Project MAC's SYmbolic MAnipulator, would rely on multiple representations and would be written in LISP. The general representation for expressions would be similar to that of FORMAC, except that it would use LISP's list structures. It could represent any expression, and its simplifier would have limited capabilities. The rational function representation would handle ratios of polynomials in multiple variables with integer coefficients, like ALPAK. It would rely on a GCD algorithm to keep rational functions in simplified form. Over time we added other representations, such as one for power series. The design meetings began in earnest in 1968. We obtained research support from ARPA beginning in July 1969, and we made our first staff hire, Jeff Golden, at that point. Our growing system created a major load on the AI PDP-10 computer so that ARPA agreed to let us buy a new memory of one quarter million words, for a total memory that was double the maximum available from DEC at that time. The fun and games that I was previously having now began to have some serious consequences, given the great expense of the memory ($400K in 1968). Not much later our Project MAC director, JCR Licklider, was able to convince ARPA to let us buy our own PDP-10, called the Mathlab machine. We made it a node on the growing ARPANET. During some months that machine was one of the most popular nodes on the ARPANET.

Our coming-out occurred at the 1971 SIGSAM Symposium [22]. Our group had seven papers at that meeting. Martin and Fateman wrote a description of Macsyma for that symposium [23]. The 1971 Symposium indicated great depth in the community, both in algorithm design and in systems and applications. Some of the major algorithms presented were modular ones [24]. Modular algorithms worked for polynomials in several variables. All but one variable were substituted by integers, and the resulting univariate problem was solved. Given enough such substitutions one could figure out the multivariate answer for the original problem by an interpolation scheme. We came back from the Symposium very interested in implementing these new algorithms and making them available to our growing community over the ARPANET. The day we introduced the modular gcd algorithm as the standard gcd algorithm in Macsyma the system ground to a halt and we immediately received numerous complaints from the user community. We were surprised by this, since we were led to believe by some that the modular algorithms were optimal ones. I analyzed why the modular GCD algorithm performed poorly, and realized that it took essentially the same time when a multivariate polynomial was sparse as when it was dense and possibly had a number of terms that is exponential in the number of variables. This would not have bothered George Collins very much since he was interested in logic-based problems that were usually dense, but most of our users had sparse problems. We immediately replaced the modular algorithms with our previous algorithms, and began to perform research on algorithms that could handle sparse polynomials.

Soon after we returned from the 1971 conference Bill Martin surprised me by saying that he wanted to leave the project. I took over his role and ran the group for the next dozen years. Bill's role in the development of Macsyma was critical. He led the project for three years. He emphasized the goal of creating a system that had multiple representations and included most of the algorithms that were known at the time. He probably was the one who emphasized the need for developing a comprehensive system that would be useful to engineers, scientists as well as mathematicians. On the other hand, all the MIT doctoral students in the project were supervised by me, and thus Bill did not get all the credit he deserved.

Paul Wang did his doctoral thesis on limits and definite integration [25]. As a faculty member in mathematics he worked on polynomial factorization with a mathematics post-doc, Linda Preiss Rothschild. They began with Berlekamp's algorithm for factorization over the integers modulo a prime. They extended the resulting factors to factors over a prime power. When the prime power exceeded the integers that could be coefficients in a factorization over the integers, then one can check to see if the generated factors are indeed factors over the integers. They generalized the approach to factorization in several variables by substituting integers for all but one variable and extending the result to several variables [26]. The key new idea in their multivariate algorithm was the extension technique, which is called the Hensel lemma in algebra and is a variant of Newton's method. Hensel's lemma could usually be employed with just one factorization over the integers of a univariate polynomial, as opposed to an exponential number that might have been needed with a modular approach.

One day David Yun, whom I had asked to look into GCD algorithms for sparse polynomials, pointed out that the GCD of two polynomials is a factor of each polynomial, and hence a similar approach to factorization used by Wang and Rothschild could apply. We were very excited by this idea. We soon discovered some problems with the approach, which we called the EZ GCD algorithm. We were able to circumvent one problem, but had difficulty with another problem that arose when the substitution for all but one variable trivialized the resulting univariate polynomial. We made other randomly chosen substitutions to get around the problem, but such substitutions often increased the size of the resulting problem. Nevertheless, the EZ GCD algorithm was better than the alternative ones in many cases, sometimes by many orders [27].

By 1974 ARPA decided that it had contributed enough to the system's development for the past five years. It asked MIT to turn over the support for further R&D to the Macsyma user community. As a going away present ARPA paid for a newer and faster version of the PDP-10 we were using. It became known as the Macsyma Consortium computer, and was also made available on the ARPANET. The Consortium members included the DOE, NASA, US Navy and Schlumberger. The consortium funded the group for the next 7-8 years. The Macsyma system as of 1974 is described briefly in my paper "Macysma - The Fifth Year" [28].

I began to get increasingly involved in academic administration, initially as associate director of the Laboratory for Computer Science in 1974. The first group of doctoral students had graduated by then, and only two remained, namely Richard Zippel and Barry Trager. Some years later Zippel would write a thesis on extending the EZ GCD algorithm in the cases where a straight-forward substitution failed [29]. Trager spent several years at IBM Research but returned to finish a thesis on the integration of algebraic functions [30].

Much of the effort in the group turned to the development of a relatively bug free system as well as new features, such as tensor calculations. Some of the effort went into a better LISP compiler. The overall system had grown quite large and the core system was having great difficulty fitting in the 256K word limit of the PDP-10 computer. Our hope was that DEC would develop a version of the PDP-10 that would have a large address space. This was also a hope of the rest of the Laboratory for Computer Science (the new name for Project MAC) and the AI Lab. DEC's VP Gordon Bell promised to deliver a much cheaper version of the PDP-10 with large address space by 1978. We were quite surprised when Bell returned with some of his colleagues and unveiled the DEC VAX architecture. So much of the Lab's software and that of the other main ARPA-funded universities was based on the PDP-10 architecture. On the other hand, DEC made a business decision to go with the VAX architecture. This change of architecture by DEC cost the ARPA community several years of system development. Our group bit the bullet and undertook a project to develop a LISP for the VAX, called NIL for New Implementation of LISP. VAX-based versions of Macysma would permit many users to have their own copies of the system, even on microprocessor-based machines. Such versions would eventually be written in COMMON LISP in the 1980's.

An exciting event took place in 1977. Richard Fateman, formerly of our group, and then on the faculty of UC Berkeley, ran the first Macsyma Users Conference at Berkeley. The member of our group who attracted the most attention was Ellen Lewis, who was the main interface to our users. I recall introducing Richard Gosper as the only living 18th century mathematician since the problems he was interested in were generally from the 18th century. Gosper was an expert on summation in closed form [31], and was the only one I knew who had a deep understanding of Ramanujan's notebooks.

I began a 20 year stint as a full-time academic administrator in 1978. My positions were: head of the MIT computer science faculty, head of the electrical engineering and computer science department, dean of engineering and finally provost of MIT. These positions meant that I could not devote much time to running the group. I also lost some interest in algorithm development. For example, Groebner basis algorithms did not fascinate me since I assumed that many problems that relied on Groebner bases simply took exponential time. In contrast our use of the Hensel lemma reduced the cost of computation in practice by many orders. I was interested in the mathematics of special functions, which would broaden the use of symbolic mathematics well beyond the usual functions in the calculus [32]. However, I assumed that this was a programme that would take decades. Thus in 1981 I began discussions within MIT about forming a company, which would distribute and develop Macsyma to a large number of users on VAX-like machines and even smaller computers. The Bayh-Dole Act had recently passed in the US and this meant that work sponsored by the US government could be licensed by universities for a fee, as long as the government obtained the ability to use it for itself. Unfortunately, at that early point there was little experience with the Act. In particular, the Department of Energy, one of our consortium sponsors, was asked by some of our users and developers to force the software to be available for free to everyone. I opposed such a move because significant funds were needed to maintain and develop the system further. The MIT administration was concerned that it might be in a conflict of interest in permitting one of its faculty members and some of its staff to profit from government-sponsored research and development. The administration decided to let the Arthur D Little company, a local consulting firm, determine to whom to license the software. Arthur D Little decided to license it to Symbolics, Inc, a company that was formed by former MIT staffers to produce LISP machines. I opposed this license because I felt that Symbolics would have a conflict of interest in licensing VAX-based Macsyma systems in competition with its LISP machine-based systems. In fact, Arthur D Little had a conflict of its own since it had a fund for its employees that was a major investor in Symbolics. MIT decided, however, to license Macsyma to Symbolics, and some of our staff, such as Jeff and Ellen Golden, went to work for them. The group terminated activities at MIT in 1982.

The early 80's also saw the development of new systems. SMP was developed largely by Steve Wolfram, a former Macsyma user from Cal Tech [33]. It had the feature that coefficients were floating point numbers, which made the GCD algorithm not applicable to its expressions. Maple was presented at the 1984 Macsyma Users' Conference [34]. The emphasis, from our perspective, was on careful engineering of the system. One emphasis was on reducing Maple's core system's memory requirements so that it could operate on hardware that was cheaper than Macsyma's at the time. A second emphasis that we noted was on careful programming of the basic algorithms so that speed was increased in common cases.

Symbolics was able to sell Macsyma licenses on a VAX soon after obtaining the license from MIT, but Macsyma systems on personal computers were late in coming, and this became a serious competitive disadvantage during the 1980's. Furthermore the Department of Energy insisted on a free version and MIT finally gave one to them to be placed in a public data base. My concern about internal conflicts within Symbolics was justified, and the "AI winter" caused in part by the overselling of rule-based expert systems, usually implemented in LISP, eventually led to the demise of Symbolics as a hardware manufacturer. The Macsyma software was finally sold to a company called Macsyma Inc, but it was too little and too late, and that company failed as well in the early 90's. A version of Macsyma, called MAXIMA, is currently available on the net, but it does not contain the improvements made at Symbolics.

My research in the past 25 years can be said to be influenced, in part, by my experience with SIN and Macsyma. As I developed SIN I was increasingly concerned over the classic approach to AI in the 1950's, namely heuristic search, a top-down tree-structured approach to problem solving. In the late 1960's there began the development of the software engineering approach in Computer Science, which is another version of a top-down tree structured approach to design. In the 1970's I began reading the literature on the management of human organizations, and there was Herb Simon again emphasizing a top-down hierarchical approach to organization. I could not understand why Americans were so enamored with what I considered an approach that would fail as systems became larger, more complex, and in need of greater flexibility.

In the 1980's the US became very concerned over the loss of manufacturing jobs to the Japanese and to a degree the Germans. When I began reading the literature on Japanese management, I recognized ideas I had used in SIN and Macsyma [35]. There was an emphasis on abstraction and layered organizations as well as flexibility. These notions are present in abstract algebra. In particular, a hierarchy of field extensions, called a tower in algebra, is a layered system. Such hierarchies are extremely flexible since one can have an infinite number of alternatives for the coefficients that arise in each lower layer. But why were such notions manifest in some societies and not so much in Anglo-Saxon countries? My answer is that these notions are closely related to the national culture, and countries where there are multiple dominant religions (e.g., China, Germany, India, and Japan) would tend to be more flexible than ones where there is one dominant religion. Furthermore, if one of the religions had a layered approach to hierarchies (e.g., Shinto in Japan) then that country would have a deeper understanding of relatively flat, layered hierarchies. My recent work deals with the design of large scale engineering systems using approaches to design that are based on notions, such as platform-based design and layering [36, 37]. Further discussion of these issues and many others can be found in my memoirs [38].

References

1. Slagle, J. R., A heuristic program that solves symbolic integration problems in freshman calculus: symbolic automatic integrator (SAINT), PhD dissertation, MIT, 1961

2. Matiyasevich, Y. V., *Hilbert's Tenth Problem*, MIT Press, 1993

3. Richardson, D., "Some unsolvable problems involving elementary functions of a real variable," *J. Symbolic Logic* 3, pp. 511-520, 1968

4. Moses, J., *Symbolic Integration*, MAC-TR-47, MIT, 1967

5. Feigenbaum E.A. et al, On generality and problem solving: A case study using the DENDRAL program, Machine Intelligence 6, Edinburgh University Press

6. Feigenbaum, E. and McCorduck, P., *Fifth Generation Artificial Intelligence and Japan's Computer Challenge to the World*, Addison Wesley, 1983

7. J. F. Ritt, *Integration in finite terms*, Columbia University Press, 1948.

8. R. H. Risch, "The Problem of Integration in Finite Terms". Transactions of the American Mathematical Society 139: 167-189, 1969.

9. G. H. Hardy, *The Integration of Functions of a Single Variable,* 2nd Edition, Dover, 2005

10. Hammer, M. and Champy, J. A.: *Reengineering the Corporation: A Manifesto for Business Revolution*, Harper Business Books, New York, 1993

11. Van der Waerden, B.L., Algebra, Part 2, Springer, 1959

12. Moses, J., “Algebraic Simplification – A Guide for the Perplexed,” Comm. ACM, vol. 14, pp. 527-537, 1971

13. Martin, W. A., *Symbolic Mathematical Laboratory*, Project MAC, MAC-TR-36, MIT, 1967

14. Bond, E. et al, FORMAC - an experimental formula manipulation compiler, ACM, 1964

15. Hearn, A.C., *REDUCE 2 user's manual*, Rep. UCP-19, Univ. of Utah, Salt Lake City, 1973

16. Strubbe, H. SCHOONSCHIP user manual, *Comput. Phys. Commun*. 1975

17. Brown, W.S., A language and system for symbolic algebra on a digital computer, Proceedings of the First ACM Symposium on Symbolic and Algebraic Manipulation, pp. 501 - 540, 1966

18. Collins, G.E., Polynomial remainder sequences and determinants. Amer. Math. Monthly 73, 7 1966, 708-712

19. Griesmer, J.H., Jenks, R.D., and Yun, D.Y.Y. *SCRATCHPAD user's manual*, Rep. RA70, IBM T.J. Watson Research Center, Yorktown Heights, N.Y., 1975

20. Perlis, A. J. et al, "An Extension of ALGOL for Manipulating Formulae", CACM 7(2):127-130, 1964

21. Berlekamp E. R., Factoring Polynomials over Finite Fields, BSTJ, vol 46, pp 1853-1859, 1967

22**.** Petrick, S. R., Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation, Los Angeles, California, ACM, 1971

23. Martin, W.A., and Fateman, R.J., The MACSYMA System, Proc. 2nd Symposium on Symbolic and Algebraic Manipulation, pp. 59-75, 1971

24. Brown, W. S., On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors, JACM, 18(4), pp. 478-504, 1971

25. Wang P.S., *Evaluation of Definite Integrals by Symbolic Manipulation*, PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1971

26. Wang, P.S. and Rothschild, L.P., Factoring multivariate polynomials over the integers. Math. Comp, 29:935--950, 1975

27. Moses, J. and D.Y.Y. Yun, ``The EZ GCD Algorithm," Proc. ACM 1973, ACM, New York, pp.159-166, 1973

28. Moses, J., **"**MACSYMA **-** The Fifth Year**."** SIGSAM Bulletin 8(3), Aug 1974

29**.** Zippel, R. E., Probabilistic algorithms for sparse polynomials, Proceedings of EUROSAM ’79, Springer-Verlag LNCS 72, pp. 216–226, 1979.

30. Trager B. M., *Integration of Algebraic Functions*, PhD. dissertation, Mass. Inst. of Tech, EECS Dept. 1984

31. Gosper, R. W., “Indefinite hypergeometric sums in MACSYMA," Proceedings of the First MACSYMA Users' Conference (Berkeley), 1977

32. Moses, J., “Towards a General Theory of Special Functions,” CACM, Vol. 15, no. 7, pp. 550-554, 1972

33. Cole, C.A., Wolfram, S. et al, SMP: a symbolic manipulation program, Cal. Institute of Tech., 1981

34. Char B., et al, “On the Design and Performance of the Maple System,” Proc. 1984 Macsyma Users’ Conference, pp. 199-219, 1984

35. Ouchi, W.G., *Theory Z*, Addison-Wesley, 1981

36. Moses, J., “Foundational Issues in Engineering Systems: A Framing Paper,” Engineering Systems Monograph, esd.mit.edu, March 2004

37. Moses, J., “Three Design Methodologies, Their Associated Structures, and Relationship to Other Fields,” Engineering Systems Symposium, esd.mit.edu, March 2004

38. http://esd.mit.edu/Faculty_Pages/moses/moses_memoirs.pdf

How fast can we multiply and divide sparse polynomials?

Michael B Monagan
Simon Fraser University

Abstract

Most of today's computer algebra systems use either a sparse distributed data representation for multivariate polynomials or a sparse recursive representation. For example Axiom, Magma, Maple, Mathematica, and Singular use distributed representations as their primary representation. Macsyma, REDUCE, and TRIP use recursive representations. In 1984 David Stoutemyer suggested "recursive dense" as an alternative and showed that it was the best overall across the Derive test suite. Of the newer systems, only Pari uses recursive dense.

In 2003, Richard Fateman compared the speed of polynomial multiplication in many computer algebra systems. He found that Pari was clearly the fastest system on his benchmarks. His own implementation of recursive dense came in second. So is recursive dense best?

In this talk we take another look at the sparse distributed representation with terms sorted in a monomial ordering. Our algorithms for polynomial multiplication and division use an auxiliary data structure, a "chained heap of pointers". Using a heap for polynomial arithmetic was first suggested by Stephen Johnson in 1974 and used in the Altran system. But the idea seems to have been lost. Let $h = fg$ where the number of terms in f, g, and h are #f, #g, and #h. The heap gives us the following:

1. It reduces the number of monomial comparisons to O(#f #g log min(#f, #g)).
2. For dense polynomials, chaining reduces this to $O(\#f\#g)$.
3. By using O(1) registers for bignum arithmetic, multiplication can be done so that the terms of the product fg are output sequentially with no garbage created.
4. The size of the heap is $O(\min(\#f, \#g))$ which fits in the cache.
5. For polynomials with integer coefficients, the heap enables multivariate pseudo-division - our division code is as fast as multiplication.

In the talk I would like to first show Maple's distributed data structure, Pari's recursive dense structure, Yan's "geobuckets" which are used in Singular for division, and our own distributed structure. Second I will show how heaps of pointers work. Third, our benchmarks suggest pointer heaps are really good. The timings are much faster than Pari, Magma, and Singular and much much faster than Maple. Fourth, I will show some of the other "necessary optimizations" needed to get high performance. The most important one is to encode monomials into a single word. But there are others. Including the right way of extracting an element from a heap.

This is joint work with Roman Pearce at Simon Fraser University.

Maple as a prototyping language: a concrete and successful experience

Gaston Gonnet
ETH Zurich

Abstract

Aruna PLC, a database company, embarked around 1999-2001 in the writing of an SQL (a relational database query language) query processor. A query processor normally consists of a parser, a simplifier of the query expression, a plan generation (how to execute the query) and a plan optimizer. Since we had many ideas about how to implement such a query processor and these were difficult to transmit to the programmer team, we decided to write a prototype of the QP in Maple to provide a "live" example of the algorithms. The prototype in Maple turned to be a full fledged prototype which is alive and active even today. We will describe several of the highlights of the prototype and how it contributed in many ways to the development of the query processor. Some software engineering aspects are also quite remarkable. The pervasive "symbolic" nature of the Maple implementation was also an extremely positive feature.

Integrals, Sums and Computer Algebra

Peter Paule
Research Institute for Symbolic Computation (RISC),
Johannes Kepler University, Linz, Austria

Abstract

The types of mathematics being considered in in this talk are related to some of Keith Geddes' research interests, namely: computational aspects of algebra and analysis, including the solution of problems in integral and differential calculus, and closed-form summation. The thread of my presentation will be Victor Moll's article "The evaluation of integrals: A personal story" (Notices of the AMS, 2002) which begins with the remark, "... It was even more a surprise to discover that new things can still be said today about the mundane subject of integration of rational functions of a single variable and that this subject has connections with branches of contemporary mathematics as diverse as combinatorics, special functions, elliptic curves, and dynamical systems." In this talk I will add another ingredient to Moll's story, namely computer algebra. I will show how recently developed procedures can be used to retrieve observations which in Moll's original approach were derived with classical methods, like the positivity of the coefficients of a specialized family of Jacobi polynomials. In addition, as a result from a recent collaboration with Manuel Kauers (RISC), I will demonstrate that computer algebra can do even more, namely by proving Moll's longstanding log-concavity conjecture with a combination of various algorithms.

Linear Algebra

B. David Saunders
University of Delaware, USA

Abstract

Computer algebra systems have become a major tool of science and engineering education and practice. Thoughts of CAS immediately bring to mind Maple, and thus Keith Geddes, but not so much linear algebra. I'm sure Keith has never long entertained a linear thought. That is good! To solve a linear system is perhaps the best understood of mathematical problems. However, this is largely because the concepts of matrix and linear system are overly general. In fact there is not one ”solve linear system” problem, but many, depending on the structure of the matrix and application. It remains a challenge to compute solutions efficiently as hardware evolves, and the matter is rich in interesting computer science and mathematics and of growing importance to computer algebra. I will survey the history and the state of the art, and offer a view of the road ahead.

Ten Commandments for Good Default Expression Simplification

David R. Stoutemyer
dstout at Hawaii Dot edu

October 22, 2008

Abstract

This article motivates and identifies ten goals for good default expression simplification in computer algebra. Although oriented toward computer algebra, many of these goals are also applicable to manual simplification. The article then explains how the Altran partially-factored form for rational expressions was extended for Derive and the computer algebra in Texas Instruments products to help achieve these goals. In contrast to the distributed Altran representation, this recursive partially-factored semi-fraction form

- doesn't unnecessarily force common denominators,
- discovers and preserves significantly more factors,
- can represent general expressions, and
- can produce the entire spectrum from fully factored over a common denominator through complete partial fractions, including a dense subset of intermediate forms.

1 Introduction

Simplicity is the peak of civilization
— Jessie Sampter

First, an explanation for the title: "Goals" is a more accurate word than "commandments", because no current computer-algebra system thoroughly obeys all of them, and incompletely fulfilling goals seems less reprehensible than disobeying commandments. However, with apologies to the author of the original Ten Commandments, these goals are called commandments in the title because:

- Moses (1971 AD) is cited in this article.
- Ten years later he reappeared in Biblical apparel at a computer-algebra conference. (Moses, 1981 AD).
- It is foretold that he will be present at the Milestones in Computer Algebra conference where this Ten Commandments article will be presented. (Moses, 2008 AD).

Computer-algebra programs such as MathPert (Beeson 1998) and the Texas Instruments Student Math Guide (SMG 2003) help teach mathematics by having students choose a sequence of elementary transformations to arrive at a result. The transformations can be as elementary as combining numeric sub-expressions, applying 0 and 1 identities, sorting factors or terms, combining similar factors or terms, subtracting an expression from both sides of an equation, or applying a specific differentiation rule. With such **step-oriented** systems, the overall goal is a wise path having several steps at an appropriate tutorial granularity. The interface is oriented around a sequence of equivalent expressions annotated by rewrite rules selected from a context-dependent menu by the user. MathPert and Derive also have tutorial modes wherein the system automatically chooses and displays a sequence of steps either uninterrupted or one step per press of the [Enter] key.

In contrast, for the **result-oriented** computer-algebra systems being considered here, the overall goal is a satisfying final result in as few steps as possible – preferably one step. The interface is typically oriented around a sequence of input-result pairs. With some changes for annotation, the result-oriented interface could be a special one-step case of the result-oriented interface.

Default simplification means what a computer-algebra system does to a standard mathematical expression when the user presses the [Enter] key, using factory-default mode settings, without enclosing the expression in a transformational function such as expand(. . .), factor(. . .), or tryHarder(. . .).

Computer-algebra users generally expect some transformation when they press [Enter]. Otherwise they already have the desired result and need at most a system for 2D input and display of mathematical expressions. If the input expression contains an unevaluated integral, derivative, sum or limit, most often users want to have that sub-expression replaced with a closed-form result. Otherwise, in the absence of a transformational function such as expand(. . .) or factor(. . .), users haven't indicated a strong preference for any particular form. However, they presumably want the result simpler if practical. Section 2 motivates and presents ten goals that are applicable to this most common case.

Section 3 describes how the recursive partially-factored form in Derive and in the separate computer algebra in some Texas Instruments products helps meet some of these goals.

Section 4 describes how the form is further extended to partial fractions and to intermediate forms to further meet some of these goals.

Appendix A describes further details for partial fractions.

Appendix B describes further details for ratios of polynomials.

Appendix C describes additional issues for fractional exponents.

Appendix D describes additional issues for functional forms.

Appendix E contains pseudo-code for multiplying and adding partially-factored semi fractions.

2 What should we want from default simplification?

Nothing is as simple as we hope it will be
— Jim Horning

This section discusses key issues for default simplification and some goals that are highly desirable for default simplification.

2.1 Correctness is non-negotiable.

Everything should be made as simple as possible, but not simpler.
— Albert Einstein

Definition 1 (domain of interest) *The domain of interest for an expression is the Cartesian product of the default or declared domains of the variables therein, as further restricted by any user-supplied equalities and/or inequalities.*

To determine a concise result within the domain of interest, we can use transformations that aren't necessarily valid outside that domain. For example, some transformations that are valid for all integers or for all positive numbers aren't valid for more general real numbers, and some transformations that are valid for all real numbers aren't valid for all complex numbers.

There might be some points in the domain of interest where some users regard the input expression as being undefined.

For example, although we can represent $\sin(\infty)$ as the interval $[-1, 1]$, and the expression ± 1 as the multi-interval $\langle -1, 1\rangle$, many users regard such non-unique values as undefined, at least in some contexts.

Here are some of the rewrite rules that can define a *domain of uniqueness* function dou(...):

$$\begin{aligned}
\mathrm{dou}(\pm u) &\rightarrow u = 0,\\
\mathrm{dou}(\sin(\infty)) &\rightarrow \mathit{false},\\
\mathrm{dou}(\mathit{number}) &\rightarrow \mathit{true},\\
\mathrm{dou}(\mathit{variable}) &\rightarrow \mathit{true},\\
\mathrm{dou}(u+v) &\rightarrow \mathrm{dou}(u) \wedge \mathrm{dou}(v),\\
&\ldots
\end{aligned}$$

As another example, most computer algebra systems represent and correctly operate on ∞, $-\infty$, and various complex infinities. Even $0/0$ is representable as the interval $[-\infty, \infty]$ or the complex interval $[-\infty - \infty i, \infty + \infty i]$. Nonetheless, many user's regard at least some of these as undefined, at least in some contexts. Here are some of the rewrite rules that can define a *domain of finiteness* function dof(...):

$$\begin{aligned}
\mathrm{dof}(\ln(u)) &\rightarrow \mathrm{dof}(u) \wedge u \neq 0,\\
\mathrm{dof}(\infty) &\rightarrow \mathit{false},\\
\mathrm{dof}(u^v) &\rightarrow \mathrm{dof}(u) \wedge \mathrm{dof}(v) \wedge (u \neq 0 \vee v > 0)\,,\\
&\ldots
\end{aligned}$$

Some users also regard non-real values as undefined, at least in some contexts. Here are some rewrite rules that can define a *domain of realness* function dor(...):

$$\begin{aligned}
\mathrm{dor}(\mathit{number}) &\rightarrow \mathit{number} \in \mathbb{R},\\
\mathrm{dor}(\mathit{variable}) &\rightarrow \mathit{variable} \in \mathbb{R},\\
\mathrm{dor}(\ln(u)) &\rightarrow \mathrm{dor}(u) \wedge u > 0,\\
&\ldots
\end{aligned}$$

Given these three functions, we can define a function dod(...) that computes the *domain of definition* according to any desired combination of the domains of uniqueness, finiteness and realness.

Definition 2 (domain of equivalence) *The domain of equivalence of two expressions is the domain for which they give equivalent values when ground-domain elements are substituted for the variables therein.*

> **Goal 1 (correctness)** *Within the domain of interest, default simplification should produce an equivalent result wherever the input is defined.*

Some transformations can yield expressions that are not equivalent everywhere. For example with the principal branch, $1/\sqrt{z} - \sqrt{1/z}$ is equivalent to 0 everywhere in the complex plane except where $\arg(z) = \pi$. Along $\arg(z) = \pi$ the expression is equivalent to $2/\sqrt{z}$. Therefore transforming the expression to either result would incorrectly contract the domain of equivalence in the domain of interest unless the input includes a constraint that implies one of these two results. Thus one way to achieve correctness is to leave the relevant sub-expression unchanged until the user realizes that an appropriate constraint must be attached to the input, then does so. Unfortunately, many users will fail to realize this, and they will judge the system unfavorably as being incapable of the desired transformation.

In an interactive environment, a more kindly route to correctness and favorable regard is for the system to ask the user whether or not $\arg(z) = \pi$, and automatically append a corresponding constraint to the users input, then do the corresponding transformation. If interested, the user can repeat the input with a different combination of replies to see another part of the complete result.

Unfortunately the query might end up being unnecessary. For example, the sub-expression might be multiplied by another sub-expression that subsequently simplifies to 0. Users who are conscientious enough to repeat the input with the opposite constraint or reply will be annoyed about being pestered with an irrelevant question. Users who don't try the opposite constraint will falsely conclude that the result isn't equivalent to the input without the constraint.

Also, this method can be baffling to a user if the question entails a variable such as a Laplace transform variable that is generated internally rather than present in the user's input. In such situations and for non-interactive situations, an alternative treatment is for the system to assume automatically the reply that seems most likely, such as $\arg(z) \neq \pi$, then append the corresponding constraint to the user's input before proceeding. Thus notified of the assumption, the user can later edit the input to impose the opposite assumption if desired.

A more thorough method, which doesn't require interaction or risk disdain, is for the system to develop a piecewise result equivalent for all z, such as

$$\frac{1}{\sqrt{z}} - \sqrt{\frac{1}{z}} \rightarrow \text{ if } \left(\arg(z) = \pi \text{ then } \frac{2}{\sqrt{z}} \text{ else } 0\right).$$

As another example, monic normalization does the transformation

$$a \cdot x + 1 \rightarrow a \cdot \left(x + \frac{1}{a}\right).$$

The left side is defined at $a = 0$, but the right sided isn't, as further discussed in Stoutemyer (2008a). A piecewise result equivalent for all a and x is

$$a \cdot x + 1 \rightarrow \text{ if } \left(a = 0 \text{ then } 1 \text{ else } a \cdot \left(x + \frac{1}{a}\right)\right).$$

Quite often users are interested in only one of the alternatives, which they can obtain by cut and paste or by resimplifying the input or result with an appropriate input constraint. However, such piecewise results can become combinatorially cluttered when combined, so there is still a place for a "query and modify input" mode.

Goal 2 (contraction prevention) *If necessary for equivalence where the input is defined in the domain of interest, a result should be piecewise or the system should append an appropriate constraint to the input, preferably after querying the user.*

2.2 Managing domain enlargement

Some transformations can yield results that are defined where the input is undefined. For example, many users regard $(z^2 - \pi \cdot z)/(z - \pi)$ as undefined at $z = \pi$. For such users, the transformation

$$\frac{z^2 - \pi \cdot z}{z - \pi} \rightarrow z$$

enlarges the domain of definition.

The enlargement is a benefit rather than a liability because:

- Unlike the input, the result doesn't suffer catastrophic cancellation for z near π.
- Defining a value at $z = \pi$ turns a partial function into a more desirable total function, and the omni-directional limit of the input as $z \rightarrow \pi$ is clearly the best choice.
- Removable singularities are often merely a result of an earlier transformation or a modeling artifact that introduced them. For example, perhaps they are a result of a monic normalization or a spherical coordinate system.
- The phrase "removable singularities" gives us permission to remove them.

Transformations that enlarge the domain of definition make the result *better than equivalent*.

However, there are vocal critics of such gratuitous improvements. To appease them, there should be a mode they can turn on to preserve equivalence but still enjoy the other benefits of the transformation:

> **Goal 3 (enlargement prevention)** *Results should optionally include appropriate constraints if necessary to prevent enlarging the domain of definition within the domain of interest.*

For example:

$$\frac{z^2 - \pi \cdot z}{z - \pi} \to z \mid z \neq \pi.$$

Complaints about domain enlargement are an unfortunate consequence of the historical emphasis on equivalence rather than on transformation to an expression that is equivalent *or better*.

Table 1: Candidate notation for changes in domain of definition

Example	Read as	Analogy
$A \sqsubset B$	B is more universal than A	$\mathrm{dod}(A) \subset dod(B)$
$A \sqsubseteq B$	B is equivalent to A or better	$\mathrm{dod}(A) \subseteq dod(B)$
$A \sqsupset B$	B is less universal than A	$\mathrm{dod}(A) \supset dod(B)$
$A \overset{:)}{\to} B$	A improves to B	happy-face emoticon
$A \overset{:)}{\Rightarrow} B$	A transforms to or improves to B	equivalent to or better
$A \overset{:(}{\to} B$	A degrades to B	sad-face emoticon

Some convenient notations and phrases might help make domain enlargement more acceptable. Table 1 lists three relational operators then three corresponding transformational operators that are easily constructible in LaTeX. For example,

$$\frac{z^2 - \pi \cdot z}{z - \pi} \overset{:)}{\to} z.$$

2.3 Protecting users from inappropriate substitutions

Seek simplicity, and distrust it
— Alfred North Whitehead

As a corollary to Murphy's law, someone will eventually apply any widely-used result outside the domain of equivalence to the inputs, unless explicitly prevented from doing so or unless the result is universally valid. For example, as discussed by Jeffrey and Norman (2004) most publications containing the Cardano formula for the solution of a cubic equation don't state that it isn't always correct for non-real coefficients. Many people including some computer-algebra implementers have misused the formula with non-real coefficients. (*Mea culpa*.) The consequences can be disastrous.

Implementing the enlargement-prevention goal entails a mechanism for attaching constraints to results. With that mechanism implemented, it is not much additional effort also to propagate to the output any input constraints introduced by the user or the system, and to have the system return the representation for "undefined" whenever a user subsequently makes a substitution that violates result constraints.

The system should also determine and propagate the intersection of domains when expressions having different domains are combined. The domains are most generally represented as Boolean expressions involving inequalities, equalities and type constraints such as $n \in \mathbb{Z}$.

The constraint expression should be simplified as much as is practical, to make it most understandable. We can omit it if it simplifies to *true*. If it simplifies to *false*, then the algebraic result is undefined. The simplification should preferably be done in a way that doesn't introduce additional constraints.

Constraint simplification can be quite difficult or undecidable. However, perfection isn't mandatory. The purpose of the constraint is to return the representation for undefined if a substitution makes the Boolean constraint simplify to *false*. Otherwise the result is that of the substitution, with a more specialized attached constraint when it doesn't simplify to *true*. A properly-derived result such as $5 \mid B$ is still correct even if Boolean expression B could be simplified to *true* or to *false* but wasn't.

For safety, the output constraint should indicate the basic domain of every variable in the output expression. This can be done by including type constraints of the form *variable* $\in$ *domain*. However, to reduce clutter, the types of variables can often be inferred from constraints. For example, the constraint $x \geq 0$ implies $x \in \mathbb{R}$. In such cases we can omit a type constraint for x if it isn't a more restricted type such as integer. As another example, the sub-expression $\neg b$ implies that variable b is Boolean. Also, if one type such as $\mathbb{C}$ includes all other possible declared or default numeric types, then declarations of that type can be omitted if arithmetic or comparison operators imply that the variable is numeric.

Despite such economies, results can become distractingly cluttered. Therefore, the default could be to represent any complicated or routine portions of the constraint with an ellipsis that could be expanded by clicking on it. Moreover, there could be an option to totally hide output constraints. They would still, however, be attached to the result to insure safe substitutions within the computer-algebra system. They would also at least *encourage* safe substitutions outside the system if displayed in publications produced from the results.

> **Goal 4 (domain propagation)** *Input domains and constraints should be propagated into results, where they then cause substitution of inappropriate values to return the representation for undefined.*

2.4 Disabling default transformations

No matter how modest the set of default transformations, many mathematics educators wish they could sometimes be selectively disabled. At these times, such users would be better served by a voyage-oriented system such as MathPert or Student Math Guide that is designed for that purpose.

However, even for research or the exposition thereof, mathematicians sometimes want to disable transformations that they most often want as default. For example, many users might prefer 2^{9999} to the 3010 digits of the decimal form.

As another example, combining numeric sub-expressions in the coefficients of a truncated power series can mask revealing patterns such as in

$$\frac{1}{2}x^0 + \frac{1\cdot 3}{2\cdot 4}x^2 + \frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}x^4 + o(x^4) \text{ versus } 1 + \frac{3}{8}x^2 + \frac{5}{16}x^4 + o(x^4),$$

where $x^0 \overset{:)}{\rightarrow} 1$ is also disabled.

As another example, about 30% of the examples using "$\rightarrow$" in this article are multi-step examples.

Such users want to be able to do this stepping in the same software environment that they use for destination-oriented mathematics. Therefore, a compassionate expression simplifier allows selectively disabling such default transformations.

> **Goal 5 (disable transformations)** *It should be possible to selectively disable default transformations.*

The necessary expression representation and algorithms to support such low-level algebraic control are so different from what is best for high-performance destination-oriented computer algebra that it is probably best to implement transformation disablement as a mode that switches to a different data representation and simplifier. For example:

- Fine-grain syntactic control or teaching the laws of signs requires internal representation of negation and subtraction of terms, whereas performance-oriented simplification typically forces signs into the numeric coefficients so that cases for negation, subtraction, addition, and interactions between them don't have to be implemented. (A post-simplification pass typically restores subtractions and negations for display.)

- Similarly for division versus multiplication by a negative power.
- Fine-grain syntactic control or teaching trigonometry requires internal representations of all the trigonometric functions and their inverses. In contrast, converting them all internally to a lean subset such as sines, cosines, inverse sines and inverse tangents automatically accomplishes many desirable transformations, such as $\tan(\theta)\cdot\cos(\theta) \rightarrow \sin(\theta)$. Tangents, inverse cosines etc. can often be restored for display either optionally or by default when it makes a result more compact.
- Similarly for fractional powers, versus square roots, cube roots etc. For example, allowing students to choose an appropriate transformation to simplify $\sqrt{2} - 2^{1/2}$ requires separate internal representations for square roots and fractional powers, which is a bad idea for a destination-oriented system. (A distressing portion of math education is devoted to contending with our many redundant functions and notations rather than learning genuinely new concepts!)

When default transformations are disabled, it is probably more appropriate to have the interface switch from destination mode to a step mode.

2.5 We want candid forms

Cancellation is key.

Definition 3 (candid) *A candid expression is one that is not equivalent to an expression that manifests a simpler expression class*

A candid form is "What You See Is What It Is" (WYSIWII).

Definition 4 (misleading) *A misleading expression is one that isn't candid.*

Misleading expressions masquerade as something more complicated than necessary. Examples of misleading expressions are:

- Expressions that are equivalent to 0 but don't automatically simplify to 0. For example,

$$f\left((x-1)\cdot(x+1)\right) - f\left(x^2-1\right).$$

- Expressions that contain superfluous variables. For example,

$$2\sinh(x) - e^x + e^{-x} + \ln(y), \text{ which is equivalent to } \ln(y).$$

- Apparently-irrational expressions that are equivalent to rational expressions. For example,

$$\frac{\sqrt{z}+1}{\sqrt{z}\cdot(z+\sqrt{z})}, \text{ which is equivalent to } \frac{1}{z}.$$

- Irrational expressions that are equivalent to other irrational expressions containing less nested and/or fewer distinct irrationalities. For example,

$$\sin(2\arctan(z)) + \left(15\sqrt{3}+26\right)^{1/3}, \text{ which is equivalent to } \frac{2z}{z^2+1} + \sqrt{3} + 2.$$

- Non-polynomial rational expressions that can be improved to polynomials. For example,

$$\frac{x^2-1}{x-1}, \text{ which improves to } x+1.$$

- Expressions that contain exponent magnitudes larger or smaller than necessary. For example,

$$(x+1)^2-(x-1)^2, \text{ which is equivalent to } 4x.$$

  This example is also a monomial masquerading as a multinomial. Reducible rational expressions provide another example of superfluous exponent magnitude. For example,

$$\frac{x^2-1}{x^2+2x+1}, \text{ which is equivalent to } \frac{x-1}{x+1}.$$

- Expressions that mislead us with disordered terms or factors. For example,

$$x^{18}+x^{17}+x^{16}+x^{15}+x^{14}+x^{13}+x^{19}+x^{11}+x^{10}+x^9+x^8+x^2+x^6+x^5+x^4+x^3.$$

  One could easily assume this is a degree 18 polynomial having a minimum exponent of 3. Worse yet, imagine the unlikeliness of noticing otherwise if the expression was several pages long and included lengthy coefficients. Complying with traditional ordering for commutative and associative operators greatly aids comprehension. Moses (1971) contains an analysis of the subtleties involved.

- Expressions that contain i or a fractional power of -1 but are actually real for real values of all variables therein. For example,

$$\frac{i\cdot((3-5i)\cdot x+1)}{((5+3i)\cdot x+i)\cdot x}, \text{ which improves to } \frac{1}{x}.$$

- Non-real expressions that have a concise rectangular or polar equivalent but aren't displayed that way. For example,

$$\frac{(-1)^{1/8}\cdot\sqrt{i+1}}{2^{3/4}}+i\cdot e^{i\pi/2}, \text{ which is equivalent to } \quad -\frac{1}{2}+\frac{i}{2}.$$

  Most users can easily envisage a useful geometric image only for rectangular and polar representations of either the form $(-1)^\alpha$ or $e^{i\theta}$.

- Expressions that mislead about important qualitative characteristics such as frequencies, discontinuities, symmetries or asymptotic behavior. For example,

$$\frac{\sin(4x)}{\cos(2x)}, \text{ which improves to } 2\sin(2x).$$

$$2\arctan\!\left(3\tan\!\left(\frac{x}{2}\right)\right)-\operatorname{mod}(x-\pi,2\pi)+\pi \mid x\in\mathbb{R}, \text{ equivalent to } 2\arctan\!\left(\frac{\sin x}{2-\cos x}\right).$$

- Boolean expressions that contain superfluous sub-expressions. For example, the prime implicant disjunctive normal form

$$(a\wedge b\wedge c\wedge d)\vee(a\wedge\neg c\wedge\neg d)\vee(\neg a\wedge\neg c\wedge d)\vee(\neg a\wedge\neg b\wedge\neg c)\vee(\neg b\wedge\neg c\wedge\neg d),$$

  for which either of the last two conjuncts (but not both) is superfluous.

- Boolean combinations of equalities and inequalities that can be expressed more succinctly. For example,

$$\operatorname{mod}(2\cdot\lfloor x\rfloor,2)=0\wedge((x>3\wedge\neg(x\le 5))\vee x=5), \text{ which is equivalent to } x\ge 5.$$

- Higher transcendental expressions that are equivalent to elementary transcendental expressions. For example, with $j_0(z)$ and $j_1(z)$ being spherical Bessel functions of the first kind,

$$z \cdot j_0(z) - j_1(z), \text{ which improves to } \cos(z).$$

- Hypergeometric expressions that are equivalent to expressions containing instead more familiar higher transcendental functions such as Bessel functions. For example,

$$\left(\frac{x}{2}\right) \cdot {}_0F_1 \left[\begin{matrix} \cdots \\ p+1 \end{matrix} ; -\frac{x^2}{4} \right], \text{ which is equivalent to } p! \cdot J_p(x).$$

These frauds are not equally heinous. I invite additions to the list together with opinions about their ranking (or rankness?)

It is important for default simplification to be as candid as is practical because:

- The consequences of misleading intermediate or final results can be ruinous. For example, not recognizing that an expression is equivalent to 0 or is free of a certain variable or is a polynomial of a particular degree can lead to incorrect matrix pivot choices, limits, integrals, series, and equation solutions.
- The need for identifying such properties occurs in too many places to require implementers and users to unfailingly employ a tryHarder(...) function at all of those places.
- If we want a candid result, the easiest way to implement that is to use bottom-up simplification and have every intermediate result also candid.
- A tryHarder(...) function probably entails at least one extra pass over the expression after default simplification, which wastes time, code space and expression space compared to making the first pass give a candid result. This can make simplification exponentially slower if tryHarder(...) function is used in a function that recursively traverses expression trees. Such recursion is so ubiquitous in computer algebra that this performance penalty precludes using tryHarder(...) in many of the places that it is needed.

In nontrivial cases it is impractical for a single form to reveal all possibly-important features of a function. Therefore it is unreasonable to insist that a candid form reveal all such features. However, a candid form at least shouldn't mislead us about those features.

Goal 6 (candid classes) *Default simplification should be candid for rational expressions and for as many other classes as is practical. Simplification should try hard even for classes where candidness can't be guaranteed for all examples.*

2.6 Canonical forms are necessary optional forms but insufficient default forms

Definition 5 (canonical form) *A canonical form is one for which all equivalent expressions are represented uniquely.*

With bottom-up simplification of an expression from its simplest parts, merely forcing a canonical form for every intermediate result guarantees that every operand is canonical. This makes the simplifier particularly compact, because there are fewer cases to consider when combining sub-expressions.

Table 2 lists examples and informational advantages of three canonical forms for rational expressions.

Factored form depends on the amount of factoring, such as square-free, over $\mathbb{Z}$, over $\mathbb{Z}[i]$, with algebraic extensions, with radicals, with rootOf(...), with approximate coefficients, etc. Partial-fraction form similarly depends on the amount of factoring of denominators. Moreover, for multivariate examples there are choices for the ordering of the variables and for which subsets of the variables are factored and/or expanded. Also, Stoutemyer (2008b) discusses alternative forms of multivariate partial fractions.

Table 2: Three canonical forms on the main spectrum for rational expressions

Form	Univariate example	Notable advantages
Factored on a common denominator	$\frac{x^3 \cdot (2x+\sqrt{5}-1)(2x-\sqrt{5}-1)}{4(x-1)^2(x+1)}$	zeros, poles, their multiplicities, often less roundoff
Expanded on a common denominator	$\frac{x^5-x^4-x^3}{x^3-x^2-x+1}$	degrees of numerator, degrees of denominator
Partial fractions	$x^2-\frac{1}{2(x-1)^2}-\frac{3}{4(x-1)}-\frac{1}{x+1}$	poles, their multiplicities, residues, asymptotic polynomial

There are also canonical forms for some classes of irrational expressions, such as some kinds of trigonometric, exponential, logarithmic, and fractional-power expressions.

However:

- No one canonical form can be good for all purposes.
- Any one canonical form can exhaust memory or patience to compute.
- Any and all canonical forms can be unnecessarily bulky or unnecessarily different from a user's input, masking structural information that the user would prefer to see preserved in the result. For example:

  Both the factored and expanded forms of candid $(x^{99}-y^{99}) \cdot (9x+8y+9)^{99}$ are much bulkier. As another example, both the common denominator and complete partial fraction forms of candid

  $$\frac{a^8}{a^9-1}+\frac{b^8}{b^9-1}+\frac{c^8}{c^9-1}+\frac{d^8}{d^9-1}$$

  are much bulkier.
- Users prefer default results to preserve input structure that is meaningful or traditional to the application, as much as possible consistent with candidness.

Thus canonical forms are too costly and extreme for good default simplification. Given *optional* transformation functions that return canonical forms, there is no need for default simplification to rudely force one of them or even the most concise of them.

Goal 7 (factored through partial fractions) *For rational expressions, the set of results returnable by default simplification should be a dense subset of all forms obtained by combining some or all factors of fully factored form or combining some or all fractions of complete multivariate partial fractions.*

Goal 8 (nearby form) *Default simplification should deliver a result that isn't unnecessarily distant from the user's input.*

2.7 Please don't try my patience!

If a lengthy computation is taking an unendurable amount of time, a user will terminate the computation, obtaining no result despite the aggravating wasted time.

Users have to accept that for certain inputs, certain optional transformations sometimes exhaust memory or patience. However, if default simplification also does this, then the system is useless for those problems. Thus with the availability of various optional transformations such as fully factored over a common denominator through complete or total partial fraction expansion, we should strive to return a candid form that can be computed quickly without exhausting memory. As an associated benefit of avoiding costly transformations when we can candidly do so, the result is likely to be closer than most other candid forms to the users input.

Definition 6 (guessed least cost) *A guessed least cost candid form is one derived such that when more than one supported transformation is applicable, one of those guessed to be least costly is selected.*

It is important that the time spent guessing which alternative is least costly is modest compared to actually doing the transformations. It is good if the guesses take into account not only the costs of the immediate alternative transformations, but also the cost of likely possible subsequent operations. For example, with the transformation

$$\frac{A}{B}+\frac{C}{D}\rightarrow\frac{A\cdot D+B\cdot C}{B\cdot D}$$

it is less costly at this step to avoid gratuitous expansion of $B\cdot D$. Moreover, this retained factorization is likely to reduce the cost of subsequently combining this result with other expressions.

> **Goal 9 (economy)** *Default simplification should be as economical of time and space as is practical.*

2.8 Idempotent simplification is highly desirable

Definition 7 (ephemeral) *An ephemeral function or operator is one that can produce a form that default simplification would alter.*

For example, most computer-algebra systems have a function that does a transformation such as

$$\text{integerFactor}(20)\rightarrow 2^2\cdot 5;$$

but if you enter $2^2\cdot 5$, it transforms to 20.

Results that would otherwise be ephemeral can be protected by returning a special form such as a list of factors or a string that doesn't readily combine with numeric expressions. Another alternative is to passively encapsulate $2^2\cdot 5$ in a functional form whose name is the null string. However, all of these alternatives are a nuisance to undo if you later want the result to combine with numeric expressions. For example, such protection can prevent $2\cdot\text{integerFactor}(20)$ from automatically transforming to either $2^3\cdot 5$ or to 40. More seriously, such protection can prevent $\text{integerFactor}(20)-20$ from automatically simplifying to 0. Moreover, the *invisible-function encapsulation* alternative is so visually subtle that many users won't realize there is anything to undo, and won't notice a dangerously non-candid sub-expression in their result.

This is the *ephemeral form dilemma*: You are *damned if you do* protect results that would otherwise be ephemeral, and you are *damned if you don't*.

Perhaps the ephemerality of integerFactor(...) could be candidly overcome at the expense of performance by extending the partially-factored semi-fraction form discussed in the sections 3 and 4 to apply also to rational numbers. Until then, whether protected or ephemeral, integerFactor(...) doesn't compose the same as most mathematical functions, such as sin(...) and ln(...). Perhaps for that reason traditional mathematics exposition uses natural language annotation in the accompanying text rather than a function for such purposes. Consequently it might be better for computer-algebra systems to use top-level commands for such transformations, such as

```
integerFactor 20;
```

so that user's couldn't unwisely compose them.

Non-ephemeral functions and operators should include the standard named mathematical operators, elementary functions and higher transcendental functions.

It would be nice if our default simplifier was flexible enough to recognize and leave unchanged all candid expressions. Until then we must accept ephemeral results from some functions that request transfomations into forms that default simplification doesn't recognize as candid.

Definition 8 (idempotent) *Simplification is idempotent for a class of input expressions if simplification of the result yields the same result.*

Without this property, a cautious user would have to re-simplify such results until they stop changing or enter a cycle.

Failure of idempotency is usually a sign that a result sub-expression was passively constructed where there should have been a recursive invocation of a simplification function. It can be disasterous, because it can cause a non-candid result.

> **Goal 10 (idempotency)** *Default simplification should be idempotent for all inputs composed of standard named mathematical functions and operators.*

3 Recursive partially-factored form

Factors, factors everywhere, with opportunities to spare.
— W.S. Brown

By default, the Altran computer-algebra system represents rational expressions as a reduced ratio of two polynomials that are individually allowed to range between fully factored and fully expanded. Distributed representation is used for each multinomial factor. This form is candid and the set of result forms that it can produce is a dense subset of the spectrum that it spans.

Brown (1974) explains why many factors arise with polynomials and rational expressions during operations such as addition, multiplication, gcd computation, differentiation, substitution, and determinates, then explains why it is important to preserve such factors and how to do so. Hall (1974) gives additional implementation details and compelling test results. Those articles are highly-recommended background for this one.

By default, Altran polynomials are expanded only when necessary to satisfy the constraints of the form. The resulting denominators are usually at least partially factored, and often the numerators are too, greatly reducing the total time spent expanding polynomials and computing their gcds. The results are also usually more compact than a ratio of two expanded polynomials.

Derive also uses partially-factored representation, but with recursive rather than distributed representation for sums. The opportunities for shared factors are thereby not confined to the top level. This can dramatically further reduce the result size, its distance from a user's input, the need for polynomial expansion, and the total cost of polynomial gcds.

The computer algebra embedded in the TI-92, TI-89, TI-Interactive, TI-Voyage 200 and TI-Nspire products has no official generic name. Therefore it is referred to here as TI-Math-Engine (2008). The implementation language is C rather than muLISP. Although there are algorithmic differences throughout, both systems use similar algorithms for the recursive partially-factored semi fraction forms described in the remaining sections.

3.1 Recursive representation of extended polynomials

The representational statements in the remaining sections are abstract enough so that they apply to both Derive and TI-Math-Engine. Stoutemyer (2008c) describes the extremely different concrete data structures used by both systems.

In the internal representation, negations and subtractions are represented using negative coefficients, and non numeric ratios are represented using multiplication together with negative powers.

Definition 9 (functional form) *A functional form is the internal representation of anything that isn't internally a number, variable, sum, product or rational power.*

For example, $\ln(\dots)$ is a functional form. The arguments of functional forms recursively use the same general representation.

Definition 10 (unomial) *A unomial is a variable, a functional form, or a rational of a variable or a functional form.*

The exponent can be negative and/or fractional. Non-numeric exponents are represented internally using $\exp(\dots)$ and $\ln(\dots)$. For example, x^y is represented as $\exp(y\cdot\ln(x))$. A post-simplification pass transforms it back to x^y for display. However, this representation has some disadvantages:

- Representing 2^n as $\exp(n\cdot\ln(2))$ is somewhat awkward.
- Representing $(-1)^n$ as $\exp(n\cdot\ln(-1)) \to \exp(n\pi i) \to \cos(n\pi) + i\sin(n\pi)$ is more awkward.
- Representing 0^z as $\exp(z\cdot\ln(0))$ is quite awkward.

Perhaps the exponents could be extended to use instead general expressions as exponents, but that alternative probably has its own set of difficulties.

A unomial is automatically candid if it is a variable, a power of a variable, or a functional form that that has candid arguments and doesn't simplify to a simpler class. Otherwise we should check for transformations that make the unomial more candid, such as $|x|^2 \mid x \in \mathbb{R} \to x^2$.

Definition 11 (unomial headed) *A unomial-headed term is a unomial or a unomial times a coefficient that is either a number or any candid expression having only lesser variables.*

A unomial-headed term is automatically candid if the unomial is a variable or a power thereof. For example, distributing such a unomial over the terms of the candid coefficient that is a sum couldn't enable any cancellations, because all of the terms in the coefficient are dissimilar to each other and have only lesser variables than the distributed unomial. However, if the unomial is a functional form or a power thereof, then we should check for possible cancellation or combination with functional forms in the coefficient. For example with ordering $|x| \succ \operatorname{sign}(x) \succ y$,

$$(\operatorname{sign}(x) + y)\cdot|x| \to y\cdot|x| + x,$$

which is more candid because the superfluous $\operatorname{sign}(x)$ has been eliminated.

Definition 12 (extended polynomial) *An extended polynomial is one of*

- *a number,*
- *a unomial-headed term,*
- *a unomial-headed term plus a number,*
- *a unomial-headed term plus a candid expression whose variables are less main than that of the unomial,*

- *a unomial-headed term plus an extended polynomial whose leading term has a greater variable or the same main variable and a greater exponent.*

Extended polynomials are automatically candid if the unomial for each unomial-headed term is a variable or a power thereof. For example, distributing these distinct unomials over their associated coefficients that are sums cannot enable cancellations: Distributed terms arising from different recursive terms will have distinct leading unomials. However, extended polynomials containing fractional powers or functional forms might require additional checks and transformations to achieve or strive for candidness, as explained in appendices C and D.

Here is an example of a recursive extended polynomial in $\ln(x)$, and y, with ordering $\ln(x) \succ y \succ z$:

$$\left(2y^{-5/2} + 3.27i\right)\cdot\ln(x)^2 + (5z-1)^{7/3}\cdot\ln(x) + (z+1)^{1/2}, \tag{1}$$

which displays as

$$\left(\frac{2}{y^{5/2}} + 3.27i\right)\cdot\ln(x)^2 + (5z-1)^{7/3}\cdot\ln(x) + \sqrt{z+1}\,.$$

Notice that this is *not* an extended polynomial in z, because of the fractional powers of $5z-1$ and $z+1$. However, the two sub-expressions involving z are candid as required for the entire expression to be candid.

The general-purpose Derive and TI-Math-Engine data structures are flexible enough to represent both recursive and distributed forms, and they use distributed form when needed for algorithmic purposes and for displaying the result of the expand(...) function when expanding with respect to more than one variable. However, expression 1 is as expanded as it can be for recursive representation.

Another example, with $x \succ y$ is

$$\frac{x^2}{y} + \left(y^2+y+5\right) + \frac{8y}{x}, \tag{2}$$

which is represented internally as

$$y^{-1}x^2 + 8y\cdot x^{-1} + y^2 + y + 5. \tag{3}$$

The terms that are 0-degree in the main variable x are artificially grouped as (y^2+y+5) in expression 2 only for emphasis. They are not collected under a single pointer internally. Notice how internally the term with lead unomial x^{-1} term occurs *between* the term with lead unomial x^2 and the (implicitly) x^0 terms in expression 3. This makes it faster to determine when the reductum of a sum is free of the previously-main variable. A minor disadvantage of this concession to efficiency is that distributing or factoring out negative-degree unomials can change the relative order of terms. For example,

$$x^{-1}\cdot\left(x^2+5x-7\right) \equiv x - 7x^{-1} + 5.$$

The more traditional ordering could be restored for display during a post-simplification pass.

3.2 Recursively partially factored representation

What is good for the goose is good for the goslings.

Polynomial distribution is often less costly with recursive form than with distributed form because:

- Unomials are shared by terms that differ only in lesser variables.
- At any one level the term count for multivariate sums tends to be much less, reducing the sorting costs.

Moreover, for many purposes distribution is often necessary only with respect to the top-level variable. Such partial distribution is possible for recursive representation, but not for distributed representation.

Nonetheless, co-distribution of two sums having the same main variable can be costly in time and usually also in the resulting expression size. Moreover, it is even more costly to recover the factorization. Therefore, effort is made to avoid such co-distribution wherever candidly possible.

One-way distribution of an expression over a sum is less costly and less costly to reverse.

With recursive representation we can employ partial factoring at all levels, with dramatic benefits. Even when some expansion is necessary to enable possible cancellations, the recursive representation might enable us to confine the expansion to certain levels. For example with recursive form and $x \succ a \succ b$,

$$\begin{aligned}((a^2-1)^{1000}x+b)x-bx &\rightarrow (a^2-1)^{1000}x^2+bx-bx\\ &\rightarrow (a^2-1)^{1000}x^2,\end{aligned}$$

eliminating the superfluous variable b.

In contrast, with distributed representation we would have

$$\begin{aligned}((a^2-1)^{1000}x+b)x-bx &\rightarrow (a^{2000}x-1000a^{1998}x+\cdots+x+b)x-bx\\ &\rightarrow (a^{2000}x^2-1000a^{1998}x^2+\cdots+x^2+bx)-bx\\ &\rightarrow a^{2000}x^2-1000a^{1998}x^2+\cdots+x^2,\end{aligned}$$

with 1001 terms, from which only the factor x^2 is easily recoverable.

At each recursive level it is helpful to order factors internally by decreasing mainness of their most main variable, with any signed numeric factor last. Ties are broken lexically, according to the bases of the factors. This makes the main variable most accessible. Also, when the main variable of a factor is less than for the previous factor, then the previous main variable won't occur from there on.

Ordering functional forms first according to the function or operator is advantageous for a few purposes such as recognizing opportunities for transformations such as, with $x \succ y \succ z$:

$$\begin{aligned}|z|\cdot y\cdot |x| &\rightarrow y\cdot|z|\cdot|x|\\ &\rightarrow y\cdot|z\cdot x|,\\ \ln(x)+y+\ln(z) \mid x>0 &\rightarrow \ln(x)+\ln(z)+y \mid x>0\\ &\rightarrow \ln(x\cdot z)+y \mid x>0.\end{aligned}$$

Such transformations can be helpful for limits, equation solving, and to reduce the number of functional forms for display during a post-simplification pass. However, for default internal simplification it is more helpful to order functional forms according to lexical comparison of their successive arguments, using the function or operator name only as a final tie breaker. This helps group together factors depending on the main variable. Therefore this is the order used by Derive and TI-Math-Engine.

It is also helpful to have any unomial factor immediately after all non-unomial factors having the same main variable. That way we can be sure that when the first factor of a product is a unomial then the rest of the product is free of the unomial's variable. This is a very common case because for fully expanded recursive extended polynomials, non-unomial factors can only occur as the last factor of all products.

A post-simplification pass can rearrange the factors to the more traditional display order described by Moses (1971).

3.3 Units and unit normal expressions

Polynomials over $\mathbb{Z}, \mathbb{Z}[i], \mathbb{Q}$ and $\mathbb{Q}[i]$ are unique factorization domains. However, to exploit that uniqueness we must uniquely represent sums that differ only by a unit multiple such as -1. This has the additional benefit of making syntactic common factors more frequent, reducing the need for polynomial expansion.

More seriously, not making the numerator and denominator multinomials unit normal in an example such as

$$\frac{((3-5i)\cdot z+1)\cdot((7+i)\,z+i)}{((5+3i)\cdot z+i)\cdot((1-7i)\,z+1)}$$

can prevent improving this expression to 1.

Definition 13 (leading numeric coefficient) *The leading numeric coefficient of an extended polynomial is*

- The polynomial if it is a number.
- Otherwise 1 if the polynomial is a unomial.
- Otherwise the leading numeric coefficient of the coefficient if the polynomial is a unomial-headed term.
- Otherwise the leading numeric coefficient of the leading term.

Definition 14 (unit normal over $\mathbb{Z}$) *An expression is unit normal over $\mathbb{Z}$ if its leading numeric coefficient is positive.*

If not, it can be made so by factoring out the unit -1.

Definition 15 (unit normal over $\mathbb{Q}$ and Gaussian rationals) *An expression is unit normal over $\mathbb{Q}$ or $\mathbb{Q}[i]$ if its leading numeric coefficient is* 1.

If not, it can be made so by factoring out the leading coefficient, which is a unit in these domains.

Definition 16 (unit normal over $\mathbb{Z}[i]$) *An expression is unit normal over $\mathbb{Z}[i]$ if for its leading numeric coefficient c, $-\pi/4 < \arg(c) \leq \pi/4$.*

If not, it can be made so by factoring out the unit -1 and/or the unit i. This is one of two alternative definitions motivated and described in more detail in Stoutemyer (2008d).

3.4 Recursive factorization of unit quasi content

We can further increase the likelihood of syntactic common factors by factoring out their *quasi content*:

Definition 17 (quasi content) *The quasi content of a recursive partially-factored sum is the product of the least powers of all syntactic factors among its terms, multiplied by the gcd of the numeric factors of those terms. The quasi content is computed and factored out level by level, starting with the least main variables and functional forms.*

Definition 18 (unit quasi content) *The unit quasi content of a multinomial is the product of its quasi content and the unit that is factored out to make the multinomial unit normal.*

Definition 19 (quasi primitive) *A recursive partially factored sum that has its quasi content factored out at all levels is quasi primitive.*

Definition 20 (unit quasi primitive) *A recursive sum is unit quasi primitive if it is unit normal and quasi primitive at every level.*

For example with $x \succ a \succ b$,

$$-6a^2bx + 6a^2x + 6b^2x - 6x + 8a^2 \cdot (a+b+9)^9 - 8(a+b+9)^9$$

$$\overset{recursive}{\longrightarrow} \left(\left(-6b^2+6\right)a^2 + \left(6b^2-6\right)\right)x + 8\left(a^2-1\right)(a+b+9)^9$$

$$\rightarrow \left(-6\left(b^2-1\right)a^2 + 6\left(b^2-1\right)\right)x + 8\left(a^2-1\right)(a+b+9)^9$$

$$\rightarrow -6\left(\left(b^2-1\right)\right)\left(a^2-1\right)x + 8\left(a^2-1\right)(a+b+9)^9$$

$$\rightarrow -2\left(a^2-1\right)\left(3\left(b^2-1\right)x - 4(a+b+9)^9\right).$$

Not only have we preserved the entered internal factor $(a+b+9)^9$; we have also discovered another internal factor b^2-1 and a top-level factor a^2-1. In contrast, distributed partially factored form can't represent the internal factors, and it would discover only the top-level numeric content of 2. Worse yet, the unavoidable forced expansion of $(a+b+9)^9$ would add many more terms, with many non-trivial coefficients.

Determining minimum degrees of syntactic factors requires a number of base and exponent comparisons that are each bounded by the number of non-numeric syntactic factors in the recursive form. Determining the units to factor out of the leading coefficients at each level requires less work. Determining the mutual gcd of n numeric coefficients is $n-1$ sequential gcds that start with the first coefficient magnitude and decrease from there. This is significantly faster than $n-1$ independent gcds between those coefficients, because whenever the net gcd doesn't decrease much, few remainders were required, whereas whenever the net gcd decreases substantially, fewer remainders are required for subsequent gcds.

At each level, if a unit quasi content isn't 1, factoring it out requires effort proportional to the number of top-level terms at that level, plus some possible effort for numeric divisions. Thus the overall cost of making a polynomial unit quasi primitive is at most a few times the cost of thoroughly distributing the units and quasi contents back over the terms as much as is allowed with recursive representation.

3.5 Demand-driven extraction of signed quasi content

We want to represent and operate directly on both recursively expanded and partially factored expressions. The recursive sums are candid regardless of whether they are unit quasi primitive or not. They can even have signed quasi contents fortuitously factored out at some of the deepest levels but not at shallower levels, with the quasi-primitive boundary different for different terms.

The default policy of least guessed cost together with the desire to represent and operate on recursively expanded polynomials as well as on partially factored ones indicates that we should not automatically unit quasi-primitize a sum result. The next operation, if any, might force partial or total redistribution. Instead we can postpone unit quasi-primitization until we guess that it is the least costly alternative.

For a fully expanded recursive extended polynomial, sums can occur only at the top level and/or as the last factor in products. Therefore it is helpful to require that all powers of sums and all products containing sums anywhere else are unit quasi-primitive. This makes similar factors more likely and makes it easy to infer that such sums are already unit quasi-primitive. Powers and/or products containing unit quasi-primitive sums are efficiently and candidly multiplied merely by merging and combining similar factors. Also, similar factors having such sums as bases can be extracted to reduce the amount of expansion when adding two powers or products.

When a sum or a power thereof that isn't the last factor in a product thus implies that all sums in the product are unit quasi primitive, this information can be passed down recursively so that recursions that treat terminal sum-factors will know that they are unit quasi primitive without having to traverse the data to verify that.

It is possible to store information about known unit quasi primitiveness or other factorization levels with each sum. However, consistently managing such information substantially complicates the implementation. Moreover, this approach increases the data size — particularly if done at every recursive level, as it has to be for full effectiveness. Therefore Derive and TI-Math-Engine instead re-determine such properties when they can't easily be inferred or passed as a flag to functions that can exploit the information. Even without such information, the time it requires to determine that a sum is already unit quasi primitive is less than the time that it requires to unit quasi-primitize it if it isn't. Moreover, the mean time it requires to determine that a sum *isn't* unit quasi primitive is even less.

4 Recursively partially-factored semi-fractions

Fractions, fractions everywhere, with opportunities to share.

Common denominators can enable cancellations that reduce degrees, eliminate denominators, or eliminate variables. For example,

$$\frac{1}{x-1}-\frac{1}{x+1}-\frac{2}{x^2-1}\xrightarrow{:)}0$$

However, common denominators can be costly in computing time and in the size of the result. For example,

$$\begin{aligned}&\frac{a}{a-1}+\frac{b}{b-1}+\frac{c}{c-1}+\frac{d}{d-1}\\&\qquad\rightarrow\frac{(((4d-3)\,c-3d+2)\,b-(3d-2)\,c+2d-1)\,a-((3d-2)\,c-2d+1)\,b+(2d-1)\,c-d}{(a-1)\cdot(b-1)\cdot(c-1)\cdot(d-1)}.\end{aligned}$$

This is *the common denominator dilemma*: You might be *damned if you do* force common denominators, but you might be *damned if you don't* force common denominators.

Also, users feel too constrained if all of their default results have a common denominator except perhaps polynomials over $\mathbb{Q}$ or $\mathbb{Q}[i]$.

In contrast to Altran, Derive and TI-Math-Engine use a candid form for extended rational expressions that also accommodates at each recursive level either a reduced ratio of two partially-factored extended polynomials or a sum of an extended polynomial and any number of proper ratios of extended polynomials having denominators whose mutual gcds are numeric. The denominators do not need to be square-free or irreducible, and they together with the polynomial part and the numerators of the fractions can be partially factored. This *partially-factored semi fraction form* thus flexibly extends the Altran spectrum through partial fractions. Regarding rational expressions and the rational aspects of irrational expressions, the broad spectrum from factored over a common denominators through partial fractions accommodates most of the result forms often wanted by most users for default simplification.

4.1 Common denominators ≡ quasi-primitation

Recursive form with negative exponents leads to the insight that combining expressions over a common denominator is simply quasi-primitation, which is a mild form of factorization. For example,

$$\begin{aligned}\frac{A}{B}+\frac{C}{D}&\rightarrow A\cdot B^{-1}+C\cdot D^{-1}\\&\rightarrow (A\cdot D+B\cdot C)\cdot B^{-1}\cdot D^{-1}\\&\rightarrow \frac{A\cdot D+B\cdot C}{B\cdot D}.\end{aligned}$$

The only additional responsibility for negative exponents of sums is to check for possible gcd cancellations between dissimilar sum factors raised to positive and negative multiplicities.

Definition 21 (extended rational expression) *Extended rational expressions are composed of extended polynomials and ratios of integer powers of extended rational expressions.*

Making an extended rational expression unit quasi primitive and canceling multinomial gcds between numerators and denominators makes it candid: Quasi-primitation recursively factors out the lowest occurring degree of syntactically similar factors, forcing a common denominator and making the exponents all positive within multinomial factors. This together with the multinomial gcd cancellation guarantees that the total apparent degree for each variable or functional form is the actual degree. That in turn guarantees that there are no superfluous variables and that polynomials don't appear to be more general rational expressions.

4.2 Extended Polynomials with extended rational expressions as coefficients

As discussed in section 3.1, the coefficients in a candid extended polynomial can be any candid expressions having only lesser variables. This includes candid extended rational expressions.

Thus recursive form easily accommodates expressions that are extended polynomials having coefficients that are candid extended rational expressions in lesser variables, such as for $x \succ y \succ z$:

$$\left(y^2 + \frac{z^2}{2z+1}y + \frac{7}{z+4}\right)\cdot x^2 + \left(\frac{1}{y}\right)\cdot x + \left(\frac{3}{y^2+5}\right).$$

This form is candid despite the lack of a common denominator, because the coefficients (including those of implicit y^0 and x^0, are all candid expressions in lesser variables.

4.3 The importance of proper ratios

Definition 22 (term) *A term is any expression that isn't a sum at the top level.*

Definition 23 (sum-headed term) *A sum-headed term is a term whose leading factor is a sum or a power thereof.*

It is helpful to order terms in a sum primarily according to their main variables. Among terms having the same main variable it is helpful to order the sum-headed terms first so that the presence of sum-headed terms in the main variable is more quickly determined. Among sum-headed terms having the same main variable, it is important to order the terms in some well-defined and easily computed order.

In contrast, rational expressions are traditionally displayed with the polynomial part left of any proper fractional parts. A post-simplification pass can be used to display such terms in this more traditional order.

Definition 24 (proper) *A term is proper if it isn't sum headed or if for its main variable the degree of its numerator is less than the degree of its denominator.*

An improper sum-headed term can be made proper by using division with remainder with respect to its main variable. This transforms the term into an extended polynomial in that variable plus a proper sum-headed term in that variable.

We have already seen that an *improper* term can candidly coexist with a unomial-headed terms having greater main variables.

Regardless of the variables therein, a *proper* term can always candidly coexist with unomial-headed terms and/or a numeric term.

If an *improper* term would order before another term, then we can either force a common denominator for those two terms or make the ratio proper to allow possible important cancellations. For example, using the internal ordering of terms with $x \succ y \succ b$,

$$\begin{aligned} x + \frac{b\cdot y + b + 1}{y+1} + y - b \quad &\rightarrow \quad x + \frac{y^2+y+1}{y+1} \\ \text{or} \quad &\rightarrow \quad x + \left(b + \frac{1}{y+1}\right) + y - b \\ &\rightarrow \quad x + \frac{1}{y+1} + y, \end{aligned}$$

either of which eliminates the superfluous variable b.

Unfortunately, making a ratio proper can contract the domain of equivalence by introducing singularities if the leading coefficient of the denominator isn't constant. For example, using the internal ordering of terms with $x \succ a$,

$$\frac{x}{a\cdot x - 1} \overset{:(}{\rightarrow} \frac{1}{a\cdot(a\cdot x-1)} + \frac{1}{a}.$$

At $a = 0$ the left side simplifies to x, whereas the right side is undefined. Also, for approximate arithmetic the right sides is more prone to underflow, overflow and catastrophic cancellation near $a = 0$.

In contexts such as where integration specifically requests a proper fraction, we can use a piecewise result such as

$$\begin{aligned}\int \frac{x}{ax-1}\,dx \quad &\rightarrow \quad \int \text{if}\left(a=0 \ \text{ then } \ x \ \text{ else } \ \frac{1}{a\cdot(ax-1)}+\frac{1}{a}\right)dx\\ &\rightarrow \quad \text{if}\left(a=0 \ \text{ then } \ \frac{x^2}{2} \ \text{ else } \ \frac{\ln(ax-1)}{a^2}+\frac{x}{a}\right).\end{aligned}$$

Otherwise, to avoid the clutter and difficulties of simplifying expressions containing piecewise expressions, default simplification should use the common denominator choice if the leading coefficient of the denominator could be 0 for some values of the variables therein within the domain of interest.

If a sum-headed term is proper, then it is candid to have it in a sum with unomial-headed terms and a numeric term. For example, if the user *entered*

$$\frac{1}{a\cdot(a\cdot x-1)}+\frac{1}{a}$$

with $x \succ a$, then we could return that as a candid result. However, the gcd of the denominators is non-numeric. Therefore default simplification combines them over a common denominator, which in this case improves the expression to

$$\frac{x}{a\cdot x-1},$$

making the result defined at $a = 0$.

Reduced proper sum-headed terms having different main variables can candidly be joined together as a sum: Effectively the proper fraction having the less-main variable is a degree-0 term of a polynomial part accompanying the proper fraction having the greater main variable. However, here too there are opportunities for improving the expression by combining two such terms when the gcd of the denominators isn't numeric. For example with $x \succ y \succ a$,

$$\frac{-1}{a\cdot(a\cdot x+1)}+\frac{1}{a\cdot(a\cdot y+1)} \overset{:)}{\rightarrow} \frac{x-y}{(a\cdot x+1)\cdot(a\cdot y+1)},$$

which makes the result defined at $a = 0$.

4.4 Sums of ratios having the same main variable

Even if their main variables are the same, proper ratios can candidly be joined together as a sum if the gcd of their denominators is numeric. In contrast, there might be important cancellations between sums of *improper* ratios even if the gcd of their denominators is numeric. For example:

$$\begin{aligned}\frac{a\cdot x+a+1}{x+1}-\frac{a\cdot x-a-1}{x-1} \quad &\rightarrow \quad \left(\frac{1}{x+1}+a\right)-\left(-\frac{1}{x-1}+a\right)\\ &\rightarrow \quad \frac{1}{x+1}+\frac{1}{x-1},\end{aligned}$$

which eliminates the superfluous variable a. Therefore a good default is to combine such ratios over a common denominator if it removes a singularity or if making the ratios proper requires a piecewise result. Otherwise make the ratios proper.

There can also be important cancellations between the sum of two ratios A/B and C/D if the gcd G of their denominators is non-numeric. One alternative is to combine such ratios; and that is what Derive and TI-Math-Engine do. However, if the main variable of G is the same as that of B and D, then we can instead:

- Split A/B into an extended polynomial part and two proper semi fractions having denominators G and B/G.

- Split C/D into an extended polynomial part and two proper semi fractions having denominators G and D/G.
- Combine the extended polynomial parts and the numerators of the fractions having denominator G, then passively merge that result with the passive sum of the ratios having denominators B/G and D/G.

If G is small compared to both B and D, then splitting is more likely to give a less bulky result than combining. Here is a borderline example:

$$\begin{aligned}\frac{x^2+x-3}{(x^2-1)(x-2)} - \frac{2x}{(x-2)(x+2)} &\rightarrow -\frac{x^2-x-3}{(x^2-1)(x+2)} \\ \text{or} &\rightarrow \left(\frac{1}{x^2-1}+\frac{1}{x-2}\right)-\left(\frac{1}{x-2}+\frac{1}{x+2}\right) \\ &\rightarrow \frac{1}{x^2-1}-\frac{1}{x+2}.\end{aligned}$$

Notice how $1/(x^2-1)$ wasn't split. There was no need to split it.

Splitting a proper fraction into semi fractions can introduce singularities if the leading coefficient of the given denominator can be 0 in the domain of interest. For example,

$$\frac{2x}{a^2x^2-1} \rightarrow \text{ if } \left(a=0 \text{ then } -2x \text{ else } \frac{1}{a\cdot(a\cdot x-1)}+\frac{1}{a\cdot(a\cdot x+1)}\right).$$

Combining fractions is a better default in such cases or when combining fractions eliminates a singularity.

5 Summary

There are at least ten worthy goals for default simplification:

I. **(correctness)** Within the domain of interest, default simplification should produce an equivalent result wherever the input is defined.

II. **(contraction prevention)** If necessary for equivalence where the input is defined in the domain of interest, a result should be piecewise or the system should append an appropriate constraint to the input, preferably after querying the user.

III. **(enlargement prevention)** Results should optionally include appropriate constraints if necessary to prevent enlarging the domain of definition within the domain of interest.

IV. **(domain propagation)** Input domains and constraints should be propagated into results, where they then cause substitution of inappropriate values to return the representation for undefined.

V. **(disable transformations)** It should be possible to selectively disable default transformations.

VI. **(candid)** Default simplification should be candid for rational expressions and for as many other classes as is practical. Simplification should try hard even for classes where candidness can't be guaranteed for all examples.

VII. **(factored through partial fractions)** For rational expressions, the set of results returnable by default simplification should include a dense subset of all forms obtained by combining some or all factors of fully factored form or combining some or all fractions of complete multivariate partial fractions.

VIII. **(nearby form)** Default simplification should deliver a result that isn't unnecessarily distant from the user's input.

IX. **(economy)** Default simplification should be as economical of time and space as is practical.

X. **(idempotency)** Default simplification should be idempotent for all inputs composed of standard named mathematical functions and operators.

Derive and TI-Math-Engine implement a partially-factored semi fraction form and associated default simplification algorithms. For the rational expression aspect of general expressions, the form and algorithms can produce a dense subset of the candid spectrum from fully factored over a common denominator through multivariate partial fractions. Moreover, the default simplification attempts to avoid unnecessary cost and to produce a result in this spectrum that is not unnecessarily distant from the user's input.

Appendices

A Very-proper ratios

Definition 25 *A sum-headed term of the form $\frac{N}{D^m}$ with main variable x and $m \geq 1$ is very proper if the degree of x in N is less than the degree of x in D.*

For fractions having denominators that are the same polynomial raised to different powers:

- They can candidly coexist if they are all very proper.
- Otherwise we should either combine the terms or further split them so that all of them are very proper. For example,

$$\begin{aligned} \frac{x+2}{(x+1)^2} - \frac{1}{x+1} \quad &\rightarrow \quad \frac{(x+2)-(x+1)}{(x+1)^2} \rightarrow \frac{1}{(x+1)^2} \\ \text{or} \quad &\rightarrow \quad \left(\frac{1}{(x+1)^2} + \frac{1}{x+1}\right) - \frac{1}{(x+1)} \rightarrow \frac{1}{(x+1)^2}. \end{aligned}$$

Further expansion of proper ratios into very-proper ratios often increases total bulk. However, some algorithms such as integration sometimes require very-proper ratios. Default Derive and TI-Math-Engine simplification combine ratios for which the gcd of the denominators is non-numeric. Therefore although very-proper fractions are produced by the optional expand(...) function and when needed for purposes such as integration, they can be ephemeral.

B Preserving recursively primitive factors in reduced ratios

Definition 26 (content) *The content of a recursively-represented multinomial is the gcd of its top-level coefficients.*

Definition 27 (recursively primitive) *A recursively-represented multinomial is recursively primitive if its content is* 1 *at every recursive level.*

To recursively primitize a polynomial: At each recursive level of each quasi-primitive multinomial factor, starting with the deepest levels, factor out the gcd of the coefficients. This might entail non-trivial polynomial gcds if any coefficients have multinomial factors.

Most polynomial gcd algorithms and many factoring algorithms either require or benefit from further factoring a quasi-primitive multinomial into a primitive polynomial times a content, and from recursively making that content primitive with respect to its main variable, etc. Therefore as a side effect of primitizing the numerator and denominator to assist computing their gcd, the immediate result of the reduction is that every factor in the reduced result is recursively primitive with respect to its main variable. This knowledge can save significant time when the ratio is combined with another expression or when further factorization is desired. For example:

- We can skip the primitization step on the numerator or denominator when computing its gcd with another polynomial.
- If two primitive polynomials have different main variables, then they are relatively prime, allowing us to avoid computing their gcd.
- If a polynomial is primitive in a variable and linear in that variable, then the polynomial is irreducible, allowing us to avoid a futile attempt at gcds or further factorization.

Also, primitation further increases the chances of syntactically similar factors that can be combined and shared.

Primitation involves gcds of polynomials having fewer variables than the original quasi-primitive multinomial, and the cost of multinomial gcds generally grows rapidly with the number of variables. For this reason and reasons similar to computation of the numeric content, primitation is often less costly than computing the gcds between the resulting primitive polynomials. In fact, it is worth considering the primitated coefficients in order of increasing complexity so that their iteratively updated gcd is likely to approach 1 more quickly. For quasi-primitive multinomials, the content must be multinomial, and any variable not present in all of the coefficients can't occur in the final content. Therefore the multinomial part of the content is 1 if the intersection of the variables occurring in the multinomial factors of the coefficients, S, is the empty set. Also, we can substitute judicious numeric values for variables not in S. For these reasons, recursive primitation can be worth the investment in some circumstances even when not needed for ratio reduction.

The fact that default simplification leaves the numerators and denominators of ratios recursively primitive when they have sums in their denominators means that if the user requests an expanded numerator and/or denominator, it might be ephemeral. However, his is alright, because:

- Primitive factorization is generally preferable in most respects.
- In the rare cases where a fully-expanded numerator and/or denominator is helpful, such as facilitating some default and optional transformations for fractional powers and functional forms as described in appendices C and D, such transformations can be facilitated by a provisional expansion followed by re-primitation if any such transformation then occur.

C Additional considerations for fractional exponents

Hearn and Loos (1973) remark that quotients, remainders, gcds and many other polynomial operations can be well defined for fractional exponents of variables. For division and gcds we want non-negative exponents, and quasi-primitation accomplishes that. As examples of division and gcds for such extended multinomials,

$$\frac{z-1}{z^{1/2}+1} \xrightarrow{:)} z^{1/2}-1,$$

and

$$\gcd(x-1, x+2x^{1/2}+1) \rightarrow x^{1/2}+1.$$

Polynomial remainder sequence gcd algorithms require no change. However, any polynomial division or gcd algorithm that relies on substituting numbers for variables should first temporarily substitute for any variable x that has fractional exponents in either polynomial, a new variable $t^{1/g}$, where g is the gcd of all the occurring exponents of that variable in both polynomials. (The gcd of two reduced fractions is the gcd of their numerators divided by the least common multiple of their denominators. Even for all integer exponents this division and gcd isomorphism can have the advantage of reducing the degrees, which is important to algorithms that substitute numbers for variables.)

Regarding factoring, allowing the *introduction* of fractional exponents of variables makes factoring non-unique and not very useful. For example, we could factor $x-1$ into $(x^{1/2}-1)\cdot(x^{1/2}-1)$ or into $(x^{1/3}-1)\cdot(x^{2/3}+x^{1/3}+1)$ or into an infinite number of different such products. Instead, we should bias the partially factored form to expand by default when fractional powers of a variable might thereby be eliminated or have the least common multiple of their denominators reduced.

Common denominators can similarly help eliminate fractional powers. For example,

$$\frac{1}{z^{1/2}-1}-\frac{1}{z^{1/2}+1}\rightarrow\frac{2}{z-1}.$$

Thus it is also worth biasing toward common denominators when fractional powers might thereby be eliminated or reduced in severity.

Collecting similar factors that are fractional powers can enlarge the domain of definition for variables that are real by declaration or default. For example,

$$x^{1/2}\cdot x^{1/2}\mid x\in\mathbb{R}\overset{:)}{\rightarrow}x\mid x\in\mathbb{R}$$

enlarges the domain from $x\geq 0$ to all x. Thus with domain-enlargement protection enabled, the result would be $x\mid x\geq 0$. If the user is also using the real branch of fractional powers having odd denominators, such as $(-1)^{1/3}\rightarrow -1$, then we should append the constraint only if for some base u, fractional powers having an even denominator entirely disappear in the result.

Fractional powers of numbers, powers, products and sums involve additional complications that can be superimposed on the algorithms for extended rational expressions over $\mathbb{Z}[i]$. For example, there are additional considerations such as de-nesting and rationalization of denominators or numerators. Also, for internal simplification it is helpful to distribute powers over products and to multiply the exponents of powers of powers. However, it is not always correct to do so for fractional powers without including a rotational correction factor. Two always-correct principal-branch rewrite rules for exponents are

$$\begin{aligned}(z\cdot w)^{\alpha} &\rightarrow (-1)^{(\arg(z\cdot w)-\alpha\cdot\arg(z)-\alpha\cdot\arg(w))/\pi}\cdot z^{\alpha}\cdot w^{\alpha}, &(4)\\ \left(z^{\beta}\right)^{\alpha} &\rightarrow (-1)^{\left(\arg\left(z^{\beta}\right)-\beta\cdot\arg(z)\right)\cdot\alpha/\pi}\cdot z^{\beta\cdot\alpha}. &(5)\end{aligned}$$

Depending on any declared realness or intervals for $\arg(z)$ and $\arg(w)$, the simplified exponent of -1 tends to be quite complicated unless it simplifies to a rational constant. Therefore, transformations based on these identities are usually a bad idea unless that happens, as it always does for $z\geq 0$ or for integer α.

To maximize opportunities for exploiting these identities, it is generally best to factor multinomial radicands over $\mathbb{Z}$ or $\mathbb{Z}[i]$. Often, this is enough to extract at least a numeric factor from a radicand.

D Additional considerations for functional forms

It is helpful to force the arguments of a functional form to a particular canonical form that can depend on the set of optional or default rewrite rules for the function or operator.

Expanded arguments are a good choice for functions or operators that have a desired rewrite rule for arguments that are sums or numeric multiples. For example,

$$\begin{aligned}\exp(u+v) &\rightarrow \exp(u)\cdot\exp(v), &(6)\\ \exp(nu) &\rightarrow (\exp(u))^{n}, &(7)\\ \sin(u+v) &\rightarrow \sin(u)\cdot\cos(u)+\cos(u)\cdot\sin(v), &(8)\\ \sin(2u) &\rightarrow 2\sin(u)\cdot\cos(u), &(9)\\ \int(u+v)\,dx &\rightarrow \int u\,dx+\int v\,dx. &(10)\end{aligned}$$

Even if the rewrite rule is optional rather than default, expanded arguments relieve users from having to explicitly request the expansion. Moreover, expanded arguments suggest the applicability of the optional rules.

For analogous reasons, a canonical factored form is a good choice if the function or operator has a rewrite rule for products or powers in one of its arguments, such as

$$\begin{aligned} |u\cdot v| &\rightarrow |u|\cdot|v|, &(11)\\ |u^k| &\rightarrow |u|^k. &(12)\end{aligned}$$

In the absence of either kind of rewrite rule, it is nonetheless important to force the arguments to some one canonical form. Otherwise, opportunities for collecting and canceling similar factors or terms can be lost, leading to a non-candid result. For example, we want

$$f\left(x^2-1\right) - f\left((x-1)\cdot(x+1)\right) \overset{:)}{\Rightarrow} 0$$

for any $f(\ldots)$.

For the arguments of functional forms such as $f(\ldots)$ we could choose a canonical form that tends to be compact and not too costly to compute, such as square-free factored form or square-free partial fractions. However, sub-expressions outside functional forms rarely move inside them. Consequently argument size tends to be small compared to top-level extended rational expressions containing those functional forms. Therefore, a fully factored or fully expanded form over $\mathbb{Z}$ is rarely costly for functional form arguments. Moreover, for internal representation it is most often helpful instead to move as much of the arguments as possible *outside* functional forms, which increases the chance of them simplifying away.

Rewrite rules for powers or products of functional forms must be superimposed on the algorithms for the partially-factored semi fraction form. For example, consider the rules

$$\begin{aligned} \cos(u)^2 &\rightarrow 1-\sin(u)^2,\\ \sin(u)\cdot\cos(v) &\rightarrow \frac{\sin(u-v)+\sin(u+v)}{2}.\end{aligned}$$

Opportunities for using such rules are easier to recognize and exploit if the default is biased toward common denominators and expanding products and powers of sums when they contain appropriate sinusoids. For example,

$$\begin{aligned} \frac{\sin(x)}{\cos(x)+1}+\frac{\sin(x)}{\cos(x)-1} &\rightarrow \frac{2\sin(x)\cdot\cos(x)}{(\cos(x)+1)\cdot(\cos(x)-1)}\\ &\rightarrow \frac{2\sin(x)\cdot\cos(x)}{\cos(x)^2-1}\\ &\rightarrow \frac{2\sin(x)\cdot\cos(x)}{-\sin(x)^2}\\ &\rightarrow \frac{-2\cos(x)}{\sin(x)},\end{aligned}$$

which the post-simplification pass could display as $-2\cot(x)$.

Even where such rules are optional rather than default, expanding over a common-denominator makes the opportunities more obvious to users.

Rewrite rules for sums of functional forms must also be superimposed on the algorithms for the partially factored form. For example, consider the always-correct principal-value rewrite rule

$$\ln(u\cdot v) \rightarrow \ln(u)+\ln(v)+(\arg(u\cdot v)-\arg(u)-\arg(v))\cdot i. \tag{13}$$

Opportunities for using such rules are easier to recognize and exploit if the default is biased toward factoring sums when they contain such functional forms. For example,

$$\begin{aligned}\frac{\ln(2z)^2-1}{\ln(z)+\ln(2)+1} &\rightarrow \frac{(\ln(z)+\ln(2))^2-1}{\ln(z)+\ln(2)+1}\\ &\rightarrow \frac{(\ln(z)+\ln(2)+1)\cdot(\ln(z)+\ln(2)-1)}{\ln(z)+\ln(2)+1}\\ &\overset{:)}{\rightarrow} \ln(z)+\ln(2)-1.\end{aligned}$$

If unomials have functional forms as bases, then rewrite rules between functional forms might require additional checks to make sure that recursive form is candid. For example with $x \succ y \succ \cos(z)$,

$$\left(y+\cos(z)^2\right)\cdot x+\sin(z)^2 x$$

complies with recursive representation. However, for candidness it should be transformed to $(y+1)\cdot x$.

One approach to simplifying expressions containing functional forms is to exploit dependency theorems such as the Risch structure theorem (1979). The basic idea is: Each time you combine two simplified expressions both containing functional forms, you set up then attempt to solve a system of equations. If there is no solution, then all of the functional forms are independent and can candidly coexist. Otherwise the solution indicates how to represent a subset of the functional forms in terms of the other functional forms. By itself, this method doesn't prescribe which subset to use as a basis, so it isn't canonical. Also, functional forms can meet each other many times during the course of simplifying an input, so there can be many times requiring a complete scan of both operands to set up then solve the equations — perhaps the same set that has already been considered for a different sub-problem.

An approach that tends to avoid these difficulties is to use rewrite rules that move as much of the arguments as possible outside the arguments of functional forms, driving them toward canonicality. For example, all of the numbered rewrite rules in this appendix and the one that precedes it are of that type. However, for output, results are often more concise if the number of functional forms is reduced by using such rewrite rules in the opposite direction during a post-simplification pass for display.

Special attention must also be given to infinities and multi-valued expressions. For example, we don't want either $\infty-\infty$ or $\pm1-\pm1$ to simplify to 0.

E Multiplying and adding partially-factored semi fractions

This appendix contains pseudo code for the top-level functions that simplify products and sums of recursively partially-factored semi fractions.

Definition 28 (polynomially expandable) *A term is polynomially expandable if it is sum headed and has as least one factor that is either a sum containing the main variable or a positive integer power of such a sum, and contains no negative integer powers of a sum containing that main variable.*

Operators such as "^", "·" or "+" in quotes designate nearly-passive construction of the corresponding data structures from the operands: The only simplifications for such nearly-passive operations are that 0 and 1 identities are exploited and operands are merged lexically.

When quotes are omitted from the operators, it designates an invocation of the corresponding active simplification function. For brevity, additional considerations for domain enlargements and for irrational or multi-valued operands are omitted. Also, experienced implementers will notice places where efficiency can be improved at the expense of concise clarity. For example, some recursion and redundant computations can be avoided by swapping operands and using *knownFactorLevel* parameters.

Variables that aren't formal parameters are local. Numeric elements can be any mixture of elements of $\mathbb{Z}, \mathbb{Q}, \mathbb{Z}[i], \mathbb{Q}[i]$, and approximate numbers. They could also be multi-intervals as recommended in Stoutemyer (2007), but aren't in the current implementation.

```
function $E_1 \cdot E_2$:
{ if $E_1$ is numeric, then
   { if $E_2$ is numeric, then
         return their numeric product;   // Floating-point is locally infectious
      if $E_2$ is a non-sum or a unit quasi-primitive sum, then
         return leadFactor($E_2$)“·”($E_1 \cdot$remainingFactors($E_2$);
      if it seems significantly easier to distribute $E_1$ than to make $E_2$ unit quasi primitive, then
         return distribute($E_1$,$E_2$,mainVariable($E_2$));          //Eg: common denominators are hard
      return $E_1 \cdot$ unitQuasiPrimitate($E_2$);
   }
   if $E_2$ is numeric, then return $E_2 \cdot E_1$;
   $L_1 \leftarrow$ leadFactor($E_1$);                    // leadFactor($E_1$) $\rightarrow E_1$ if $E_1$ is a non-product
   $R_1 \leftarrow$ remainingFactors($E_1$);            // remainingFactors($E_1$) $\rightarrow 1$ if $E_1$ is a non-product.
   $L_2 \leftarrow$ leadFactor($E_2$);    $R_2 \leftarrow$ remainingFactors($E_2$);
   if $L_1$ is similar to $L_2$, then              // The lead bases are identical
      return leadBase($L_1$)$^{\text{leadExponent}(L_1)+\text{leadExponent}(L_2)} \cdot (R_1 \cdot R_2)$;
   if leadBase($L_2$) $\succ$ leadBase($L_1$) then return $E_2 \cdot E_1$;
   $P \leftarrow R_1 \cdot E_2$;
   if $L_1$ is a unomial then return $L_1$“·”$P$;
   // $L_1$ is a sum or a power of a unit quasi-primitive sum:
   $x \leftarrow$ mainVariable($L_1$);
   if $L_1$ and/or $P$ are not unit quasi primitive and it seems significantly harder
         to make them so than to coDistribute $L_1$ with $P$ with respect to $x$, then
      return coDistribute($L_1, P, x$);
   if $P$ isn't unit quasi primitive, then return $L_1 \cdot$ unitQuasiPrimitize($P$);
   if $L_1$ isn't unit quasi primitive, then return unitQuasiPrimitize($L_1$) $\cdot P$;
   if $L_1$ has a negative power of a sum and $P$ has a positive power of a sum or vice versa,
   { if context doesn't guarantee that $P$ is primitive, then
         $P \leftarrow$ primitize($P$);                  // $P \leftarrow$ primitize(content($P$))$\cdot$primitivePart($P$)
      if context doesn't guarantee that $L_1$ is primitive, then
         $L_1 \leftarrow$ primitize($L_1$);
      return productOfPrimitives($L_1, P$);     // Henrici-Brown algorithm (Knuth 1998)
   }                                                     // See also Brown (1974) and Hall (1974)
   if $L_1$ is a product, then return $L_1 \cdot P$;      // Primitation might make $L_1$ be a product.
   return $L_1$“·”$P$;
} // end function ·
```

```
function $E_1 + E_2$:
{ if $E_1$ is numeric, then
   { if $E_2$ is numeric, then return their numeric sum;
      return leadTerm($E_2$) + ($E_1$ + reductum($E_2$));
   }
   if $E_2$ is numeric, then return $E_2 + E_1$;
   $L_1 \leftarrow$ leadTerm($E_1$);      $R_1 \leftarrow$ reductum($E_1$); // reductum($E_1$) $\rightarrow 0$ if $E_1$ is a non-sum.
   $L_2 \leftarrow$ leadTerm($E_2$);      $R_2 \leftarrow$ reductum($E_2$); // leadTerm($E_2$) $\rightarrow E_2$ if $E_2$ is a non-sum.
   if $L_1$ is similar to $L_2$, then              // The lead factors are identical
      return (leadFactor($L_1$)$\cdot$(remainingFactors($L_1$) + remainingFactors($L_2$)) + ($R_1 + R_2$);
   if leadFactor($L_2$) $\succ$ leadFactor($L_1$), then return $E_2 + E_1$;
   $S \leftarrow R_1 + E_2$;
   if $L_1$ is unomial-headed, then return $L_1$"+"$S$;
   // $L_1$ is a unit quasi-primitive sum-headed term:
   $x \leftarrow$ mainVariable($L_1$);
   if $L_1$ has a negative power of a sum containing $x$, then
   { $D_1 \leftarrow$ product of denominator factors in $L_1$ depending on $x$;
      return ratPlus($L_1, D_1, x, S, false$);
   }
   if $S$ is unit quasi primitive, then      // $L_1$ is already unit quasi primitive
   { if the syntactic content of $L_1$ and $S$ is 1, then
      { if $L_1$ is polynomially expandable, then return expand($L_1, x$) + $S$;
         return $L_1$"+"$S$;
      }
      $G \leftarrow$ syntacticNumericContent($L_1, S$);
      return $G\cdot(L_1/G + S/G)$;
   }
   if $L_1$ is polynomially expandable and it seems easier to do so than make $S$
      unit quasi primitive, then return expand($L_1, x$) + $S$;
   return $L_1$ + unitQuasiPrimitate($S$);
} // end function +
function ratPlus($L_1, D_1, x, E, DoMakeProper$):
{ // See invocation in function "+" for the roles of $L_1, D_1, x$ and , $E$.
  if $E$ is zero, then
  { if $DoMakeProper$ and $L_1$ is improper, then return makeProper($L_1$);
    return $L_1$;
  }
  $L_2 \leftarrow$ leadTerm($E$);
  if the denominator of $L_2$ is free of $x$, then
    return $L_2$"+"ratPlus($L_1, D_1, x$, reductum($E$), $true$);
  $D_2 \leftarrow$ product of denominator factors in $L_2$ depending on $x$;
  $G \leftarrow$ gcd($D_1, D_2$);          // Hall (1974), but recursive
  if $G$ is numeric, then
  { $S \leftarrow$ ratPlus($L_1, D_1, x$, reductum($E$), $true$);
    if $S$ is zero, then return $L_2$;
    if $L_2$ is improper, then return makeProper($L_2$) + $S$;
    return $L_2$"+"$S$;
  }
  if it seems easier to split denominator $G$ from $L_1$ and from $L_2$ than to combine them,
      and splitting doesn't introduce new singularities in the domain of interest, then
    return reductum($E$) + splitThenAdd($L_1, L_2, D_1, D_2, G, x$);
  return reductum($E$) + $G^{-2}$ "·" combineOverCommonDenominator($L_1/G, L_2/G$);
} // Use the Henrici-Brown algorithm (Knuth 1998) for the common denominator.
```

Acknowledgments

Thank you Albert Rich for being so patient when I struggled so long experimenting with alternative data structures and partially-factored semi fractions. Thanks also to Sam Rhoads for helping with LaTeX. Thanks in advance to anyone who makes suggestions that lead to improvement of this draft publication.

References

[Beeson 1998] Beeson, M: Design principles of MathPert: Software to support education in algebra and calculus, in *Computer-human Interaction in Symbolic Computation*, N. Kajler editor, Springer-Verlag, pp. 89-115.

[Brown 1974] Brown, W.S: On computing with factored rational expressions. *Proceedings of EUROSAM '74, ACM SIGSAM Bulletin* 8 (3), August, Issue Number 31, pp. 26-34.

[Hall 1974] Hall, A.D: Factored rational expressions in ALTRAN. *Proceedings of EUROSAM '74, ACM SIGSAM Bulletin* 8 (3), August, Issue Number 31, pp. 35-45.

[Hearn and Loos 1973] Hearn, A.C. and Loos, R.G.K: Extended polynomial algorithms. *Proceedings of ACM 73*, pp. 147-52.

[Jeffrey and Norman 2004] Jeffrey, D.J. and Norman, A.C: Not seeing the roots for the branches. *ACM SIGSAM Bulletin* 38 (3) pp. 57-66.

[Knuth 1998] Knuth, D.E: The Art of Computer Programming, Volume 2, 3rd edition, Addison-Wesley, Section 4.5.1.

[Moses 1971] Moses, J: Algebraic simplification a guide for the perplexed. *Proceedings of the second ACM symposium on symbolic and algebraic manipulation*, pp. 282-304.

[Moses 1981] Moses, J: Algebraic computation for the masses. (abstract) *Proceedings of the 1981 ACM symposium on symbolic and algebraic computation*, p. 168.

[Moses 2008] Moses, J: Macsyma: A personal history. *Proceedings of the Milestones in Computer Algebra conference.*

[Risch 1979] Risch, R.H., Algebraic properties of the elementary functions of analysis. *American Journal of Mathematics* 101, pp. 743-759.

[Stoutemyer 2007] Stoutemyer, D.R: Useful computations need useful numbers. *Communications in Computer Algebra* 41 (3-4), December.

[Stoutemyer 2008a] Stoutemyer, D.R: Monic normalization considered harmful. (submitted).

[Stoutemyer 2008b] Stoutemyer, D.R: Multivariate partial fraction expansion. (submitted).

[Stoutemyer 2008c] Stoutemyer, D.R: Some ways to implement computer algebra compactly. Applications of Computer Algebra 2008, forthcoming talk.

[Stoutemyer 2008d] Stoutemyer, D.R: Unit normalization of polynomials over Gaussian integers. (submitted).

[TI-Math-Engine 2008] http://education.ti.com/educationportal

[TI 89/TI-92 Plus Developers Guide 2001] http://education.ti.com/educationportal

[SMG 2003] TI Symbolic Math Guide (SMG): http://education.ti.com/educationportal

Tropical Algebraic Geometry in Maple

Jan Verschelde
University of Illinois at Chicago

Abstract

Finding a common factor of two multivariate polynomials with approximate coefficients is a problem in symbolic-numeric computing. Taking a tropical view on this problem leads to efficient preprocessing techniques, alternatingly applying polyhedral methods on the exact exponents with numerical techniques on the approximate coefficients. With Maple we will illustrate our use of tropical algebraic geometry. This is work in progress jointly with Danko Adrovic.

High-Precision Numerical Integration: Progress and Challenges

D.H. Bailey[*] J.M. Borwein[†]

October 22, 2008

Abstract. One of the most fruitful advances in the field of experimental mathematics has been the development of practical methods for very high-precision numerical integration, a quest initiated by Keith Geddes and other researchers in the 1980s and 1990s. These techniques, when coupled with equally powerful integer relation detection methods, have resulted in the analytic evaluation of many integrals that previously were beyond the realm of symbolic techniques. This paper presents a survey of the current state-of-the-art in this area (including results by the present authors and others), mentions some new results, and then sketches what challenges lie ahead.

1 Introduction

Numerical evaluation of definite integrals (often termed "quadrature") has numerous applications in applied mathematics, particularly in fields such as mathematical physics and computational chemistry. Beginning in the 1980s and 1990s, researchers such as Keith Geddes and others began to explore ways to extend some of the many known techniques to the realm of high precision — tens or hundreds of digits beyond the realm of standard machine precision. A recent paper by Keith Geddes and his coauthors [19] provides a good summary of how hybrid multi-dimensional integration can be done to moderate precision within *Maple*, a subject which has been of long-term interest to Keith Geddes [20]. In more recent years related methods, one of which we will describe below, have been developed to evaluate definite integrals to an *extreme precision* of thousands of digits accuracy.

These techniques have now become a staple in the emerging discipline of experimental mathematics, namely the application of high-performance computing to research questions in mathematics. In particular, high-precision numerical values of definite integrals, when combined with integer relation detection algorithms, can be used to discover previously unknown analytic evaluations (i.e., closed-form formulas) for these integrals, as well as previously unknown interrelations among classes of integrals. Large computations of this type can further be used to provide strong experimental validation of identities found in this manner or by other techniques.

We wish to emphasize that in this study we are principally *consumers* of symbolic computing tools rather than developers. However, we note that the techniques described here can be seen as an indirect means of symbolic computing — using *numerical* computation to produce *symbolic* results.

The underlying reason why very high precision results are needed in this context is that they are required if one wishes to apply *integer relation detection* methods. An integer relation detection scheme is a numerical algorithm which, given an n-long vector (x_i) of high-precision floating-point values, attempts to recover the

[*]Lawrence Berkeley National Laboratory, Berkeley, CA 94720, dhbailey@lbl.gov. Supported in part by the Director, Office of Computational and Technology Research, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy, under contract number DE-AC02-05CH11231.

[†]Faculty of Computer Science, Dalhousie University, Halifax, NS, B3H 2W5, Canada, jborwein@cs.dal.ca. Supported in part by NSERC and the Canada Research Chair Programme.

integer coefficients (a_i), not all zero, such that

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \quad = \quad 0$$

(to available precision), or else determine that there are no such integers less than a certain size. The algorithm *PSLQ* operates by developing, iteration by iteration, an integer-valued matrix A which successively reduces the maximum absolute value of the entries of the vector $y = Ax$, until one of the entries of y is zero (or within an "epsilon" of zero corresponding to the level of numeric precision being used). With PSLQ or any other integer relation detection scheme, if the underlying integer relation vector of length n has entries of maximum size d digits, then the input data must be specified to at least nd-digit precision and this level of precision must also be used in the operation of the integer relation algorithm, or else the true relation will be lost in a sea of spurious numerical artifacts.

1.1 Preliminary Applications

Example 1. A Parametric Integral. In one of the first applications of this methodology, the present authors and Greg Fee of Simon Fraser University in Canada were inspired by a then recent problem in the *American Mathematical Monthly* [2]. We found, by using a version of the integration scheme described below, coupled with a high-precision PSLQ program, that if $Q(a)$ is defined by

$$Q(a) \quad := \quad \int_0^1 \frac{\arctan(\sqrt{x^2+a^2})\,dx}{\sqrt{x^2+a^2}(x^2+1)},$$

then

$$\begin{aligned} Q(0) &= \pi \log 2/8 + G/2 \\ Q(1) &= \pi/4 - \pi\sqrt{2}/2 + 3\sqrt{2}\arctan(\sqrt{2})/2 \\ Q(\sqrt{2}) &= 5\pi^2/96. \end{aligned}$$

Here $G = \sum_{k\geq 0}(-1)^k/(2k+1)^2 = 0.91596559417\ldots$ is *Catalan's constant*. These specific experimental results then led to the following general result, which now has been rigorously established, among several others [15, pg. 307]:

$$\int_0^\infty \frac{\arctan(\sqrt{x^2+a^2})\,dx}{\sqrt{x^2+a^2}(x^2+1)} \quad = \quad \frac{\pi}{2\sqrt{a^2-1}}\left[2\arctan(\sqrt{a^2-1}) - \arctan(\sqrt{a^4-1})\right].$$

■

Example 2. A Character Sum. As a second example, to assist with a study of two-variable *character Euler-sums* [18], we empirically determined that

$$\begin{aligned} \frac{2}{\sqrt{3}}\int_0^1 \frac{\log^6(x)\arctan[x\sqrt{3}/(x-2)]}{x+1}\,dx \quad &= \quad \frac{1}{81648}\,[-229635 L_{-3}(8) \\ &+29852550 L_{-3}(7)\log 3 - 1632960 L_{-3}(6)\pi^2 + 27760320 L_{-3}(5)\zeta(3) \\ &-275184 L_{-3}(4)\pi^4 + 36288000 L_{-3}(3)\zeta(5) - 30008 L_{-3}(2)\pi^6 \\ &-57030120 L_{-3}(1)\zeta(7)]\,, \end{aligned}$$

where

$$L_{-3}(s) \quad := \quad \sum_{n\geq 1}\left[1/(3n-2)^s - 1/(3n-1)^s\right]$$

is based on the character modulo 3, and $\zeta(s) = \sum_{n\geq 1} 1/n^s$ is the Riemann zeta function. Based on these experimental results, general results of this type have been conjectured but few have yet been rigorously established—and may well never be unless their proof is somehow required. ■

2 High-Precision Numerical Quadrature

Several more challenging examples will be described below, but first we discuss our preferred underlying one-dimensional numerical techniques for high- and extreme-precision integration.

In our experience, we have come to rely on two schemes: *Gaussian* quadrature and *tanh-sinh* quadrature. Each has advantages in certain realms. We have found that if the function to be integrated is regular, both within the interval of integration as well as at the endpoints, and if the precision level is not too high (no more than a few hundred digits), then Gaussian quadrature is usually the most efficient scheme, in terms of achieving a desired accuracy in the lowest computer run time. In other cases, namely where the function is not regular at end points, or for precision levels above several hundred digits, tanh-sinh is usually the best choice.

2.1 Gaussian Quadrature

By way of a brief review, Gaussian quadrature approximates an integral on $[-1,1]$ as the sum $\sum_{0 \le j < n} w_j f(x_j)$, where the abscissas x_j are the roots of the n-th degree Legendre polynomial $P_n(x)$ on $[-1,1]$, and the weights w_j are

$$w_j \quad := \quad \frac{-2}{(n+1)P_n'(x_j)P_{n+1}(x_j)},$$

see [3, pg. 187]. Note that the abscissas and weights are independent of $f(x)$.

In our high-precision implementations, we compute an individual abscissa by using a Newton iteration root-finding algorithm with a dynamic precision scheme. The starting value for x_j in these Newton iterations is given by $\cos[\pi(j-1/4)/(n+1/2)]$, which may be calculated using ordinary 64-bit floating-point arithmetic [22, pg. 125]. We compute the Legendre polynomial function values using an n-long iteration of the recurrence $P_0(x) = 0,\ P_1(x) = 1$ and

$$(k+1)P_{k+1}(x) \quad = \quad (2k+1)xP_k(x) - kP_{k-1}(x)$$

for $k \ge 2$. The derivative is computed as $P_n'(x) = n(xP_n(x) - P_{n-1}(x))/(x^2-1)$. For functions defined on intervals other than $[-1,1]$, a linear scaling is used to convert the Gaussian abscissas to the actual interval.

In our implementations, we precompute several sets of abscissa-weight pairs corresponding to values of n that roughly double with each "level." Then in a quadrature calculation we repeat Gaussian quadrature with successively higher "levels" until either two consecutive results are equal within a specified accuracy, or an estimate of the error is below the specified accuracy. In most cases, once a modest precision has been achieved, increasing the level by one (i.e., doubling n) roughly doubles the number of correct digits in the result. Additional details of an efficient, robust high-precision implementation are given in [12].

One factor that limits the applicability of Gaussian quadrature for very high precision is that the cost of computing abscissa-weight pairs using this scheme increases quadratically with n, since each Legendre polynomial evaluation requires n steps. Since the value of n required to achieve a given precision level typically increases linearly with the precision level, this means that the total run-time cost of computing the abscissas and weights increases even faster than n^2 (in fact, it increases at least as $n^3 \log n$ or n^4, depending on how the high-precision arithmetic is implemented). There is no known scheme for generating Gaussian abscissa-weight pairs that avoids this quadratic dependence on n. High-precision abscissas and weights, once computed, may be stored for future use. But for truly extreme precision calculations — i.e., several thousand digits or more — the cost of computing them even once becomes prohibitive.

2.2 Tanh-Sinh Quadrature

The other quadrature algorithm we will mention here is the tanh-sinh scheme, which was originally discovered by Takahasi and Mori [23]. This scheme, as we will see, rapidly produces very high-precision results even if the integrand function has an infinite derivative or blow-up singularity at one or both endpoints. Its other major advantage is that the cost of generating abscissas and weights increases only linearly with the number of such pairs. For these reasons, the tanh-sinh scheme is the algorithm of choice for integrating any type of function, well-behaved or not, once the precision level rises beyond several hundred digits.

The tanh-sinh scheme is based on the observation, rooted in the *Euler-Maclaurin summation* formula, that for certain bell-shaped integrands (i.e., where the function and all higher derivatives rapidly approach zero at the endpoints of the interval), a simple block-function or trapezoidal approximation to the integral is remarkably accurate [3, pg. 180]. This principle is exploited in the tanh-sinh scheme by transforming an integral of a given function $f(x)$ on a finite interval such as $[-1, 1]$ to an integral on $(-\infty, \infty)$, by using the change of variable $x = g(t)$, where $g(t) = \tanh(\pi/2 \cdot \sinh t)$. The function $g(t)$ has the property that $g(x) \to 1$ as $x \to \infty$ and $g(x) \to -1$ as $x \to -\infty$, and also that $g'(x)$ and all higher derivatives rapidly approach zero for large positive and negative arguments. Thus one can write, for $h > 0$,

$$\int_{-1}^{1} f(x)\,dx \quad = \quad \int_{-\infty}^{\infty} f(g(t))g'(t)\,dt \;\approx\; h \sum_{j=-N}^{N} w_j f(x_j),$$

where $x_j = g(hj)$, $w_j = g'(hj)$ and N is chosen large enough that terms beyond N (positive or negative) are smaller than the "epsilon" of the numeric precision being used. In many cases, even where $f(x)$ has an infinite derivative or an integrable singularity at one or both endpoints, the transformed integrand $f(g(t))g'(t)$ is a smooth bell-shaped function for which the Euler-Maclaurin argument applies. In these cases, the error in this approximation decreases very rapidly with h, more rapidly than any fixed power of h.

In our implementations, we typically compute sets of abscissa-weight pairs for several values of h, starting at one and then decreasing by a factor of two with each "level." The abscissa-weight pairs (and corresponding function values) at one level are merely the even-indexed pairs (and corresponding function values) for the next higher level, so this fact can be used to save run time. We terminate the quadrature calculation when two consecutive levels have produced the same quadrature results to a specified accuracy, or when an estimate of the error is below the specified accuracy. Full details of an efficient, robust high-precision implementation are given in [12].

Both the tanh-sinh scheme and Gaussian quadrature often achieve "quadratic" or "exponential" convergence — in most cases, once a modest accuracy has been achieved, increasing the "level" by one (i.e., reducing h by half and roughly doubling N), approximately doubles the number of correct digits in the result. A proof of this fact, given certain regularity conditions, is discussed in [24].

2.3 Error Estimation

As suggested above, we often rely on estimates of the error in a quadrature calculation, terminating the process when the estimated error is less than a pre-specified accuracy. In several cases we have desired a more rigorous bound on the error, so that we can produce a "certificate" that the computed result is correct to within a given accuracy. In other cases, it is well worth avoiding the cost of the additional final step by use of an ad hoc estimate, such as the one described in [12].

There are several well-known rigorous error estimates for Gaussian quadrature [3, pg. 279]. Recently we established some similarly rigorous error estimates for tanh-sinh quadrature and other Euler-Maclaurin-based quadrature schemes, estimates that are both highly accurate and can be calculated in a similar manner to the quadrature result itself [4]. For example,

$$E_2(h, m) \quad := \quad h(-1)^{m-1}\left(\frac{h}{2\pi}\right)^{2m} \sum_{j=a/h}^{b/h} D^{2m}[g'(t)f(g(t))](jh) \tag{1}$$

yields extremely accurate estimates of the error, even in the simplest case $m = 1$. What's more, one can derive the bound

$$
\begin{aligned}
|E(h,m) - E_2(h,m)| \quad &\le \\
2\left[\zeta(2m) + (-1)^m \zeta(2m+2)\right] \left(\frac{h}{2\pi}\right)^{2m} & h\sqrt{\int_a^b \left|D^{2m}[g'(t)f(g(t))]\right|^2 \, dt}, \qquad (2)
\end{aligned}
$$

where $E(h,m)$ is the true error. These error bounds can be used to obtain rigorous certificates on the values of quadrature results.

Example 3. A Hyperbolic Volume. By using these methods we were able to establish rigorously that the experimentally observed identity

$$
\begin{aligned}
\frac{24}{7\sqrt{7}} \int_{\pi/3}^{\pi/2} \log \left| \frac{\tan t + \sqrt{7}}{\tan t - \sqrt{7}} \right| dt \quad &\overset{?}{=} \quad L_{-7}(2) \qquad (3) \\
&:= \sum_{n=0}^{\infty} \left[\frac{1}{(7n+1)^2} + \frac{1}{(7n+2)^2} \right. \\
&\left. - \frac{1}{(7n+3)^2} + \frac{1}{(7n+4)^2} - \frac{1}{(7n+5)^2} - \frac{1}{(7n+6)^2} \right]
\end{aligned}
$$

holds to within 3.82×10^{-49}. ■

Other examples of such certificates are given in [4]. Obtaining these certificate results is typically rather expensive, but by applying highly parallel implementation techniques these results can be obtained in reasonable run time.

2.4 Highly Parallel Quadrature

Both Gaussian quadrature and tanh-sinh quadrature are well-suited to highly parallel implementations, since each of the individual abscissa-weight calculations can be performed independently, as can each of the terms of the quadrature summation. One difficulty, however, is that if one is proceeding from level to level until a certain accuracy condition is met, then a serious load imbalance may occur if a cyclic distribution of abscissa-weight pairs is adopted. This can be solved by distributing these pairs in a more intelligent fashion [5].

Example 4. The Hyperbolic Volume. In the previous section, we introduced the conjectured identity (3). This arose out of quantum field theory, in analysis of the volumes of ideal tetrahedra in hyperbolic space. The question mark is used in (3) because no formal proof is known. Note that the integrand function has a nasty singularity at $t = \arctan(\sqrt{7})$ (see Figure 1).

Because of the interest expressed by researchers in the above and some related conjectures [16], we decided to calculate the integral

$$
\frac{24}{7\sqrt{7}} \int_{\pi/3}^{\pi/2} \log \left| \frac{\tan t + \sqrt{7}}{\tan t - \sqrt{7}} \right| dt \quad = \quad 1.15192547054449104710169239732054996\ldots
$$

to 20,000-digit accuracy (which approaches the limits of presently feasible computation) and compare with a 20,000-digit evaluation of the six-term infinite series on the right-hand side of (3). This integral was evaluated by splitting it into two integrals, the first from $\pi/3$ to $\arctan(\sqrt{7})$, and the second from $\arctan(\sqrt{7})$ to $\pi/2$, and then applying the 1-D tanh-sinh scheme to each part, performed in parallel on a highly parallel computer system. This test was successful—the numerical value of the integral on the left-hand side of (3) agrees with the numerical value of the six-term infinite series on the right-hand side to at least 19,995 digits. The infinite series was evaluated in approximately five hours on a personal computer using *Mathematica*.

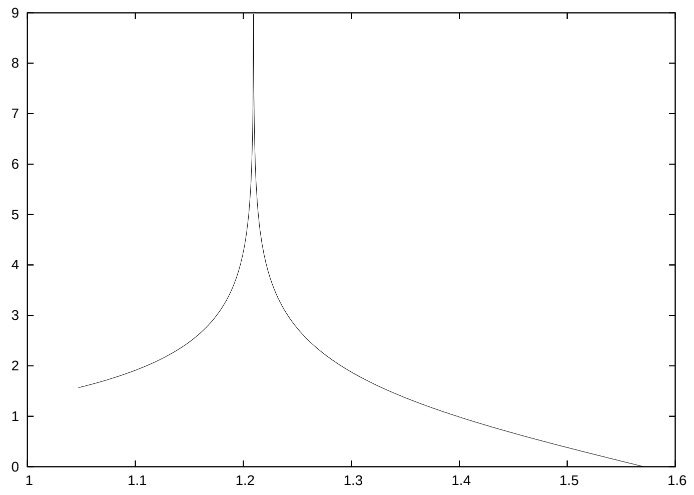


Figure 1: **Integrand function in (3) with singularity**

The computation of the integral was performed on the Apple-based parallel computer system at Virginia Tech. For this calculation, as well as for a number of others we describe below, we utilized the ARPREC arbitrary precision software package [11]. Parallel execution was controlled using the Message Passing Interface (MPI) software, a widely available system for parallel scientific computation [21]. The run required 45 minutes on 1024 CPUs of the Virginia Tech system, and ran at 690 Gflop/s (i.e., 690 billion 64-bit floating-point operations per second). It achieved an almost perfect parallel speedup of 993 (a perfect speedup would be 1024). Additional details of parallel quadrature techniques, for both one-dimensional and multi-dimensional integrals, are given in [5]. ■

Example 5. A Bessel Function integral. In a 2008 study, the present authors together with mathematical physicists David Broadhurst and Larry Glasser, examined various classes of integrals involving Bessel functions [6]. In this process we examined the constant $s_{6,1}$ defined by

$$s_{6,1} \quad = \quad \int_0^\infty t I_0(t) K_0^5(t)\, dt,$$

where $I_0(t)$ and $K_0(t)$ are the *modified Bessel functions* of the first and second kinds, respectively. This constant can be evaluated to high-precision by applying quadrature to this definition, or, even faster, by using a rapidly converging summation formula as described in [6]. Subsequently we discovered, by numerical experimentation, that

$$12 s_{6,1} \quad = \quad \pi^4 \int_0^\infty e^{-4t} I_0^4(t)\, dt. \tag{4}$$

Note that the integrand in (4) is singular at the origin (see Figure 2).

In order to test this conjecture, the present authors computed the right-hand side integral (4) to 14,285 digits, using a modified version of the parallel tanh-sinh program described above. This calculation required a run of 99 minutes on 1024 CPUs of the "Franklin" system (a Cray XT4 model, based on dual-core Opteron CPUs) in the National Energy Research Scientific Computing Center (NERSC) at the Lawrence Berkeley National Laboratory. These digits exactly matched the first 14,285 digits of $s_{6,1}$ computed by Broadhurst using the infinite series formula. Broadhurst now reports that he has a proof for (4). ■

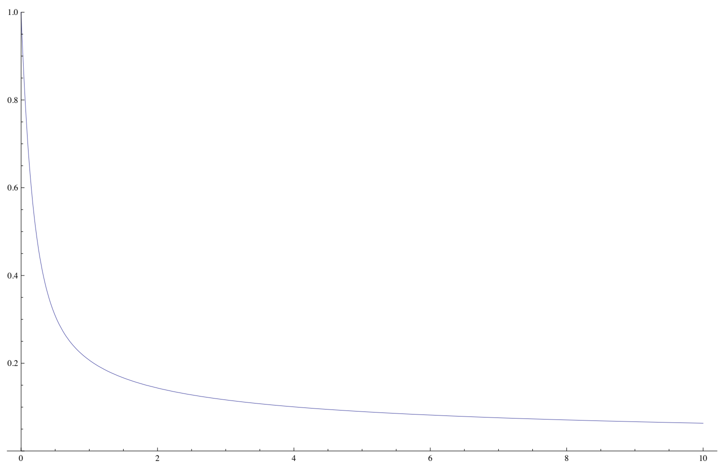


Figure 2: **Integrand function in (4) with singularity**

2.5 Multi-Dimensional Integration

The above examples are ordinary one-dimensional integrals, but two-dimensional, three-dimensional and higher-dimensional integrals are of at least as much interest. At present, the best available scheme for such computation is simply to perform either Gaussian quadrature or tanh-sinh quadrature in each dimension. Such computations are however vastly more expensive than in one dimension. For instance, if in one dimension a certain class of definite integral requires, say, 1000 function evaluations to achieve a certain desired accuracy, then in two dimensions one can expect to perform roughly 1,000,000 function evaluations for similar types of integrals, and 1,000,000,000 evaluations in three dimensions.

Fortunately, modern highly parallel computing technology makes it possible to consider the evaluation of some multiple integrals that heretofore would not have been feasible. Some of the techniques used in highly parallel, multi-dimensional integration are discussed in [5]. They typically require significant symbolic work prior to numerical computation.

3 Application to Ising Integrals

In a recent study, the present authors together with Richard Crandall examined the following classes of integrals, which arise in the Ising theory of mathematical physics [7]:

$$\begin{aligned}
C_n &:= \frac{4}{n!}\int_0^\infty \cdots \int_0^\infty \frac{1}{\left(\sum_{j=1}^n (u_j + 1/u_j)\right)^2} \frac{du_1}{u_1} \cdots \frac{du_n}{u_n} \\
D_n &:= \frac{4}{n!}\int_0^\infty \cdots \int_0^\infty \frac{\prod_{i<j}\left(\frac{u_i - u_j}{u_i + u_j}\right)^2}{\left(\sum_{j=1}^n (u_j + 1/u_j)\right)^2} \frac{du_1}{u_1} \cdots \frac{du_n}{u_n} \\
E_n &= 2\int_0^1 \cdots \int_0^1 \left(\prod_{1\le j<k\le n} \frac{u_k - u_j}{u_k + u_j}\right)^2 dt_2\, dt_3 \cdots dt_n,
\end{aligned}$$

where (in the last line) $u_k = \prod_{i=1}^k t_i$.[1]

[1] There are corresponding $(n-1)$-dimensional representations for C_n and D_n.

Needless to say, evaluating these n-dimensional integrals to high precision presents a daunting challenge.

3.1 The case of C_n

Example 6. The Limit of C_n. Fortunately, in the first case, we were able to show that the C_n integrals can be written as one-dimensional integrals:

$$C_n = \frac{2^n}{n!}\int_0^\infty pK_0^n(p)\,dp,$$

where K_0 is the *modified Bessel function* [1]. We were able to identify the first few instances of $C(n)$ in terms of well-known constants. For instance,

$$\begin{aligned} C_3 &= \mathrm{L}_{-3}(2) = \sum_{n\geq 0}\left(\frac{1}{(3n+1)^2}-\frac{1}{(3n+2)^2}\right) \\ C_4 &= 14\zeta(3). \end{aligned}$$

When we computed C_n for fairly large n, e.g.

$$C_{1024} = 0.6304735033743867961220401927108789043545870787127323 4\ldots$$

we found that these values rather quickly approached a limit. By using the new edition of the *Inverse Symbolic Calculator*, available at

`http://ddrive.cs.dal.ca/~isc`

this can be numerically identified as

$$\lim_{n\to\infty} C_n = 2e^{-2\gamma}.$$

We later we able to prove this fact—indeed this is merely the first term of an asymptotic expansion [7]. ■

3.2 The case of D_n and E_n

The more fundamental integrals D_n and E_n proved to be much more difficult to evaluate. They are *not* reducible to one-dimensional integrals (as far as we can tell), but with certain symmetry transformations and symbolic integration we were able to reduce the dimension in each case by one or two, so that we were able to produce the following evaluations, all of which save the last we subsequently were able to prove:

$$\begin{aligned} D_2 &= 1/3 \\ D_3 &= 8+4\pi^2/3-27\,\mathrm{L}_{-3}(2) \\ D_4 &= 4\pi^2/9-1/6-7\zeta(3)/2 \\ E_2 &= 6-8\log 2 \\ E_3 &= 10-2\pi^2-8\log 2+32\log^2 2 \\ E_4 &= 22-82\zeta(3)-24\log 2+176\log^2 2-256(\log^3 2)/3 \\ &\quad +16\pi^2\log 2-22\pi^2/3 \\ E_5 &\overset{?}{=} 42-1984\,\mathrm{Li}_4(1/2)+189\pi^4/10-74\zeta(3)-1272\zeta(3)\log 2 \\ &\quad +40\pi^2\log^2 2-62\pi^2/3+40(\pi^2\log 2)/3+88\log^4 2 \\ &\quad +464\log^2 2-40\log 2. \end{aligned}$$

In the case of D_n these were confirmations of known results.

Example 7. The Integral E_5. The result for E_5 required considerable effort, both computational and analytical. We did find a transformation that reduced this to a 3-D integral, but the resulting 3-D integral is extremely complicated (see Table 1). Just converting this expression (originally produced as a *Mathematica* expression) to a working computer program required considerable ingenuity. The numerical evaluation of this integral to 240 digits required four hours on 64 CPUs of the Virginia Tech Apple system. Applying PSLQ to the resulting numerical value (together with the numerical values of a set of conjectured terms), yielded the experimental evaluation shown above. ■

3.3 A Negative Result

Example 8. The Integrals C_5, and D_5. The computation of D_5 is even more demanding than E_5. Nonetheless, 18 hours on 256 CPUs of the Apple system at Virginia Tech produced 500 good digits. Alas, we still have not been successful in identifying either C_5 or D_5 from their numerical values (or by any other means). However, we have shown, via PSLQ computations, that neither C_5 nor D_5 satisfies an integer linear relation involving the following set of constants, where the vector of integer coefficients in the linear relation has Euclidean norm less than $4 \cdot 10^{12}$:

1, π, $\log 2$, π^2, $\pi \log 2$, $\log^2 2$, $\mathrm{L}_{-3}(2)$, π^3, $\pi^2 \log 2$, $\pi \log^2 2$, $\log^3 2$,
$\zeta(3)$, $\pi\, \mathrm{L}_{-3}(2)$, $\log 2 \cdot \mathrm{L}_{-3}(2)$, $\pi^4\, \pi^3 \log 2$, $\pi^2 \log^2 2$, $\pi \log^3 2$, G, $G\pi^2$,
$\mathrm{Li}_4(1/2)$, $\sqrt{3}\,\mathrm{L}_{-3}(2)$, $\log^4 2$, $\pi\zeta(3)$, $\log 2 \cdot \zeta(3)$, $\pi^2 \mathrm{L}_{-3}(2)$, $\pi^2 \mathrm{L}_{-3}(2)$,
$\pi \log 2 \cdot \mathrm{L}_{-3}(2)$, $\log^2 2 \cdot \mathrm{L}_{-3}(2)$, $\mathrm{L}^2_{-3}(2)$, $\mathrm{Im}[\mathrm{Li}_4(e^{2\pi i/5})]$, $\mathrm{Im}[\mathrm{Li}_4(e^{4\pi i/5})]$,
$\mathrm{Im}[\mathrm{Li}_4(i)]$, $\mathrm{Im}[\mathrm{Li}_4(e^{2\pi i/3})]$.

Here G and L_{-3} are as above and $\mathrm{Li}_n(x) := \sum_{k \geq 1} x^k/k^n$ is the *polylogarithm* function. Some constants that may appear to be "missing" from this list are actually linearly redundant with this set, and thus were not included in the PSLQ search. These include

$\mathrm{Re}[\mathrm{Li}_3(i)]$, $\mathrm{Im}[\mathrm{Li}_3(i)]$, $\mathrm{Re}[\mathrm{Li}_3(e^{2\pi i/3})]$, $\mathrm{Im}[\mathrm{Li}_3(e^{2\pi i/3})]$, $\mathrm{Re}[\mathrm{Li}_4(i)]$,
$\mathrm{Re}[\mathrm{Li}_4(e^{2\pi i/3})]$, $\mathrm{Re}[\mathrm{Li}_4(e^{2\pi i/5})]$, $\mathrm{Re}[\mathrm{Li}_4(e^{4\pi i/5})]$, $\mathrm{Re}[\mathrm{Li}_4(e^{2\pi i/6})]$ and
$\mathrm{Im}[\mathrm{Li}_4(e^{2\pi i/6})]$.

Despite this failure, we view these computations as successful, since the numerical values of D_5 and C_5 (along with numerous other related constants) are available at [8] to any researcher who may have a better idea of where to hunt for a closed form. ■

3.4 Closed Forms for $c_{n,k}$

In a follow-on study [9], the present authors and Richard Crandall examined the following generalization of the C_n integrals above:

$$C_{n,k} \quad = \quad \frac{4}{n!} \int_0^\infty \cdots \int_0^\infty \frac{1}{\left(\sum_{j=1}^n (u_j + 1/u_j)\right)^{k+1}} \frac{du_1}{u_1} \cdots \frac{du_n}{u_n}$$

Here we made the initially surprising discovery—now proven in [17]—that there are linear relations in each of the rows of this array (considered as a doubly-infinite rectangular matrix). For instance,

$$\begin{aligned}
0 &= C_{3,0} - 84C_{3,2} + 216C_{3,4} \\
0 &= 2C_{3,1} - 69C_{3,3} + 135C_{3,5} \\
0 &= C_{3,2} - 24C_{3,4} + 40C_{3,6} \\
0 &= 32C_{3,3} - 630C_{3,5} + 945C_{3,7} \\
0 &= 125C_{3,4} - 2172C_{3,6} + 3024C_{3,8}.
\end{aligned}$$

In yet a more recent study [6], we were able to analytically recognize many of these $C_{n,k}$ integrals—because, remarkably, these same integrals appear naturally in quantum field theory (for odd k)!

$$\begin{aligned}
E_5 \;=\; & \int_0^1\int_0^1\int_0^1 \bigl[2(1-x)^2(1-y)^2(1-xy)^2(1-z)^2(1-yz)^2(1-xyz)^2 \\
& \bigl(-\bigl[4(x+1)(xy+1)\log(2)\bigl(y^5z^3x^7-y^4z^2(4(y+1)z+3)x^6-y^3z\bigl(\bigl(y^2+1\bigr)z^2+4(y+ \\
& 1)z+5\bigr)x^5+y^2\bigl(4y(y+1)z^3+3\bigl(y^2+1\bigr)z^2+4(y+1)z-1\bigr)x^4+y\bigl(z\bigl(z^2+4z \\
& +5\bigr)y^2+4\bigl(z^2+1\bigr)y+5z+4\bigr)x^3+\bigl(\bigl(-3z^2-4z+1\bigr)y^2-4zy+1\bigr)x^2-(y(5z+4) \\
& +4)x-1)\bigr]\,/\,\bigl[(x-1)^3(xy-1)^3(xyz-1)^3\bigr]+\bigl[3(y-1)^2y^4(z-1)^2z^2(yz \\
& -1)^2x^6+2y^3z\bigl(3(z-1)^2z^3y^5+z^2\bigl(5z^3+3z^2+3z+5\bigr)y^4+(z-1)^2z \\
& \bigl(5z^2+16z+5\bigr)y^3+\bigl(3z^5+3z^4-22z^3-22z^2+3z+3\bigr)y^2+3\bigl(-2z^4+z^3+2 \\
& z^2+z-2\bigr)y+3z^3+5z^2+5z+3\bigr)x^5+y^2\bigl(7(z-1)^2z^4y^6-2z^3\bigl(z^3+15z^2 \\
& +15z+1\bigr)y^5+2z^2\bigl(-21z^4+6z^3+14z^2+6z-21\bigr)y^4-2z\bigl(z^5-6z^4-27z^3 \\
& -27z^2-6z+1\bigr)y^3+\bigl(7z^6-30z^5+28z^4+54z^3+28z^2-30z+7\bigr)y^2-2\bigl(7z^5 \\
& +15z^4-6z^3-6z^2+15z+7\bigr)y+7z^4-2z^3-42z^2-2z+7\bigr)x^4-2y\bigl(z^3\bigl(z^3 \\
& -9z^2-9z+1\bigr)y^6+z^2\bigl(7z^4-14z^3-18z^2-14z+7\bigr)y^5+z\bigl(7z^5+14z^4+3 \\
& z^3+3z^2+14z+7\bigr)y^4+\bigl(z^6-14z^5+3z^4+84z^3+3z^2-14z+1\bigr)y^3-3\bigl(3z^5 \\
& +6z^4-z^3-z^2+6z+3\bigr)y^2-\bigl(9z^4+14z^3-14z^2+14z+9\bigr)y+z^3+7z^2+7z \\
& +1\bigr)x^3+\bigl(z^2\bigl(11z^4+6z^3-66z^2+6z+11\bigr)y^6+2z\bigl(5z^5+13z^4-2z^3-2z^2 \\
& +13z+5\bigr)y^5+\bigl(11z^6+26z^5+44z^4-66z^3+44z^2+26z+11\bigr)y^4+\bigl(6z^5-4 \\
& z^4-66z^3-66z^2-4z+6\bigr)y^3-2\bigl(33z^4+2z^3-22z^2+2z+33\bigr)y^2+\bigl(6z^3+26 \\
& z^2+26z+6\bigr)y+11z^2+10z+11\bigr)x^2-2\bigl(z^2\bigl(5z^3+3z^2+3z+5\bigr)y^5+z\bigl(22z^4 \\
& +5z^3-22z^2+5z+22\bigr)y^4+\bigl(5z^5+5z^4-26z^3-26z^2+5z+5\bigr)y^3+\bigl(3z^4- \\
& 22z^3-26z^2-22z+3\bigr)y^2+\bigl(3z^3+5z^2+5z+3\bigr)y+5z^2+22z+5\bigr)x+15z^2+2z \\
& +2y(z-1)^2(z+1)+2y^3(z-1)^2z(z+1)+y^4z^2\bigl(15z^2+2z+15\bigr)+y^2\bigl(15z^4 \\
& -2z^3-90z^2-2z+15\bigr)+15\bigr]\,/\,\bigl[(x-1)^2(y-1)^2(xy-1)^2(z-1)^2(yz-1)^2 \\
& (xyz-1)^2\bigr]-\bigl[4(x+1)(y+1)(yz+1)\bigl(-z^2y^4+4z(z+1)y^3+\bigl(z^2+1\bigr)y^2 \\
& -4(z+1)y+4x\bigl(y^2-1\bigr)\bigl(y^2z^2-1\bigr)+x^2\bigl(z^2y^4-4z(z+1)y^3-\bigl(z^2+1\bigr)y^2 \\
& +4(z+1)y+1\bigr)-1\bigr)\log(x+1)\bigr]\,/\,\bigl[(x-1)^3x(y-1)^3(yz-1)^3\bigr]-\bigl[4(y+1)(xy \\
& +1)(z+1)\bigl(x^2\bigl(z^2-4z-1\bigr)y^4+4x(x+1)\bigl(z^2-1\bigr)y^3-\bigl(x^2+1\bigr)\bigl(z^2-4z-1\bigr) \\
& y^2-4(x+1)\bigl(z^2-1\bigr)y+z^2-4z-1\bigr)\log(xy+1)\bigr]\,/\,\bigl[x(y-1)^3y(xy-1)^3(z- \\
& 1)^3\bigr]-\bigl[4(z+1)(yz+1)\bigl(x^3y^5z^7+x^2y^4(4x(y+1)+5)z^6-xy^3\bigl(\bigl(y^2+ \\
& 1\bigr)x^2-4(y+1)x-3\bigr)z^5-y^2\bigl(4y(y+1)x^3+5\bigl(y^2+1\bigr)x^2+4(y+1)x+1\bigr)z^4+ \\
& y\bigl(y^2x^3-4y(y+1)x^2-3\bigl(y^2+1\bigr)x-4(y+1)\bigr)z^3+\bigl(5x^2y^2+y^2+4x(y+1) \\
& y+1\bigr)z^2+((3x+4)y+4)z-1\bigr)\log(xyz+1)\bigr]\,/\,\bigl[xy(z-1)^3z(yz-1)^3(xyz-1)^3\bigr]\bigr)\bigr] \\
& /\,\bigl[(x+1)^2(y+1)^2(xy+1)^2(z+1)^2(yz+1)^2(xyz+1)^2\bigr]\;dx\,dy\,dz
\end{aligned}$$

Table 1: **The E_5 integral**

Example 10. Four Hypergeometric Forms. In particular, we discovered, and then proved with considerable effort that, with $c_{n,k}$ normalized by $C_{n,k} = 2^n\, c_{n,k}/(n!\,k!)$, we have

$$
\begin{aligned}
c_{3,0} &= \frac{3\Gamma^6(1/3)}{32\pi 2^{2/3}} = \frac{\sqrt{3}\pi^3}{8}\,{}_3F_2\left(\left.\begin{array}{c}1/2,1/2,1/2\\1,1\end{array}\right|\frac{1}{4}\right)\\
c_{3,2} &= \frac{\sqrt{3}\pi^3}{288}\,{}_3F_2\left(\left.\begin{array}{c}1/2,1/2,1/2\\2,2\end{array}\right|\frac{1}{4}\right)\\
c_{4,0} &= \frac{\pi^4}{4}\sum_{n=0}^{\infty}\frac{\binom{2n}{n}^4}{4^{4n}} = \frac{\pi^4}{4}\,{}_4F_3\left(\left.\begin{array}{c}1/2,1/2,1/2,1/2\\1,1,1\end{array}\right|1\right)\\
c_{4,2} &= \frac{\pi^4}{64}\left[4\,{}_4F_3\left(\left.\begin{array}{c}1/2,1/2,1/2,1/2\\1,1,1\end{array}\right|1\right)\right.\\
&\qquad \left.-3\,{}_4F_3\left(\left.\begin{array}{c}1/2,1/2,1/2,1/2\\2,1,1\end{array}\right|1\right)\right] - \frac{3\pi^2}{16},
\end{aligned}
$$

where ${}_pF_q$ denotes the *generalized hypergeometric* function [1]. ■

4 Application to Elliptic Integral Evaluations

The work in [6] also required computation of some very tricky two and three-dimensional integrals arising from *sunrise diagrams* in quantum field theory.

Example 10. Two Elliptic Integral Integrals. For example, with $\mathbf{K}$ denoting the *complete elliptic integral* of the first kind, we had conjectured

$$
c_{5,0} = \frac{\pi}{2}\int_{-\pi/2}^{\pi/2}\int_{-\pi/2}^{\pi/2}\frac{\mathbf{K}(\sin\theta)\,\mathbf{K}(\sin\phi)}{\sqrt{\cos^2\theta\cos^2\phi + 4\sin^2(\theta+\phi)}}\,\mathrm{d}\theta\,\mathrm{d}\phi\,. \tag{5}
$$

Note that this function has singularities on all four sides of the domain of integration (see Figure 3).

Ultimately, we were able to compute $c_{5,0}$ to 120-digit accuracy, using 240-digit working precision. This run required a parallel computation (using the MPI parallel programming library) of 43.2 minutes on 512 CPUs (1024 cores) of the "Franklin" system at LBNL. Likewise we could confirm

$$
c_{6,0} = \frac{\pi}{2}\int_{-\pi/2}^{\pi/2}\int_{-\pi/2}^{\pi/2}\frac{\mathbf{K}(\sin\theta)\,\mathbf{K}(\sin\phi)\,\mathbf{K}\left(\frac{\sin(\theta+\phi)}{\sqrt{\cos^2\theta\cos^2\phi+\sin^2(\theta+\phi)}}\right)}{\sqrt{\cos^2\theta\cos^2\phi + \sin^2(\theta+\phi)}}\,\mathrm{d}\theta\,\mathrm{d}\phi\,.
$$

■

Example 11. A Bessel Moment Conjecture. This same work also uncovered the following striking *conjecture*: for each integer pair (n,k) with $n \ge 2k \ge 2$ we have

$$
\sum_{m=0}^{\lfloor n/2\rfloor}(-1)^m\binom{n}{2m}\int_0^\infty t^{n-2k}[\pi I_0(t)]^{n-2m}[K_0(t)]^{n+2m}\,\mathrm{d}t \overset{?}{=} 0
$$

where I_0 and K_0 are again Bessel functions. ■

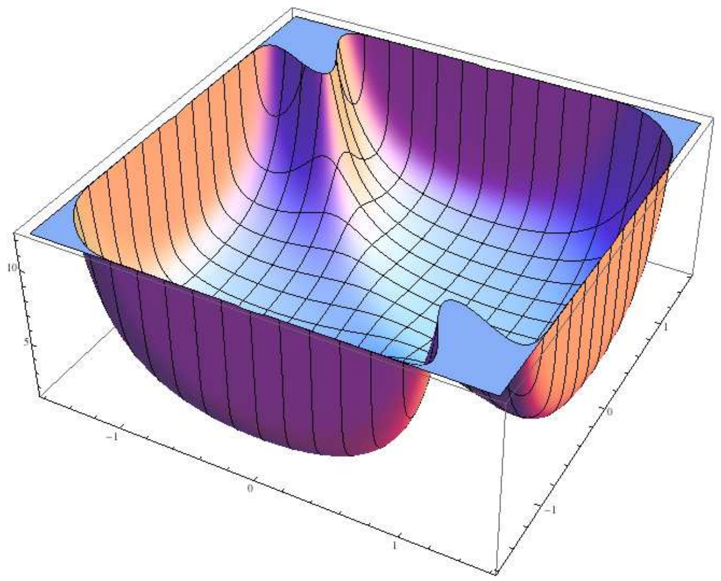


Figure 3: **Plot of $c_{5,0}$ integrand function in (5)**

5 Application to Heisenberg Spin Integrals

In another recent application of these methods, we investigated the following integrals ("spin integrals"), which arise, like the Ising integrals, from studies in mathematical physics [13, 14]:

$$\begin{aligned} P(n) \quad := \quad & \frac{\pi^{n(n+1)/2}}{(2\pi i)^n} \cdot \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} U(x_1 - i/2, x_2 - i/2, \cdots, x_n - i/2) \\ & \times T(x_1 - i/2, x_2 - i/2, \cdots, x_n - i/2)\, dx_1 dx_2 \cdots dx_n \end{aligned}$$

where

$$\begin{aligned} U(x_1 - i/2, x_2 - i/2, \cdots, x_n - i/2) \quad &= \quad \frac{\prod_{1 \le k < j \le n} \sinh[\pi(x_j - x_k)]}{\prod_{1 \le j \le n} i^n \cosh^n(\pi x_j)} \\ T(x_1 - i/2, x_2 - i/2, \cdots, x_n - i/2) \quad &= \quad \frac{\prod_{1 \le j \le n} (x_j - i/2)^{j-1} (x_j + i/2)^{n-j}}{\prod_{1 \le k < j \le n} (x_j - x_k - i)}. \end{aligned}$$

Note that these integrals involve some complex-arithmetic calculations, even though the final results are real.

Example 12. Spin Values. So far we have been able to numerically confirm the following results:

$$\begin{aligned} P(1) \quad &= \quad \frac{1}{2}, \quad P(2) = \frac{1}{3} - \frac{1}{3} \log 2, \quad P(3) = \frac{1}{4} - \log 2 + \frac{3}{8} \zeta(3) \\ P(4) \quad &= \quad \frac{1}{5} - 2 \log 2 + \frac{173}{60} \zeta(3) - \frac{11}{6} \zeta(3) \log 2 - \frac{51}{80} \zeta^2(3) - \frac{55}{24} \zeta(5) + \frac{85}{24} \zeta(5) \log 2 \\ P(5) \quad &= \quad \frac{1}{6} - \frac{10}{3} \log 2 + \frac{281}{24} \zeta(3) - \frac{45}{2} \zeta(3) \log 2 - \frac{489}{16} \zeta^2(3) - \frac{6775}{192} \zeta(5) \\ & \quad + \frac{1225}{6} \zeta(5) \log 2 - \frac{425}{64} \zeta(3) \zeta(5) - \frac{12125}{256} \zeta^2(5) + \frac{6223}{256} \zeta(7) \\ & \quad - \frac{11515}{64} \zeta(7) \log 2 + \frac{42777}{512} \zeta(3) \zeta(7), \end{aligned}$$

n	Digits	Processors	Run Time
2	120	1	10 sec.
3	120	8	55 min.
4	60	64	27 min.
5	30	256	39 min.
6	6	256	59 hrs.

Table 2: **Computation times for** $P(n)$

as well as a significantly more complicated expression for $P(6)$. We confirmed $P(1)$ through $P(4)$ to over 60-digit precision; $P(5)$ to 30-digit precision, but $P(6)$ to only 8-digit precision. These quadrature calculations were performed as parallel jobs on the Apple G5 cluster at Virginia Tech. Were we able to compute $P(n)$ for $n < 9$ to say 100 places, we might well be able to use PSLQ to determine the precise closed forms of $P(n)$. How difficult a task this is is illustrated by the run times and processors used shown in Table 2, which underscores the rapidly escalating difficulty of these computations. ■

These huge and rapidly increasing run times, as in the Ising integral study, point to the critical need for research into fundamentally new and more efficient numerical schemes for two-dimensional and higher-dimensional integrals. It is hoped that the results in this paper will stimulate research in that direction.

References

[1] Milton Abramowitz and Irene A. Stegun, *Handbook of Mathematical Functions,* Dover, NY, 1970.

[2] Zafar Ahmed, "Definitely an Integral," *American Mathematical Monthly*, vol. 109 (2002), no. 7, pg. 670–671.

[3] K. E. Atkinson, *Elementary Numerical Analysis*, John Wiley & Sons, 1993.

[4] David H. Bailey and Jonathan M. Borwein, "Effective Error Bounds in Euler-Maclaurin-Based Quadrature Schemes," *Proc. 2006 Conf. on High-Performance Computing Systems*, IEEE Computer Society, 2006, available at `http://crd.lbl.gov/~dhbailey/dhbpapers/hpcs06.pdf`.

[5] D. H. Bailey and J. M. Borwein, "Highly Parallel, High-Precision Numerical Integration," *International Journal of Computational Science and Engineering*, to appear, 2008, available at `http://crd.lbl.gov/~dhbailey/dhbpapers/quadparallel.pdf`.

[6] D. H. Bailey, J. M. Borwein, D. Broadhurst and M. L. Glasser, "Elliptic integral evaluations of Bessel moments," Jan 2008, *Journal of Physics A: Mathematical and General*, to appear, 2008, available at `http://crd.lbl.gov/~dhbailey/dhbpapers/b3g.pdf`.

[7] D. H. Bailey, J. M. Borwein and R. E. Crandall, "Integrals of the Ising Class," *Journal of Physics A*, vol. 39 (2006), pg. 12271–12302.

[8] D. H. Bailey, J. M. Borwein and R. E. Crandall, "Ising Data," 2006, available at `http://crd.lbl.gov/~dhbailey/dhbpapers/ising-data.pdf`.

[9] David H. Bailey, David Borwein, Jonathan M. Borwein and Richard Crandall, "Hypergeometric Forms for Ising-Class Integrals," Experimental Math., vol. 16 (2007), no. 3, pg. 257–276.

[10] David H. Bailey and David Broadhurst, "Parallel Integer Relation Detection: Techniques and Applications," *Math. of Computation*, vol. 70, no. 236 (2000), pg. 1719-1736.

[11] David H. Bailey, Yozo Hida, Xiaoye S. Li and Brandon Thompson, "ARPREC: An Arbitrary Precision Computation Package," technical report LBNL-53651, software and documentation available at `http://crdl.bl.gov/~dhbailey/mpdist`.

[12] David H. Bailey, Xiaoye S. Li and Karthik Jeyabalan, "A Comparison of Three High-Precision Quadrature Programs," manuscript, available at `http://crd.lbl.gov/~dhbailey/dhbpapers/quadrature.pdf`.

[13] H. E. Boos and V. E. Korepin, "Quantum Spin Chains and Riemann Zeta Function with Odd Arguments," *Journal of Physics A*, vol. 34 (2001), pg. 5311–5316, preprint available at `http://arxiv.org/abs/hep-th/0104008`.

[14] H. E. Boos, V. E. Korepin, Y. Nishiyama and M. Shiroishi, "Quantum Correlations and Number Theory," *Journal of Physics A*, vol. 35 (2002), pg. 4443, available at `http://arxiv.org/abs/cond-mat/0202346`.

[15] Jonathan M. Borwein, David H. Bailey and Roland Girgensohn, *Experimentation in Mathematics: Computational Paths to Discovery*, A K Peters, Welesly, MA, 2004.

[16] J. Borwein and D. Broadhurst, "Determination of Rational Dirichlet-zets Invariants of Hyperbolic Manifolds and Feynman Knots and Links," available at `http://arxiv.org/hep-th/9811173`.

[17] Jonathan Borwein and Bruno Salvy, "A Proof of a Recursion for Bessel Moments," *Experimental Mathematics*, to appear, D-drive Preprint 346, 2007, `http://locutus.cs.dal.ca:8088/archive/00000346/`.

[18] J. M. Borwein, I. J. Zucker and J. Boersma, "The Evaluation of Character Euler Double Sums," *Ramanujan Journal*, vol. 15 (2008), to appear.

[19] O. A. Carvajal, Orlando, F. W. Chapman, and K. O. Geddes, "Hybrid Symbolic-Numeric Integration in Multiple Dimensions via Tensor-Product Series," *ISSAC'05*, pg. 84–91 (electronic), ACM Press, New York, 2005.

[20] K. O. Geddes and G. J. Fee, "Hybrid Symbolic-Numeric Integration in Maple," *Proceedings of ISAAC'92*, pg. 36–41, ACM Press, New York, 1992.

[21] William Gropp, Ewing Lusk, Anthony Skjellum, *Using MPI: A Portable Parallel Programming with the Message-Passing Interface*, MIT Press, Cambridge, MA, 1996.

[22] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, Cambridge Univ. Press, 1986.

[23] H. Takahasi and M. Mori, "Double Exponential Formulas for Numerical Integration," Publ. of RIMS, Kyoto Univ., vol. 9 (1974), pg. 721–741.

[24] Lingyun Ye, *Numerical Quadrature: Theory and Computation*, MSc Thesis, Computer Science, Dalhousie University, available at `http://locutus.cs.dal.ca:8088/archive/00000328`.

Adaptive Polynomial Multiplication

Daniel S. Roche
Symbolic Computation Group
University of Waterloo
www.cs.uwaterloo.ca/~droche

Abstract

Finding the product of two polynomials is an essential and basic problem in computer algebra. While most previous results have focused on the worst-case complexity, we instead employ the technique of adaptive analysis to give an improvement in many "easy" cases where other algorithms are doing too much work. Three ideas for adaptive polynomial multiplication are given. One method, which we call "chunky" multiplication, is given a more careful analysis, as well as an implementation in NTL. We show that significant improvements can be had over the fastest general-purpose algorithms in many cases.

1 Introduction

Polynomial multiplication has been one of the most well-studied topics in computer algebra and symbolic computation over the last half-century, and has proven to be one of the most crucial primitive operations in a computer algebra system. However, most results focus on the worst-case analysis, and in doing so overlook many cases where polynomials can be multiplied much more quickly. We develop algorithms which are significantly faster than current methods in many instances, and which are still never (asymptotically) slower.

For univariate polynomials, multiplication algorithms generally fall into one of two classes, depending on which representation is used. Let R be a ring, and $f \in \mathsf{R}[x]$ of degree n and with t nonzero terms. The *dense* representation of f is by a vector of all $n+1$ coefficients. The *sparse* representation is a list of pairs of nonzero coefficient and exponent, with size bounded by $O(t \log n)$ from above and $\Omega(t \log t + \log n)$ from below.

Advances in dense polynomial multiplication have usually followed advances in long integer multiplication, starting with the first sub-quadratic algorithm by Karatsuba and Ofman in 1962 [7]. Schönhage and Strassen were the first to use the FFT to achieve $O(n \log n \mathrm{loglog} n)$ complexity for integer multiplication; this was extended to polynomials over arbitrary rings by Cantor and Kaltofen [3, 11, 2].

If we denote by $\mathsf{M}(n)$ the number of ring operations in R needed to multiply two polynomials with degrees less than n over $\mathsf{R}[x]$, then we have $\mathsf{M}(n) \in O(n \log n \mathrm{loglog} n)$. A lower bound of $\Omega(n \log n)$ as has also been proven under the "bounded coefficients" model [1]. Progress towards eliminating the $\log \log n$ factor is an ongoing area of research (see e.g. [4]).

To multiply two sparse polynomials with t nonzero terms, the naïve algorithm requires $O(t^2)$ ring operations. In fact, this is optimal, since the product could have that many terms. But for sparse polynomials, we must also account for other word operations that arise from the exponent arithmetic. Using "geobuckets", the bit complexity for exponent arithmetic becomes $O(t^2 \log t \log n)$ [14]; more recent results show how to reduce the space complexity to achieve an even more efficient algorithm [9].

Sparse representations become very useful when polynomials are in many variables, as the dense size grows exponentially in the number of indeterminates. In this case, others have noticed that the best overall approach may be to use a combination of sparse and dense methods in what is called the *recursive dense* representation [13]. Since most multivariate algorithms boil down to univariate algorithms, we restrict ourselves here to polynomials over $\mathsf{R}[x]$. Our algorithms will easily extend to multivariate polynomials, but the details of such adaptations are not presented here.

Section 2 gives a general overview of adaptive analysis, and how we will make use of this analysis for polynomial multiplication. Next we present one idea for adaptive multiplication, where the input polynomials are split up into dense "chunks". In Section 4, we cover an implementation of this technique in the C++ library NTL. Two other ideas for adaptive multiplication are put forth in Section 5. Finally, we discuss the practical usefulness of our algorithms and future directions for research.

2 Adaptive Analysis

By "adaptive", we mean algorithms whose complexity depends not only on the size of the input, but also on some other measure of difficulty. This terminology comes from the world of sorting algorithms, and its first use is usually credited to Mehlhorn [8]. Adaptive algorithms for sorting will have complexity dependent not only on the length of the list to be sorted, but also to what extent the list is already sorted. The results hold both theoretical interest and practical importance (for a good overview of adaptive sorting, see [10]). Some have observed at least a historical connection between polynomial multiplication and sorting [5], so it seems appropriate that our motivation comes from this area.

These algorithms identify "easy" cases, and solve them more quickly than the general, "difficult", cases. Really, we are giving a finer partition of the problem space, according to some measure of difficulty in addition to the usual size of the input. We require that our algorithms never behave worse than the usual ones, so that a normal worst-case analysis would give the same results. However, we also guarantee that easier cases be handled more quickly (with "easiness" being defined according to our chosen difficulty measure).

Adaptive analysis is not new to computer algebra, but usually takes other names, for instance "early termination" strategies for polynomial and linear algebra computations (see e.g. [6]).

But how can we identify "easy cases" for multiplication? An obvious difficulty measure is the sparsity of the input polynomials. This leads immediately to a trivial adaptive algorithm: (1) find the number of nonzero terms and determine whether sparse or dense algorithms will be best, and then (2) convert to that representation and perform the multiplication. In fact, such an approach has been suggested already to handle varying sparsity in the intermediate computations of triangular decompositions.

2.1 Our Approach

The algorithms we present will always proceed in three stages. First, the polynomials are read in and converted to a different representation which effectively captures the relevant measure of difficulty. Second, we multiply the two polynomials in the alternate representation. Finally, the product is converted back to the original representation.

The aim here is that the conversions to and from our "adaptive" representation are as fast as possible, i.e. linear time in the size of the input and output. Then the second step, whose cost should depend on the difficulty of the instance, can determine the actual complexity.

For the methods we put forth, the second step is relatively straightforward given the chosen representation. The final step will be even simpler, and it will usually be possible to combine it with step (2) for greater efficiency. The most challenging aspect will be designing linear-time algorithms for the first step, as we are somehow trying to recognize structure from chaos.

3 Chunky Multiplication

The idea here is simple, and provides a natural gradient between the well-studied dense and sparse algorithms for univariate polynomial arithmetic. For $f \in \mathsf{R}[x]$ of degree n, we represent f as a sparse polynomial with dense "chunks" as coefficients:

$$f = f_1 x^{e_1} + f_2 x^{e_2} + \cdots + f_t x^{e_t}, \tag{1}$$

with each $f_i \in \mathsf{R}[x]$ and $e_i \in \mathbb{N}$. Let $d_1, d_2, \ldots, d_t \in \mathbb{N}$ be such that the degree of each f_i is less than d_i. Then we require $e_{i+1} > e_i + d_i$ for $i = 1, 2, \ldots, t-1$, so that there is some "gap" between each dense "chunk". We do *not* insist that each f_i be completely dense, but require only that the leading and constant coefficients be nonzero. In fact, deciding how much space to allow in each chunk is the challenge of converting to this representation, as we shall see.

Multiplying polynomials in the chunky representation uses sparse multiplication on the outer loop, treating the f_i's as coefficients, and dense multiplication to find each product $f_i g_j$. By using heaps of pointers as in [9], the chunks of the result are computed in order, eliminating unnecessary additions and making the conversion back to the original representation (dense *or* sparse) linear time, as required.

3.1 Analysis

We now give a rough analysis of the multiplication step, which we use to guide the decisions in converting to the chunky representation. For this, we assume that $\mathsf{M}(n) = cn\log_2(n+1)$ for some positive constant c. This estimate will not be too far off if we use FFT-based multiplication, and in fact will be accurate when R contains a $2^k th$ primitive root of unity for $2^k \geq 2n$. We will also use the fact that two polynomials of degrees less than m, n can be multiplied with $O(\frac{n}{m}\mathsf{M}(m))$ ring operations when $n > m$, which under our assumption is $O(n\log m)$.

Now note that, when multiplying two polynomials in the chunky representation, the cost (in ring operations) is the same as if we multiplied the first polynomial by each chunk in the second polynomial, and then merged the results (although this is of course *not* what we will actually do). So for our rough analysis, we will (without loss of generality) seek to minimize the cost of multiplying a given polynomial f by an arbitrary polynomial g, which we assume is totally dense.

Theorem 3.1. *Let $f, g \in \mathsf{R}[x]$ with f as in (1) and g dense of degree m. Then the number of ring operations needed to compute the product fg is*

$$O\left(m\log\prod_{d_i\leq m}(d_i+1) + (\log m)\sum_{d_i>m} d_i\right).$$

The proof follows from the discussion above on the cost of multiplying polynomials with different degrees. Next we give two lemmas that indicate what we must minimize in order to be competitive with known techniques for dense and sparse multiplication.

Lemma 3.2. *If $\prod(d_i+1) \in O(n)$, then the cost of chunky multiplication is never asymptotically greater than the cost of dense multiplication.*

Proof. First, notice that $\sum d_i \leq n$ (otherwise we would have overlap in the chunks). And assume $\prod(d_i+1) \in O(n)$. From Theorem 3.1, the cost of chunky multiplication is thus $O(m\log n + n\log m)$. But this is exactly the cost of dense multiplication, since $\mathsf{M}(n) \in \Omega(n\log n)$. □

Lemma 3.3. *Let s be the number of nonzero terms in f. If $\sum d_i \in O(s)$, then the cost of chunky multiplication is never asymptotically greater than the cost of sparse multiplication.*

Proof. The sparse multiplication costs $O(sm)$ ring operations, since g is dense. Now clearly $t \leq s$, and

$$\log\prod(d_i+1) = \sum\log(d_i+1) \leq t + \sum d_i \in O(s).$$

Since $\log m \in O(m)$, this gives a total cost of $O(sm)$ ring operations from Theorem 3.1. The cost of exponent arithmetic will be $O(mt\log t\log n)$, which is less than $O(ms\log s\log n)$ for the sparse algorithm as well. □

It is easy to generate examples showing that these bounds are tight. Unfortunately, this means that there are instances where a single chunky representation will not always result in better performance than the dense *and* sparse algorithms. One such example is when f has $\sqrt{n}$ nonzero terms which do not cluster at all into chunks. Therefore we consider two separate cases for converting to the chunky representation, depending on the representation of the input. When the input is dense, we seek to minimize $\prod(d_i+1)$, and when it is sparse, we seek to minimize $\sum d_i$ (to some extent).

Algorithm `SparseToChunky`

```
Input: $f \in \mathsf{R}[x]$ sparse, and slack variable $\omega \geq 1$
Output: Chunky rep. of $f$ with $\sum d_i \leq \omega s$.
1: $r \leftarrow s$
2: $H \leftarrow$ doubly-linked heap with all possible gaps
   from $f$ and corresponding scores
3: while $r \leq \omega s$ do
4:     Extract gap with highest score from heap
5:     Remove gap from chunky representation, up-
       date neighboring scores, add size of gap to $r$
6: end while
7: Put back in the most recently removed gap
8: return Chunky rep. with all gaps in $H$
```

Algorithm `DenseToChunky`

```
Input: $f \in \mathsf{R}[x]$ in the dense representation
Output: Chunky rep. of $f$ with $\prod(d_i + 1) \in O(n)$
1: $G \leftarrow$ stack of gaps, initially empty
2: $i \leftarrow 0$
3: for each gap $S$ in $f$, moving left to right do
4:     $k \leftarrow |S|$
5:     while $P(S_1, \ldots, S_k, S) \neq$ true do
6:         Pop $S_k$ from $G$ and decrement $k$
7:     end while
8:     Push $S$ onto $G$
9: end for
10: return Chunky rep. with all gaps in $G$
```

3.2 Conversion from Sparse

Lemma 3.3 indicates that minimizing the sum of the degrees of the chunks will guarantee competitive performance with the sparse algorithm. But the minimal value of $\sum d_i$ is actually achieved when we make every chunk completely dense, with no spaces within any dense chunk. While this approach will always be at least as fast as sparse multiplication, it will usually be more efficient to allow some spaces in the chunks if we are multiplying f by a dense polynomial g of any degree larger than 1.

Our approach to balancing the need to minimize $\sum d_i$ and to allow some spaces into the chunks will be the use of a *slack variable*, which we call ω. Really this is just the constant hidden in the big-O notation when we say $\sum d_i$ should be $O(s)$ as in Lemma 3.3.

Algorithm `SparseToChunky` to convert from the sparse to the chunky representation is given below. We first insert every possible gap between totally dense chunks into a *doubly-linked heap*. This is a doubly-linked list embedded in a max-heap, so that each gap in the heap has a pointer to the locations of adjacent gaps.

The key for the max-heap will be a score we assign to each gap. This score will be the ratio between the value of $\prod(d_i + 1)$ with and without the gap included, raised to the power $(1/r)$, where r is the length of the gap. So high "scores" indicate an improvement in the value of $\prod(d_i + 1)$ will be achieved if the gap is included, and not too much extra space will be introduced.

We then continually remove the gap with the highest score from the top of the heap, "fill in" that gap in our representation (by combining surrounding chunks), and update the scores of the adjacent gaps. Since we have a doubly-linked heap, and there can't be more gaps than the number of terms, this is accomplished with $O(s)$ word operations at each step. There are at most s steps, for a total cost of $O(s \log s)$, which is linear in the size of the input from the lower bound on the size of the sparse representation. So we have:

Theorem 3.4. *Algorithm* `SparseToChunky` *returns a chunky representation satisfying* $\sum d_i \leq \omega s$ *and runs in* $O(s \log s)$ *time, where* s *is the number of nonzero terms in the input polynomial.*

3.3 Conversion from Dense

Converting from the dense to the chunky representation is more tricky. This is due in part to that fact that, unlike with the previous case, the trivial conversion does not give a minimum value for the function we want to minimize, which in this case is $\prod(d_i + 1)$.

Let $S_1, S_2, \ldots, S_k$ denote gaps of zeroes between dense chunks in the target representation, ordered from left to right. The algorithm is based on the predicate function $P(S_1, S_2, \ldots, S_k)$, which we define to be true iff inserting all gaps $S_1, \ldots, S_k$ into the chunky representation gives a smaller value for $\prod(d_i + 1)$ than only inserting S_k. Since these gaps are in order, we can evaluate this predicate by comparing the products of the sizes of the chunks formed between $S_1, \ldots, S_k$ and the length of the single chunk formed to the left of S_k.

Algorithm DenseToChunky performs the conversion. We maintain a stack of gaps $S_1, \ldots, S_k$ satisfying $P(S_1, \ldots, S_i)$ is true for all $2 \leq i \leq k$. This stack is updated as we move through the array in a single pass; those gaps remaining at the end of the algorithm are exactly the ones returned in the representation.

Theorem 3.5. *Algorithm* DenseToChunky *always returns a representation containing the maximal number of gaps which satisfies* $\prod(d_i + 1) \leq n$ *and runs in* $O(n)$ *time, where the degree of the input is less than* n*.*

Proof. For correctness, first observe that $P(S_1, \ldots, S_k, S_\ell)$ is true only if $P(S_1, \ldots, S_k, S_{\ell'})$ is true for all $\ell' \leq \ell$. Hence, by induction, the first time we encounter the gap S_i and add it to the stack, the stack is trimmed to contain the maximal number of gaps seen so far which do not increase $\prod(d_i + 1)$. When we return, we have encountered the last gap S_t which of course is required to exist in the returned representation since no nonzero terms come after it. Therefore, from the definition of P, inserting all the gaps we return at the end gives a smaller value for $\prod(d_i + 1)$ than using no gaps.

The complexity comes from the fact that we push or pop onto G at every iteration through either while loop. Since we only make one pass through the polynomial, each gap can be pushed and popped at most once. Therefore the total number if iterations is linear in the number of gaps, which is never more than $n/2$.

To make each calculation of $P(S_1, \ldots, S_k, S)$ run in constant time, we just need to save the calculated value of $\prod(d_i + 1)$ each time a new gap is pushed onto the stack. This means the next product can be calculated with a single multiplication rather than k of them. Also note that the product of degrees stays bounded by n, so intermediate products do not grow too large. □

The only component missing here is a slack variable ω. For a practical implementation, the requirement that $\prod(d_i + 1) \leq n$ is too strict, resulting in slower performance. So, as before, we will only require $\sum \log(d_i + 1) \leq \omega \log n$, which means that $\prod(d_i + 1) \leq n^\omega$, for some positive constant ω. This changes the definition and computation of the predicate function P slightly, but otherwise does not affect the algorithm.

4 Implementation

A complete implementation of adaptive chunky multiplication of dense polynomials has been produced using Victor Shoup's C++ library NTL [12], and is available for download from the author's website. This is an ideal medium for implementation, as our algorithms rely heavily on dense polynomial arithmetic being as fast as possible, and NTL implements asymptotically fast algorithms for dense univariate polynomial arithmetic, and in fact is often cited as containing some of the fastest such implementations.

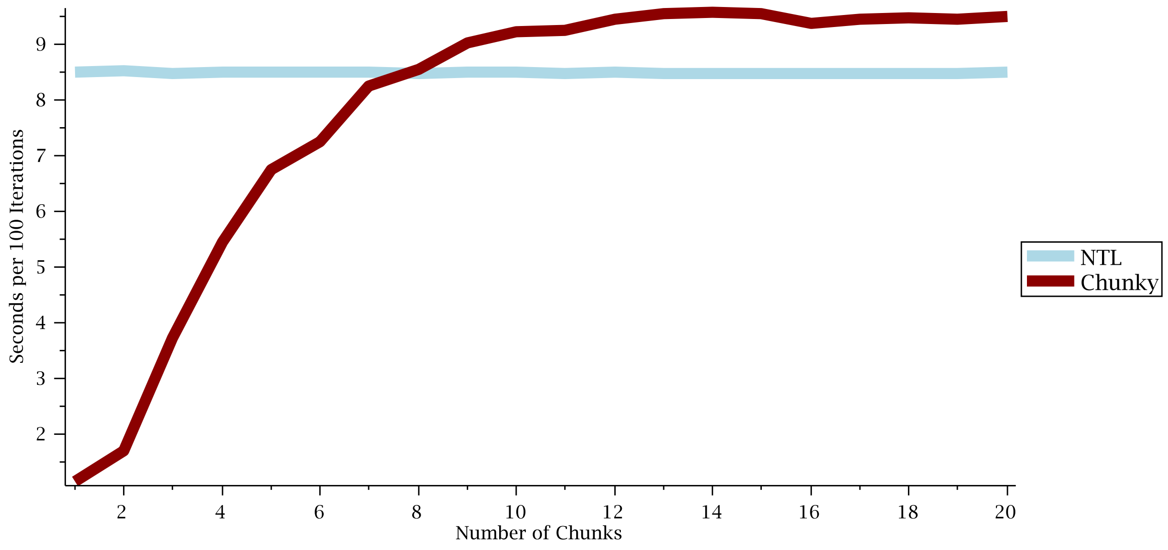
Although the algorithms we have described work over an arbitrary ring R, recall that in our analysis we have made a number of assumptions about R: Ring elements use constant storage, ring operations have unit cost, and the multiplication of degree-n polynomials over R can be performed with $O(n \log n)$ ring operations. To give the best reflection of our analysis, our tests were performed in a ring which makes all these assumptions true: $\mathbb{Z}_p$, where p is a word-sized "FFT prime" which has a high power of 2 dividing $p-1$.

As in any practical implementation, especially one that hopes to ever be competitive with a highly-tuned library such as NTL, we employed a number of subtle "tricks", mostly involving attempts to keep memory access low by performing computations in-place whenever possible. We also had to implement the obvious algorithm to multiply a high-degree polynomial by a low-degree one as mentioned at the beginning of 3.1, since (surprisingly) NTL apparently does not have this built-in.

For the timings results shown in Figure 1, we attempted to only vary the "chunkyness" of one of the input polynomials, and keep everything else fixed. So for all tests, we multiplied a polynomial f with degree 100 000 and 1 000 nonzero terms by a completely dense (random) polynomial g of degree 10 000. The nonzero coefficients of f were partitioned into a varying number of chunks, from 1 to 20. The chunks were randomly positioned in the polynomial, and each chunk was approximately 50% dense. Finally, for these tests, we chose $\omega = 3$ for the slack variable, and the machine used had a 2.4 GHz Opteron processor with 1MB cache and 16GB RAM.

Since the degree and sparsity of both input polynomials remained fixed, we would expect any standard dense or sparse algorithm to have roughly the same running time for any of the tests we performed. In fact,

Figure 1: Timing comparisons for Chunky Multiplication



we can see that this was the case for the dense algorithm in NTL that we compared ours to. According to our analysis, we would expect the chunky algorithm to perform better than the dense one when the input is chunky, and never more than a constant factor worse. In fact, we see that this is the case; our algorithm performs better up to about 8 chunks, and then seems to approach a horizontal plateau which is only about 10% worse than the dense algorithm.

These results are promising, but the crossover point is still fairly low, indicating that more improvement is likely possible. We also tried other values for the slack variable ω. Although these tests are not shown, the effect was as one might predict: a higher value for ω results in a higher crossover point (better performance for chunky input), but also a higher "plateau" value (worse performance for non-chunky input).

5 Other Ideas for Adaptive Multiplication

5.1 Equally-Spaced Terms

Suppose many of the terms of a polynomial $f \in \mathsf{R}[x]$ are spaced equally apart. If the length of this common distance is k, then we can write $f(x)$ as $f_D(x^k)$, where $f_D \in \mathsf{R}[x]$ is dense with degree less than n/k. Now say we want to multiply f by another polynomial $g \in \mathsf{R}[x]$, where $\deg g < m$, and without loss of generality assume $m \leq n$, and similarly write $g(x) = g_D(x^\ell)$. If $k = \ell$, then to find the product of f and g, we just compute $h_D = f_D g_D$ and write $f \cdot g = h_D(x^k)$. The total cost is then just $O((n/m)\mathsf{M}(m/k))$.

If $k \neq \ell$, the algorithm is a bit more complicated, but we still get a significant improvement. Let r and s be the greatest common divisor and least common multiple of k and ℓ, respectively. Split f into ℓ/r polynomials, each with degree less than n/s, as follows:

$$f(x) = f_0(x^s) + f_1(x^s) \cdot x^k + \cdots + f_{l/r-1}(x^s) \cdot x^{s-k}.$$

Similarly, split g into k/r polynomials $g_0, g_1, \ldots, g_{s/k-1}$, each with degree less than m/s. Then to multiply f by g, we compute all products $f_i g_j$, then multiply by powers of x and sum to obtain the final result. The total complexity in this more general case is $O((n/r)\mathsf{M}(m/s))$. So even when k and ℓ are relatively prime, we still perform the multiplication faster than any dense method.

As usual, identifying the best way to convert an arbitrary polynomial into this representation will be the most challenging step algorithmically. We will actually want to write f as $f_D(x^k) + f_S$, where $f_S \in \mathsf{R}[x]$ is sparse with very few nonzero terms, representing the "noise" in the input. To determine k, we must find the gcd of "most" of the exponents of nonzero coefficients in f, which is a nontrivial problem when we are restricted by the requirement of linear-time complexity. We will not go into further detail here.

5.2 Coefficients in Sequence

This technique is best explained by first considering an example. Let $\mathsf{R} = \mathbb{Z}$, $f = 1+2x+3x^2+\cdots+nx^{n-1}$, and $g = b_0+b_1x+\cdots+b_{n-1}x^{n-1}$, for arbitrary $b_0, b_1 \ldots, b_{n-1} \in \mathbb{Z}$. Let $h = fg$ be the product we wish to compute. Then the first n coefficients of h (starting with the constant coefficient) are $b_0, (2b_0 + b_1), (3b_0 + 2b_1 + b_2), \ldots$. To compute these in linear time, first initialize an accumulator with b_0, and then for $i = 1, 2, \ldots, n-1$, compute the coefficient of x^i by adding b_i to the accumulator, and adding the accumulator to the value of the previous coefficient. The high-order terms can be constructed in the same way.

So we have a method to compute fg in *linear time* for any $g \in \mathsf{R}[x]$. In fact, this can be generalized to the case where the coefficients of f form any arithmetic-geometric sequence. That is, $f = a_0 + a_1x + \cdots$ and there exist constants $c_1, c_2, c_3, c_4 \in \mathsf{R}$ such that $a_i = c_1 + c_2i + c_3c_4^i$ for all i. Even more general sequences are probably possible, but it may be more difficult to quickly identify them.

Now note that, if we wish to multiply $f, g \in \mathsf{R}[x]$, only one of the two input polynomials needs to have sequential coefficients in order to compute the product in linear time. To recognize whether this is the case, we start with the list of coefficients, which will be of the form $(c_1 + c_2i + c_3c_4^i)_{i\geq 0}$ if the polynomial satisfies our desired property. We compute successive differences to obtain the list $(c_2 + c_3(c_4 - 1)c_4^i)_{i\geq 0}$. Computing successive differences once more and then successive quotients will produce a list of all c_4's if the coefficients form an arithmetic-geometric sequence as above. We can then easily find c_1, c_2, c_3 as well.

In practice, we will again want to allow for some "noise", so we will actually write $f = \sum(c_1 + c_2i + c_3c_4^i)x^i + f_S$, for some very sparse polynomial $f_S \in \mathsf{R}[x]$. The resulting computational cost for multiplication will be only $O(n)$ plus a term depending on the size of f_S.

6 Conclusions

We have seen some approaches to multiplying polynomials in such a way that we handle "easier" cases more efficiently, for various notions of easiness. These algorithms have the same worst-case complexity as the best known methods, but will be much faster if the input has certain structure.

However, our preliminary implementation seems to indicate that the input polynomials must be very structured in order to obtain a practical benefit. Adaptive sorting algorithms have encountered the same difficulty, and those algorithms have come into wide use only because almost-sorted input arises naturally in many situations. To bring what may currently be interesting theoretical results to very practical importance, we need to investigate the structure of polynomials that frequently occur *in practice*. Perhaps focusing on specific rings, such as very small finite fields, could also provide more easy cases for adaptive algorithms.

Improvements could also come from eliminating some assumptions we have made. An obvious one is that $\mathsf{M}(n) \in \Omega(n \log n)$ used in the rough analysis for chunky conversion; we know that in practice FFT-based multiplication is only used for sufficiently large dense polynomials. Another more subtle assumption we have made is that each input polynomial must be converted separately. Somehow examining the structure of both polynomials simultaneously could give significant speed-up, and in fact we already have some ideas along these lines for chunky multiplication.

Finally, the details of the latter approaches obviously need to be worked out. In fact, some combination of all three ideas might lead to the best performance on real input. In addition, it would be interesting to compare adaptive multiplication performance in the case of sparse polynomials, where the contrast between the fast dense methods we use here and the standard sparse methods could be more striking.

References

[1] Peter Bürgisser and Martin Lotz. Lower bounds on the bounded coefficient complexity of bilinear maps. *J. ACM*, 51(3):464–482 (electronic), 2004.

[2] David G. Cantor and Erich Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Inform.*, 28(7):693–701, 1991.

[3] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Comp.*, 19:297–301, 1965.

[4] Martin Fürer. Faster integer multiplication. In *STOC '07: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 57–66, New York, NY, USA, 2007. ACM Press.

[5] Joachim von zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge University Press, Cambridge, second edition, 2003.

[6] Erich Kaltofen and Wen-shin Lee. Early termination in sparse interpolation algorithms. *J. Symbolic Comput.*, 36(3-4):365–400, 2003. International Symposium on Symbolic and Algebraic Computation (ISSAC'2002) (Lille).

[7] A. Karatsuba and Yu. Ofman. Multiplication of multidigit numbers on automata. *Dokl. Akad. Nauk SSSR*, 7:595–596, 1963.

[8] Kurt Mehlhorn. *Data structures and algorithms. 1.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1984. Sorting and searching.

[9] Michael B. Monagan and Roman Pearce. Polynomial division using dynamic arrays, heaps, and packed exponent vectors. *Lecture Notes in Computer Science*, 4770:295–315, 2007. Computer Algebra in Scientific Computing (CASC'07).

[10] Ola Petersson and Alistair Moffat. A framework for adaptive sorting. *Discrete Appl. Math.*, 59(2):153–179, 1995.

[11] A. Schönhage and V. Strassen. Schnelle Multiplikation grosser Zahlen. *Computing (Arch. Elektron. Rechnen)*, 7:281–292, 1971.

[12] Victor Shoup. NTL: A Library for doing Number Theory. Online, http://www.shoup.net/ntl/, 2007.

[13] David R. Stoutemeyer. Which polynomial representation is best? In *Proc. 1984 MACSYMA Users' Conference*, pages 221–244, Schenectady, NY, 1984.

[14] Thomas Yan. The geobucket data structure for polynomials. *J. Symbolic Comput.*, 25(3):285–293, 1998.

The `modpn` library: Bringing Fast Polynomial Arithmetic into MAPLE

X. Li, M. Moreno Maza, R. Rasheed, É. Schost
Ontario Research Center for Computer Algebra
University of Western Ontario, London, Ontario, Canada.
{xli96,moreno,rrasheed,eschost}@csd.uwo.ca

One of the main successes of the computer algebra community in the last 30 years has been the discovery of algorithms, called modular methods, that allow to keep the swell of the intermediate expressions under control. Without these methods, many applications of computer algebra would not be possible and the impact of computer algebra in scientific computing would be severely limited. Amongst the computer algebra systems which have emerged in the 70's and 80's, MAPLE and its developers have played an essential role in this area.

Another major advance in symbolic computation is the development of implementation techniques for asymptotically fast (FFT-based) polynomial arithmetic. Computer algebra systems and libraries initiated in the 90's, such as MAGMA and NTL, have been key actors in this effort.

In this extended abstract, we present `modpn`, a MAPLE library dedicated to fast arithmetic for multivariate polynomials over finite fields. The main objective of `modpn` is to provide highly efficient routines for supporting the implementation of modular methods in MAPLE.

We start by illustrating the impact of fast polynomial arithmetic on a simple modular method by comparing its implementations in MAPLE with classical arithmetic and with the `modpn` library. Then, we discuss the design of `modpn`. Finally, we provide an experimental comparison.

1 The impact of fast polynomial arithmetic

To illustrate the speed-up that fast polynomial arithmetic can provide we use a basic example: the solving of a bivariate polynomial system. We give a brief sketch of the algorithm of [LMR08].

Let $F_1, F_2 \in \mathbb{K}[X_1, X_2]$ be two bivariate polynomials over a prime field $\mathbb{K}$. For simplicity, we make three *genericity assumptions* which are easy to relax:

1. F_1 and F_2 have positive degree with respect to X_2,
2. the zero set $V(F_1, F_2) \subset \overline{\mathbb{K}}^2$ is non-empty and finite (where $\overline{\mathbb{K}}$ is an algebraic closure of $\mathbb{K}$),
3. no point in $V(F_1, F_2)$ cancels the GCD of the leading coefficients of F_1 and F_2 with respect to X_2.

Then the algorithm below, ModularGenericSolve2(F_1, F_2), computes a triangular decomposition of $V(F_1, F_2)$.

Input: F_1, F_2 as above

Output: regular chains $(A_1, B_1), \ldots, (A_e, B_e)$ in $\mathbb{K}[X_1, x_2]$ such that $V(F_1, F_2) = \bigcup_{i=1}^{e} V(A_i, B_i)$.

```
ModularGenericSolve2($F_1, F_2$) ==
(1)  **Compute** the subresultant chain of $F_1, F_2$
(2)  **Let** $R_1$ be the resultant of $F_1, F_2$ with respect to $X_2$
(3)  $i := 1$
(4)  **while** $R_1 \notin \mathbb{K}$ **repeat**
(5)        **Let** $S_j \in \mathrm{src}(F_1, F_2)$ regular with $j \geq i$ minimum
```

```
(6)     if $\mathrm{lc}(S_j, X_2) \equiv 0 \mod R_1$
            then $j := i + 1$; goto (5)
(7)     $G := \gcd(R_1, \mathrm{lc}(S_j, X_2))$
(8)     if $G \in \mathbb{K}$
            then output $(R_1, S_j)$; exit
(9)     output $(R_1 \text{ quo } G, S_j)$
(10)    $R_1 := G$; $j := i + 1$
```

In Step (1) we compute the subresultant chain of F_1, F_2 in the following *lazy fashion*:

1. Let B be a bound for the degree of R_1, for instance $B = 2\,d_1 d_2$ where $d_1 := \max(\deg(F_i, X_1))$ and $d_2 := \max(\deg(F_i, X_2))$. We evaluate F_1 and F_2 at $B + 1$ different values of X_1, say $x_0, \ldots, x_B$, such that none of these specializations cancels $\mathrm{lc}(F_1, X_2)$ or $\mathrm{lc}(F_2, X_2)$.
2. For each $i = 0, \ldots, B$, we compute the subresultant chain of $F_1(X_1 = x_i, X_2)$ and $F_2(X_1 = x_i, X_2)$.
3. We interpolate the resultant R_1 and do not interpolate any other subresultant in $\mathrm{src}(F_1, F_2)$.

In Step (5) we consider the regular subresultant S_j of F_1, F_2 with minimum index j greater than or equal to i. We view S_j as a "candidate GCD" of F_1, F_2 modulo R_1 and we interpolate its leading coefficient with respect to X_2. The correctness of this algorithm follows from the block structure theorem and the specialization property of subresultants [GCL92].

We have realized two implementations of this modular algorithm. One is based on classical polynomial arithmetic and is written entirely in MAPLE whereas the other one relies on fast polynomial arithmetic provided by our C low-level routines. Figure 1 below corresponds to experiments with the former implementation and Figure 2 with the latter. In each case, the comparison is made versus the `Triangularize` command of the `RegularChains` library [LMX05]. Note that, over finite fields, the `Triangularize` command does not use any modular algorithms or fast arithmetic.

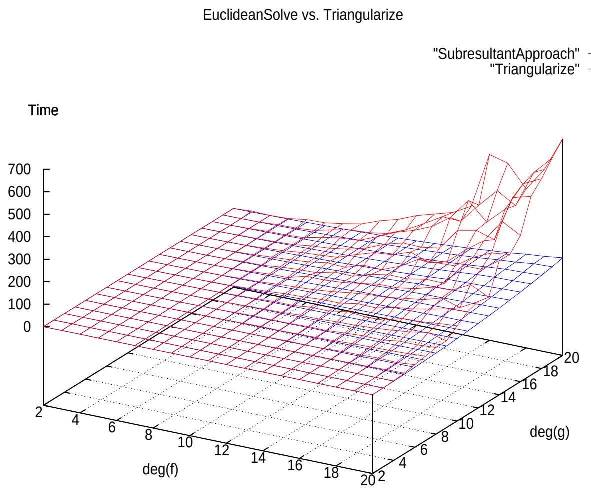


Figure 1: ModularGenericSolve2 vs. `Triangularize` in $\mathbb{Z}/p\mathbb{Z}[X_1, X_2]$, pure MAPLE code

The implementation of ModularGenericSolve2 compared in Figure 1 to `Triangularize` is written purely in MAPLE; both functions rely on MAPLE built-in DAG polynomials. The input systems are random and

d1	d2	Nsols	LexGB	FastTriade	`Triangularize`
10	10	50	0.280	0.044	1.276
15	15	100	1.892	0.104	16.181
20	20	150	6.224	0.208	54.183
25	25	200	15.041	4.936	115.479
15	15	100	1.868	0.100	7.492
20	20	200	14.544	0.308	47.683
25	25	300	49.763	1.268	282.249
30	30	400	123.932	1.152	907.649
20	20	150	6.176	0.188	17.105
25	25	300	50.631	1.852	117.195
30	30	450	171.746	1.341	575.647
35	35	600	445.040	7.260	2082.158
25	25	200	14.969	0.564	40.202
30	30	400	124.680	2.132	238.287
35	35	600	441.416	2.300	1164.244

Figure 2: ModularGenericSolve2 using `modpn` vs. `Triangularize` in $\mathbb{Z}/p\mathbb{Z}[X_1, X_2]$

dense; the horizontal axes correspond to the partial degrees d_1 and d_2. We observe that for input systems of about 400 solutions the speed-up is about 10.

In Figure 2, ModularGenericSolve2 is renamed FastTriade and relies on the `modpn` library. We also provide the timings for the command `Groebner:-Basis` using the `plex` terms order, since this produces the same output as ModularGenericSolve2, and `Triangularize` on our input systems. The invoked Gröbner basis computation consists of a degree basis (computed by the Maple code implementation of the F4 Algorithm) followed by a change basis (computed by the Maple code implementation of the FGLM Algorithm). We observe that for input systems of about 400 solutions the speed-up between ModularGenericSolve2 is now about 100.

2 The design of `modpn`

We designed and implemented a Maple library called `modpn`, which provides fast arithmetic for the polynomial ring $\mathbb{Z}/p\mathbb{Z}[X_1, \ldots, X_n]$, where p is a prime number; currently this library only supports machine word size prime.

Overview. `modpn` is a platform which supports general polynomial computations, and especially modular algorithms for triangular decomposition. The high performance of `modpn` relies on our C package reported in [Li05, FLMS06, LM06, LMS07], together with other new functionalities, such as fast interpolation, triangular Hensel lifting and subresultant chains computations. In addition, `modpn` also integrates Maple Recursive Dense (`RecDen`) polynomial arithmetic package for supporting dense polynomial operations. The calling to C and `RecDen` routines is transparent to the Maple users.

With the support of `modpn`, we have implemented in Maple a high level algorithm for polynomial system solving: regular GCD computations and their special case of bivariate systems, as presented in [LMR08]. The performance of our bivariate is satisfactory and reported in Section 3.

Challenges and solutions. Creating a highly efficient library is one of the most important and challenging components for this work. The difficulties result from the following aspects.

First, we mix Maple code with C code. Maple users may call external C routines using the `ExternalCalling` package. However, the developers need to implement efficient data type converters to transform the Maple level data representation into C level and vice versa. This task was all the more demanding as we used two Maple polynomial encodings and designed another three at C level.

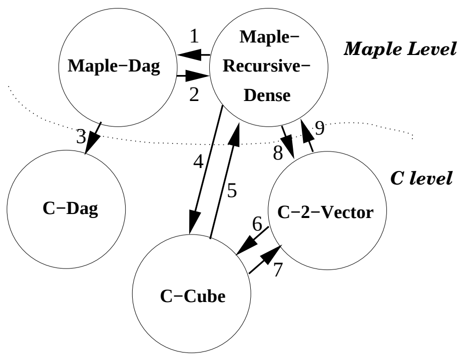


Figure 3: The polynomial data representations.

Second, the C level operations themselves form a complex setup: the top level functions, such as triangular Hensel lifting and subresultant-based methods, rely on interpolation and fast arithmetic modulo a triangular set, which themselves eventually rest on FFT/TFT-based polynomial arithmetic such as fast multiplication and division.

Finally, removing the bottlenecks in the mixed code and identifying cut-offs between different methods is a quite subtle and time-consuming task.

In the following subsections we will report on some technical aspects of the code integration and implementation methods for several core algorithms supported by `modpn`.

2.1 Code integration in Modpn

In [LMS07], we have described how to integrate our C package into AXIOM. Basically, we linked C code directly into the AXIOM kernel to make new functionalities available in the AXIOM interpreter. However, having no access to the MAPLE kernel, the only way to use external C code is to rely on the MAPLE `ExternalCalling` package. The MAPLE level data structures such as DAG's and trees have to be transformed and passed to C level, and vice versa. This step needs to be designed carefully to avoid producing bottlenecks. Moreover, the use of multiple data encodings in our library makes the code integration much harder.

Indeed, we use five polynomial encodings in our implementation, showed in Figure 3. The *Maple-Dag* and *Maple-Recursive-Dense* polynomials are MAPLE built-in types; the *C-Dag*, *C-Cube* and *C-2-Vector* polynomials are written in C. Each encoding is adapted to certain applications; we switch between different representations at run-time.

Maple polynomials. MAPLE polynomial objects by default are encoded as Directed Acyclic Graphs (DAG's). To use built-in MAPLE packages such as `RegularChains` we need such a *Maple-Dag* representation. On the other hand, we rely on the Maple `RecDen` package for several operations which are not implemented yet in our C package.

C polynomials. The *C-Cube* polynomials are our data representation for dense polynomials, already used in [LMS07]. Each polynomial is encoded by a multidimensional array holding all its coefficients (hence the name), whose dimensions are given in advance. This encoding is suitable for dense triangular set arithmetic and FFT/TFT based methods.

We also implemented a *C-Dag* polynomial representation, mainly to support Hensel lifting techniques. The *C-Dag* polynomials are encoded in the adjacency-list representation; each node contains a 32-bit header word for keeping the type, id, visiting history, and liveness information.

To make data conversion to `RecDen` more efficient, we finally designed a so-called *C-2-Vector* encoding, described below.

Conversions. In Figure 3, directed edges describe the conversions we use. Edges 1 and 2 are the conversions between *Maple-Dag* and `RecDen` polynomials; they are provided by the `RecDen` package. We implemented the other conversion functions in Maple and C: conversions between C representations are of course written in C; conversions between the two languages involve two-sided operations (preparing the data on one side, decoding it on the other side).

Edge 3 stands for the conversion from *Maple-Dag* to *C-Dag*. *Maple-Dag* polynomials are traversed and packed into an array; this array is passed to and unpacked at the C-level, where common sub-expressions are identified using a hash table.

As mentioned before, the *C-Cube* is the canonical data representation in our fast polynomial arithmetic package; edges 4-9 serve the purpose of communicating between this format and `RecDen`. Edges 4 and 5 are in theory sufficient for this task. However, the *C-Cube* polynomials are in dense encoding, including all leading zeros up to some pre-fixed degree bound. This is the appropriate data structure for most operations modulo triangular sets; however, this raises the issue of converting all useless zeros back to Maple.

To make these conversions more efficient, we used our so-called *C-2-Vector* encoding, which essentially matches `RecDen` encoding, together with edges 6-9. Roughly, a multivariate polynomial is represented by a tree structure; each sub-tree is a coefficient polynomial. Precisely, in the *C-2-Vector* encoding, we use one vector to encode the degrees of all sub-trees in their main variables, and another vector to hold the coefficients, using the same traversal order. Thus, to locate a coefficient, we use the degree vector for finding indices. This encoding avoids to use any C pointer, so it can be directly passed back from C to Maple and decoded as a `RecDen` polynomial in a straightforward manner.

2.2 The Modpn Maple level

`modpn` appears to the users a pure Maple library. However, each `modpn` polynomial contains a `RecDen` encoding, a C encoding, or both. The philosophy of this design is still based on our long-term strategy: implementing the efficiency-critical operations in C and more abstract algorithms in higher level languages. When using `modpn` library, *Maple-Dag* polynomials will be converted into `modpn` polynomials. The computation of `modpn` will be selectively conducted by either `RecDen` or our C code up, depending on the application. Then, the output of `modpn` can be converted back to *Maple-Dag* by another function call.

2.3 The Modpn C level

The basic framework of our C implementation was described in [FLMS06, LMS07]: it consists of fast finite field arithmetic, FFT/TFT based univariate/multivariate polynomial arithmetic and computation modulo a triangular set, such as normal form, inversion and gcd. In this paper, we implemented higher-level algorithms on top of our previous code: interpolation, Hensel lifting and subresultant chains.

Triangular Hensel lifting. We implemented the *Hensel lifting of a regular chain* [Sch03] since this is a fundamental operation for polynomial system solving. The solver presented in [DJMS08] is based on this operation and is implemented in the `RegularChains` library. For simplicity, our recall of the specifications of the Hensel lifting operation is limited to regular chains consisting of two polynomials in three variables.

Let $X_1 < X_2 < X_3$ be ordered variables and let F_1, F_2 be in $\mathbb{K}[X_1, X_2, X_3]$. Let $\mathbb{K}(X_1)$ be the field of univariate rational functions with coefficients in $\mathbb{K}$. We denote by $\mathbb{K}(X_1)[X_2, X_3]$ the ring of bivariate polynomials in X_2 and X_3 with coefficients in $\mathbb{K}(X_1)$. Let π be the projection on the X_1-axis. For $x_1 \in \overline{\mathbb{K}}$, we denote by Φ_{x_1} the evaluation map from $\mathbb{K}[X_1, X_2, X_3]$ to $\overline{\mathbb{K}}[X_2, X_3]$ that replaces X_1 with x_1. We make two assumptions on F_1, F_2. First, the ideal $\langle F_1, F_2 \rangle$, generated by F_1 and F_2 in $\mathbb{K}(X_1)[X_2, X_3]$, is radical. Secondly, there exists a triangular set $\mathcal{T} = \{T_2, T_3\}$ in $\mathbb{K}(X_1)[X_2, X_3]$ such that $\mathcal{T}$ and F_1, F_2 generate the same ideal in $\mathbb{K}(X_1)[X_2, X_3]$. Under these assumptions, the following holds: for all $x_1 \in \overline{\mathbb{K}}$, if x_1 cancels no denominator in $\mathcal{T}$, then the fiber $V(F_1, F_2) \cap \pi^{-1}(x_1)$ satisfies

$$V(F_1, F_2) \cap \pi^{-1}(x_1) \; = \; V(\Phi_{x_1}(T_2), \Phi_{x_1}(T_3)).$$

We are ready to specify the Hensel lifting operation. Let x_1 be in $\mathbb{K}$. Let $N_2(X_2), N_3(X_2, X_3)$ be a triangular set in $\mathbb{K}[X_2, X_3]$, monic in its main variables, and with N_3 reduced with respect to N_2, such that we have

$$V(\Phi_{x_1}(F_1), \Phi_{x_1}(F_2)) = V(N_2, N_3).$$

We assume that the Jacobian matrix of $\Phi_{x_1}(F_1), \Phi_{x_1}(F_2)$ is invertible modulo the ideal $\langle N_2, N_3\rangle$. Then, the Hensel lifting operation applied to F_1, F_2, N_2, N_3, x_1 returns the triangular set $\mathcal{T}$. Using a translation if necessary, we assume that x_1 is zero. This simplifies the rest of the presentation.

The Hensel lifting algorithm progressively recovers the dependency of $\mathcal{T}$ on the variable X_1. At the beginning of the kth step, the coefficients of $\mathcal{T}$ are known modulo $\langle X_1^{\ell-1}\rangle$, with $\ell = 2^k$; at the end of this step, they are known modulo $\langle X_1^{\ell}\rangle$. Most of the work consists in reducing the input system and its Jacobian matrix modulo $\langle X_1^{\ell}, N_2, N_3\rangle$, followed by some linear algebra, still modulo $\langle X_1^{\ell}, N_2, N_3\rangle$.

To reduce F_1, F_2 modulo $\langle X_1^{\ell}, N_2, N_3\rangle$, we rely on our DAG representation. We start from X_1, X_2, X_3, which are known modulo $\langle X_1^{\ell}, N_2, N_3\rangle$; then, we follow step-by-step the DAG for F_1, F_2, and perform each operation (addition, multiplication) modulo $\langle X_1^{\ell}, N_2, N_3\rangle$. A "visiting history" bit is kept in the headword for each node to avoid multiple visits; a "liveness" bit is used for nullifying a dead node.

To reduce the Jacobian matrix of F_1, F_2, we proceed similarly. We used the plain (direct) automatic differentiation mode to obtain a DAG for this matrix, as using the reverse mode does not pay off for square systems such as ours. Then, this matrix is inverted using standard Gaussian elimination.

In this process, modular multiplication is by far the most expensive operation, justifying the need for the FFT / TFT based multiplication algorithms presented in [LMS07]. The other operations are relatively cheap: rational reconstruction uses extended Euclidean algorithm (we found that even a quadratic implementation did not create a bottleneck); the stop criterion is a reduction in dimension zero, much cheaper than all other operations.

Evaluation and interpolation. Our second operation is the implementation of fast multivariate evaluation and interpolation on multidimensional rectangular grid (which thus reduces to tensored versions of univariate evaluation / interpolation). When having primitive roots of unity in the base field, and when these roots of unity are not "degenerate cases" for the problem at hand (e.g., do not cancel some leading terms, etc), we use multidimensional DFT/TFT to perform evaluation and interpolation at those roots. For more general cases, we use the algorithms based on subproduct-tree techniques [GG99, Chap. 10]. To optimize the cache locality, before performing evaluation / interpolation, we transpose the data of our C-cube polynomials, such that every single evaluation / interpolation pass will go through a block of contiguous memory.

3 Experimental results

We describe in this section a series of experiments for the various algorithms mentioned before. Our comparison platforms are MAPLE 11 and MAGMA V2.14-8 [BCP97]; all tests are done on a Pentium 4 CPU, 2.80GHz, with 2 GB memory. All timings are in seconds. For all our tests, the base field is $\mathbb{Z}/p\mathbb{Z}$, with $p = 962592769$ (with one exception, see below).

Bivariate systems solving. We extend our comparison of bivariate system solvers to MAGMA. As above, we consider random dense, thus generic, systems. Experimentation with non-generic, and in particular non-equiprojectable systems will be reported in another paper. We choose partial degrees d_1 (in X_1) and d_2 (in X_2); the input polynomials have support $X_1^i X_2^j$, with $i \leq d_1$ and $j \leq d_2$, and random coefficients. Such random systems are in Shape Lemma position: no splitting occurs, and the output has the form $T_1(X_1), T_2(X_1, X_2)$, where $\deg(T_1, X_1) = d_1 d_2$ and $\deg(T_2, X_2) = 1$.

In Table 1 an overview of the running time of many solvers. In MAPLE, we compare the `Basis` and `Solve` commands of the `Groebner` package to the `Triangularize` command of the `RegularChains` package and our code. In MAGMA, we use the `GroebnerBasis` and `TriangularDecomposition` commands; the columns in the table follow this order. Gröbner bases are computed for lexicographic orders.

MAPLE uses the FGb software for Gröbner basis computations over some finite fields. However, our large Fourier base field is not handled by FGb; hence, our `Basis` experiments are done modulo $p' = 65521$, for which FGb can be used. This limited set of experiment already shows that our code performs quite well. To be fair, we add that for MAPLE's `Basis` computation, most of the time is spent in basis conversion, which is interpreted MAPLE code: for the largest example, the FGb time was 0.97 sec.

We refine these first results by comparing in Figure 4 our solver with MAGMA's triangular decomposition

d_1	d_2	Maple				Magma	
		`Basis`	`Solve`	`Trig`	`us`	`GB`	`Trig`
11	2	0.3	37	12	0.1	0.03	0.03
11	5	3	306	62	0.13	0.11	0.12
11	8	18	1028	122	0.16	0.32	0.32
11	11	27	2525	256	0.2	0.61	0.66

Table 1: Generic bivariate systems: all solvers

for larger degrees. It quickly appears that our code performs better; for the largest examples (having about 5700 solutions), the ratio is about 460/7.

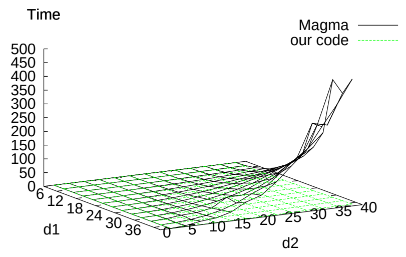


Figure 4: Generic bivariate systems: Magma vs. us.

Triangular Hensel Lifting. We conclude with testing our implementation of the Hensel lifting algorithm for a regular chain. As opposed to the previous problem, there is not a distributed package to which we could compare; hence, our reference tests are run with the original Magma implementation presented in [Sch03]. The underlying algorithms are the same; only implementations differ.

We generated trivariate systems (F_1, F_2) in $\mathbb{K}[X_1, X_2, X_3]$. Seeing X_1 as a free variable, these systems admit a Gröbner basis of the form $T_2(X_1, X_2), T_3(X_1, X_2, X_3)$ in $\mathbb{K}(X_1)[X_2, X_3]$. In our experiments, we set $\deg(T_3, X_3)$ to 2 or 4. This was achieved by generating random sparse systems f_1, f_2, and taking

$$F_1 = \prod_{j=0}^{k-1} f_1(X_1, X_2, \omega^j X_3), \quad F_2 = \prod_{j=0}^{k-1} f_2(X_1, X_2, \omega^j X_3),$$

with $\omega = -1$ or $\sqrt{-1}$, and correspondingly $k = 2$ or 4. These systems were generated by Maple and kept in unexpanded form, so that both lifting implementations could benefit from their low complexity of evaluation. We show in Figure 5 and 6 the results obtained for the cases $\deg(T_3, X_3) = 2$ and $\deg(T_3, X_3) = 4$, respectively. For the largest examples, the ratio in our favor is $21032/3206 \approx 6.5$.

4 Conclusion

To our knowledge, `modpn` is the first library making FFT/TFT-based multivariate arithmetic available to Maple end users. As illustrated in this short report, this can improve the implementation of modular algorithms in a spectacular manner. We are currently re-implementing the core operations of the `RegularChains` library by means of such algorithms creating opportunities for using the `modpn` library.

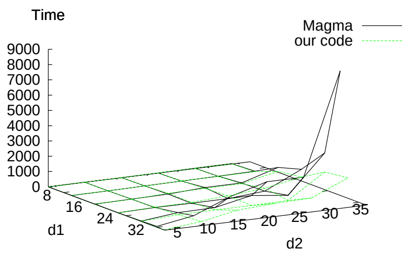


Figure 5: Lifting, MAGMA vs. us.

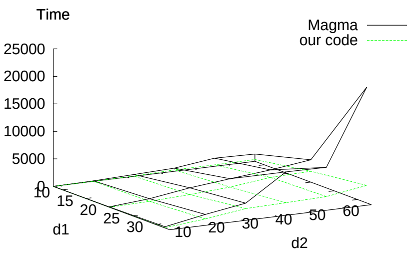


Figure 6: Lifting, MAGMA vs. us.

Acknowledgement
All authors acknowledge the continuing support of Waterloo Maple Inc., the Mathematics of Information Technology and Complex Systems (MITACS) and the Natural Sciences and Engineering Research Council of Canada (NSERC). The third Author acknowledges the support of the Canada Research Chairs Program.

References

[BCP97] W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997.

[DJMS08] X. Dahan, X. Jin, M. Moreno Maza, and É Schost. Change of ordering for regular chains in positive dimension. *Theoretical Computer Science*, 392(1-3):37–65, 2008.

[FLMS06] A. Filatei, X. Li, M. Moreno Maza, and É. Schost. Implementation techniques for fast polynomial arithmetic in a high-level programming environment. In *Proc. ISSAC'06*, pages 93–100, New York, NY, USA, 2006. ACM Press.

[GCL92] K. O. Geddes, S. R. Czapor, and G. Labahn. *Algorithms for Computer Algebra*. Kluwer Academic Publishers, 1992.

[GG99] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.

[Li05] X. Li. Efficient management of symbolic computations with polynomials, 2005. University of Western Ontario.

[LM06] X. Li and M. Moreno Maza. Efficient implementation of polynomial arithmetic in a multiple-level programming environment. In A. Iglesias and N. Takayama, editors, *Proc.* International Congress of Mathematical Software - ICMS 2006, pages 12–23. Springer, 2006.

[LMR08] X. Li, M. Moreno Maza, and R. Rasheed. Fast arithmetic and modular techniques for polynomial gcds modulo regular chains, 2008.

[LMS07] X. Li, M. Moreno Maza, and É. Schost. Fast arithmetic for triangular sets: From theory to practice. In *Proc. ISSAC'07*, pages 269–276. ACM Press, 2007.

[LMX05] F. Lemaire, M. Moreno Maza, and Y. Xie. The `RegularChains` library. In Ilias S. Kotsireas, editor, Maple Conference 2005, pages 355–368, 2005.

[Sch03] É. Schost. Complexity results for triangular sets. *J. Symb. Comp.*, 36(3-4):555–594, 2003.

The Maximality of Dixon Matrices on Corner-Cut Monomial Supports by Almost-Diagonality

Eng-Wee Chionh
School of Computing
National University of Singapore
Republic of Singapore 117590

April 4, 2008

Abstract

The maximality of the Dixon matrix on corner cut monomial supports with at most three exposed points at the corners is established using almost-diagonal Dixon matrices. The search of these matrices is conducted using Maple.

1 Introduction

Computer algebra systems have made great progress over the decades since their advent in the late sixties. Contemporary computer algebra systems are capable of computing complicated mathematical objects effectively on a personal computer. Some of these systems, notably Maple and Mathematica, exploit visualization aggressively to give further insights to the process and the result of a computation. These strengths have made them a standard tool for many research endeavors, especially in theorem proving by experimentation. Indeed this paper reports the partial proof of a theorem that owes much of its formulation and success to the extensive use of Maple.

The theorem concerns the maximality of the Dixon matrix when its monomial support undergoes corner cutting. It asserts that the Dixon matrix is maximal if and only if the number of exposed points at any corner is at most three. The theorem plays a significant role in constructing Dixon sparse resultants for corner-cut monomial supports. For if the matrix constructed is maximal, the theory of sparse resultants assures that its determinant is a multiple of the resultant. Thus if the extraneous factors are known in advance, a closed form sparse resultant formula in quotient form emerges. Currently resultants are the most efficient among the various elimination methods such as Grobner bases. Corner-cut monomial supports occur naturally in shape modeling utilising multi-sided toric patches. These two facts suggest that the seek of explicit closed form sparse resultant formulas should be profitable to solving polynomial systems in general and processing of toric patches in particular.

A standard technique in the theory of resultants for establishing the maximality of a general square matrix is to show that a specialized version of the matrix is diagonal. To fashion a proof along this technique, Maple was used to search for polynomials on a corner-cut monomial support (with at most three exposed points at any corner) that led to a diagonal Dixon matrix. The search was guided by two conflicting requirements. On one hand there should be as few monomials as possible to maximize the number of zero entries in the matrix; on the other hand there should be enough of them so that the dimension of the matrix does not degenerate. The attempt was not completely successful because either sparse or almost-diagonal matrices instead of exact-diagonal matrices were found for all cases except one. Fortunately, for almost-diagonal matrices,

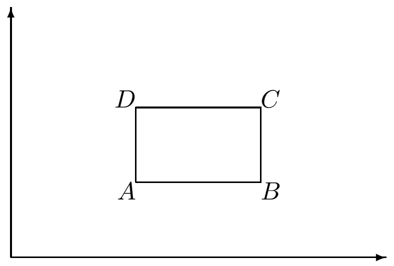


Figure 1: The rectangle is denoted by any of $A..C = C..A = B..D = D..B$.

maximality can be shown with some additional effort. But for the remaining cases when the matrices are only sparse rather than almost-diagonal, a different technique is needed to establish maximality.

The rest of the paper consists of the following sections. Section 2 presents the technical background and known results needed in the paper. Section 3 states the main result and sketches its proof. Section 4 concludes the paper with a short summary.

2 Preliminaries

Let $\mathbf{Z}$ and $\mathbf{R}$ be the set of integers and reals respectively. The real euclidean plan $\mathbf{R} \times \mathbf{R}$ contains the set of lattice points $\mathbf{Z} \times \mathbf{Z}$. A key concept in this research is that of a rectangular set of lattice points whose sides are parallel to the axes. Such a rectangular set can be identified by any of its two pairs of diagonal vertices p, q as $p..q$ or $q..p$ (Figure 1). In particular, we denote the special rectangular set $(0,0)..(m,n)$ as $S_{m,n}$. That is,

$$S_{m,n} = (0,0)..(m,n) = (m,n)..(0,0) = (m,0)..(0,n) = (0,n)..(m,0). \tag{1}$$

Given a set of lattice points $S \subseteq \mathbf{Z} \times \mathbf{Z}$, the bi-degree hull of S is the rectangular set

$$[S] = (\min S_x, \min S_y)..(\max S_x, \max S_y) \tag{2}$$

where S_x, S_y are respectively the sets of x, y coordinates of points of S. We shall write

$$S \sqsubseteq S_{m,n} \tag{3}$$

to mean the bi-degree hull of S is $S_{m,n}$. The set of the four corners of $S_{m,n}$ is denoted

$$K_{m,n} = \{(0,0), (m,0), (m,n), (0,n)\}. \tag{4}$$

Consider a set $S \sqsubseteq S_{m,n}$. A point $p \in S_{m,n} \setminus S$ is an exterior point of S with respect to the corner $\kappa \in K_{m,n}$ if

$$(\kappa..p) \cap S = \emptyset. \tag{5}$$

A point $p \in S$ is an exposed point of S with respect to the corner $\kappa \in K_{m,n}$ if

$$(\kappa..p) \cap S = \{p\}. \tag{6}$$

The sets of exterior and exposed points with respect to the corner κ will be written E_κ and X_κ respectively. For example, consider

$$S = \{(1,0), (0,1), (1,1), (2,1), (1,2)\} \sqsubseteq S_{2,2}, \tag{7}$$

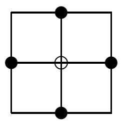


Figure 2: Exposed points are marked by filled circles. Exterior points are unmarked.

the sets of exterior points and exposed points of S at the corners are (Figure 2)

$$\begin{array}{|c|c|}\hline E_{(0,2)} & E_{(2,2)} \\ \hline E_{(0,0)} & E_{(2,0)} \\ \hline \end{array} = \begin{array}{|c|c|}\hline \{(0,2)\} & \{(2,2)\} \\ \hline \{(0,0)\} & \{(2,0)\} \\ \hline \end{array}, \tag{8}$$

$$\begin{array}{|c|c|}\hline X_{(0,2)} & X_{(2,2)} \\ \hline X_{(0,0)} & X_{(2,0)} \\ \hline \end{array} = \begin{array}{|c|c|}\hline \{(0,1),(1,2)\} & \{(1,2),(2,1)\} \\ \hline \{(0,1),(1,0)\} & \{(1,0),(2,1)\} \\ \hline \end{array}. \tag{9}$$

Given three bivariate polynomials

$$[f(s,t),g(s,t),h(s,t)] = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty}[f_{ij},g_{ij},h_{ij}]s^i t^j, \tag{10}$$

their monomial support is the lattice set

$$S = \{(i,j) : [f_{ij},g_{ij},h_{ij}] \neq 0\}. \tag{11}$$

Given a lattice set $S \sqsubseteq S_{m,n}$, the Dixon polynomial on S is the polynomial

$$P(S) = \frac{1}{(s-\alpha)(t-\beta)} \begin{vmatrix} f(s,t) & g(s,t) & h(s,t) \\ f(\alpha,t) & g(\alpha,t) & h(\alpha,t) \\ f(\alpha,\beta) & g(\alpha,\beta) & h(\alpha,\beta) \end{vmatrix}, \tag{12}$$

where S is the monomial support of f, g, h. The Dixon matrix $D(S)$ on S is the coefficient matrix of $P(S)$. That is,

$$P(S) = [\cdots, s^\sigma t^\tau, \cdots] \;\; D(S) \;\; [\cdots, \alpha^a\beta^b, \cdots]^T = \mathcal{R}(S)D(S)\mathcal{C}(S). \tag{13}$$

To be definite we assume the monomial order $t < s$ for the monomials $s^\sigma t^\tau$ in the set of row indices $\mathcal{R}(S)$ and the monomial order $\beta < \alpha$ for the monomials $\alpha^a\beta^b$ in the set of column indices $\mathcal{C}(S)$.

The row support $R(S)$ and column support $C(S)$ of $D(S)$ are respectively the set of exponents (σ,τ) in $\mathcal{R}(S)$ and the set of exponents (a,b) in $\mathcal{C}(S)$. For example, we have

$$R(S_{m,n}) = S_{m-1,2n-1}, \quad C(S_{m,n}) = S_{2m-1,n-1}. \tag{14}$$

A bracket (i,j,k,l,p,q) is a 3×3 determinant formed with the coefficients of the polynomials f, g, h:

$$(i,j,k,l,p,q) = \begin{vmatrix} f_{ij} & g_{ij} & h_{ij} \\ f_{kl} & g_{kl} & h_{kl} \\ f_{pq} & g_{pq} & h_{pq} \end{vmatrix}. \tag{15}$$

The entries of $D(S)$ are sums of brackets. The following proposition gives the row and column indices where a bracket appears [1].

Proposition 1 *The row indices $s^\sigma t^\tau$ and column indices $\alpha^a\beta^b$ of the entries where the bracket (i,j,k,l,p,q), $i \le k \le p$, appears are*

$$i \le \sigma \le k-1, j+\min(l,q) \le \tau \le j+\max(l,q)-1, a = i+k+p-1-\sigma, b = j+l+q-1-\tau; \tag{16}$$

and

$$k \le \sigma \le p-1, q+\min(j,l) \le \tau \le q+\max(j,l)-1, a = i+k+p-1-\sigma, b = j+l+q-1-\tau. \tag{17}$$

Corner cutting is the process of introducing exterior points to $S_{m,n}$ to obtain monomial support S. The following proposition tells the effect of corner cutting on the dimension of $D(S)$ [4].

Proposition 2 *For a monomial support $S \sqsubseteq S_{m,n}$, the dimension of $D(S)$ and the number of exterior points of S are related by the formula*

$$dim(D(S)) = 2mn - \sum_{\kappa \in K_{m,n}} |E_\kappa|. \tag{18}$$

3 The Maximality Theorem

With the preceding formalism, the theorem we seek to prove can be concisely stated as:

Theorem 1 *Let $S \sqsubseteq S_{m,n}$. Then*

$$|D(S)| \neq 0 \tag{19}$$

if and only if

$$\forall \kappa \in K_{m,n}, |X_\kappa| \leq 3. \tag{20}$$

The "only if" part of the theorem is proved in [4]. To establish the "if" part of the theorem, we consider all possible exposed point configurations at the corners $K_{m,n} \subseteq S_{m,n}$. By symmetry, there are exactly fifteen cases

$$1113, 1123, 1213, 1133, 1313, 1223, 1232, 1233, 1323, 1333, 2223, 2233, 2323, 2333, 3333 \tag{21}$$

to investigate, where the group of four digits *abcd* denotes the configuration that the numbers of exposed points at corners $(0,0)$, $(m,0)$, (m,n), $(0,n)$ are a, b, c, d respectively. For each case, we look for admissible monomial support $S \sqsubseteq S_{m,n}$ and special bivariate polynomials f, g, h such that $D(S)$ is as "diagonal" as possible and conditions (19), (20) hold. By experimenting on Maple, the admissible monomial support S (Figure 3) for each case and the corresponding polynomials f, g, h found are tabulated as follows.

Case	f	g	h	$\dim(D(S))$
1113	1	$s^{m-1}t+s^m$	s^mt^n	$mn+1$
1123	1	$s^{m-2}t+s^m$	$s^{m-1}t^n$	$mn+1$
1213	1	$s^{m-1}t+s^mt$	s^mt^n	$mn-m+1$
1133	$s^{m-2}t^n$	$s^{m-3}t+s^m$	$1+s^{m-1}t$	$mn+2$
1313	1	st	s^mt^n	$m+n-2$
1223	1	$st+s^mt$	s^2t^n	$mn-1$
1232	$s^{m-2}t^n$	$1+s^{m-1}t^{n-1}$	s^mt^{n-2}	$2(m+n)-3$
1233	$1+s^{m-1}t^{n-1}$	$s^{m-3}t^{n-2}+s^mt^{n-2}$	$s^{m-2}t^n$	$2m+3n-5$
1323	$s^{m-2}t^{n-2}$	$1+s^{m-1}t^n$	$s^{m-1}t^n+s^mt^{n-1}$	$2(m+n)-5$
1333	$s^{m-2}t^n+s^mt^{n-2}$	$st+s^{m-1}t^{n-1}$	$1+s^mt^{n-2}$	$3(m+n)-9$
2223	s^2t^n	$s+st^{n-1}$	$t^{n-2}+s^mt^{n-1}$	mn
2233	$t+s^{m-1}t^2$	$s^{m-2}+s^{m-2}t^n$	$s^{m-3}t^2+s^mt$	$mn+2$
2323	$t+s^mt^{n-1}$	$s+s^{m-1}t^n$	s^2t^2	$3(m+n)-10$
2333	$t+s^mt^{n-2}$	$s^2t^2+s^{m-1}t^{n-1}$	$s+s^{m-2}t^n$	$4(m+n)-16$
3333	$t^2+s^mt^{n-2}$	$st+s^3t^3+s^{m-1}t^{n-1}$	$s^2+s^{m-2}t^n$	$5(m+n)-24$

Among the fifteen case there are three levels of diagonality: exact-diagonal (case 1313), sparse triangular (cases 1113, 1123, 1213, 1133, 1223, 1232, 1233, 1323, 2223, 2323), and sparse (cases 1333, 2233, 2333, 3333). The structure of the Dixon matrix $D(S)$ in each of these cases is revealed by Proposition 1. In the following discussions, to apply the formulas given in the proposition more conveniently, we shall denote the rectangular set of lattice points $(x_1, y_1)..(x_2, y_2)$, $x_1 \le x_2$, as $x_1..x_2 \times y_1..y_2$; that is,

$$x_1..x_2 \times y_1..y_2 = (x_1, y_1)..(x_2, y_2). \tag{22}$$

For the case 1313, there is only one bracket $(0, 0, 1, 1, m, n)$. Proposition 1 says that the row support is

$$R(S_{1313}) = (0..0 \times 1..n-1) \cup (1..m-1 \times n..n). \tag{23}$$

Thus $D(S_{1313})$ is diagonal and its dimension is clearly $m+n-2$, consistent with Proposition 2.

For the case 1113, there are two brackets $(0, 0, m-1, 1, m, n)$ and $(0, 0, m, 0, m, n)$. The row indices of the first bracket are

$$(0..m-2 \times 1..n-1) \cup (m-1..m-1 \times n..n), \tag{24}$$

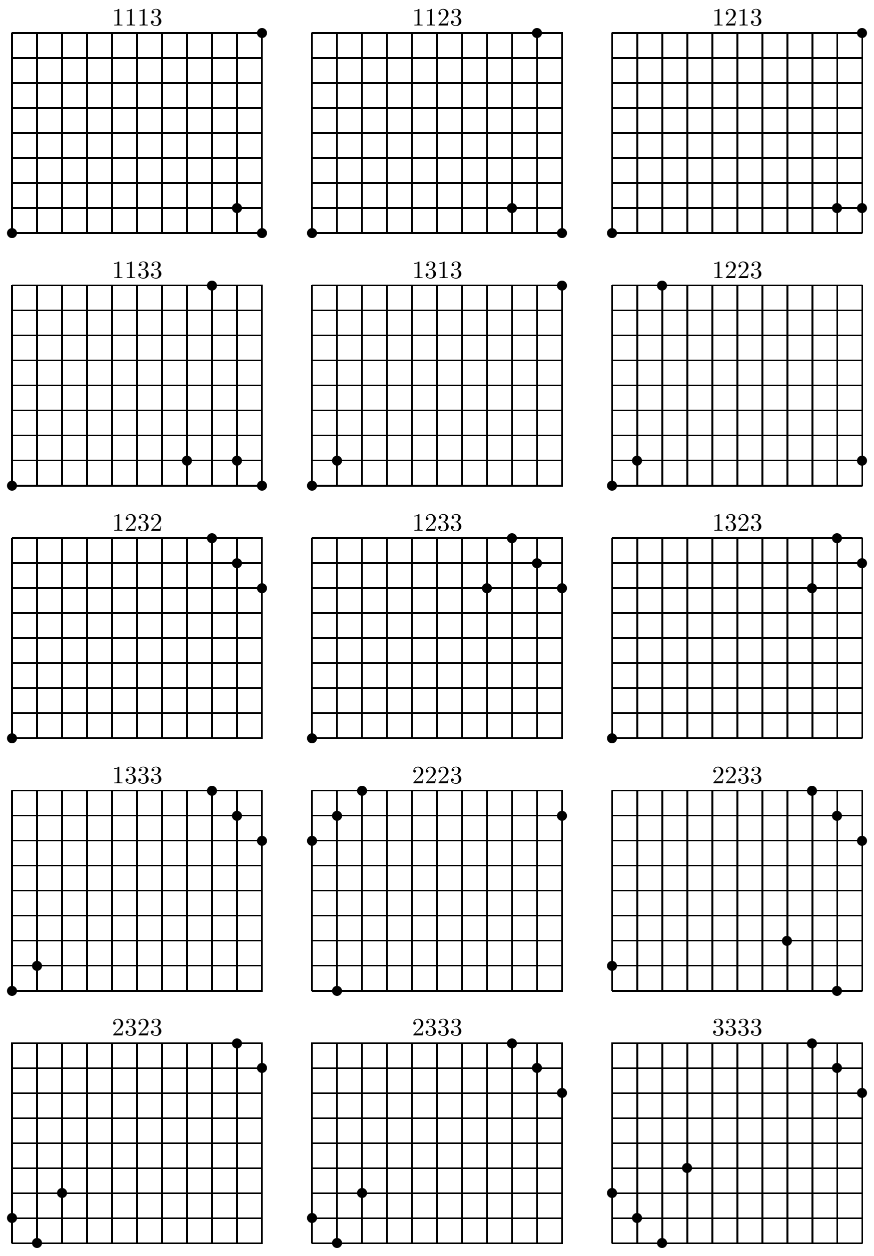


Figure 3: The monomial support corresponds to each of the fifteen cases.

and that of the second bracket are

$$0..m-1 \times 0..n-1. \tag{25}$$

Thus the row support is

$$(0..m-1 \times 0..n-1) \cup (m-1..m-1 \times n..n). \tag{26}$$

By also examining the column indices, it can be checked that $D(S_{1113})$ is upper triangular with a bottom-left to top-right diagonal. The dimension of $D(S_{1113})$ is clearly $mn+1$, again consistent with Proposition 2.

The claim that the Dixon matrix in other sparse triangular cases is upper triangular, after some permutation of rows, can be verified in a similar manner.

For the sparse cases 1333, 2233, 2333, and 3333, the structure of the Dixon matrix $D(S)$ is much more complicated and it is not obvious that it is maximal. However, specialized supports using line cutting or point pasting can be used to show maximality [3].

4 Conclusion

The paper proved the maximality of the Dixon matrix on corner cut monomial supports with at most three exposed points at the corners. The approach was to specialize the support and the polynomials so that in particular the Dixon matrix is maximal. Since in particular it is maximal so in general it has to be maximal. There were fifteen cases to be considered depending on the numbers of exposed points at the corners. This approach was successful in eleven of the fifteen cases as diagonal or sparse triangular Dixon matrix were found. For the remaining four cases, the current approach does not seem to apply. Extending the current approach to these cases will be a future effort.

References

[1] E. W. Chionh, *Parallel Dixon matrices by bracket*, Advances in Computational Mathematics, 19:373–383, 2003.

[2] A. L. Dixon, *The eliminant of three quantics in two independent variables*, Proc. London Math. Soc., 6:49–69, 473–492, 1908.

[3] M. C. Foo, *Dixon $\mathcal{A}$-Resultant Formulas*, Master thesis, National University of Singapore, 2003.

[4] W. Xiao, *Loose Entry Formulas and the Reduction of Dixon Determinant Entries*, Master thesis, National University of Singapore, 2004.

Symbolic Polynomials with Sparse Exponents

Stephen M. Watt
Ontario Research Centre for Computer Algebra
Department of Computer Science, University of Western Ontario
London Ontario, CANADA N6A 5B7
watt@uwo.ca

Abstract

Earlier work has presented algorithms to factor and compute GCDs of symbolic Laurent polynomials, that is multivariate polynomials whose exponents are integer-valued polynomials. These earlier algorithms had the problem of high computational complexity in the number of exponent variables and their degree. The present paper solves this problem, presenting a method that preserves the structure of sparse exponent polynomials.

1 Introduction

We are interested in the algebra of polynomials whose exponents are not known in advance, but rather are given by integer-valued expressions, for example $x^{2m^2+n} + 3x^n y^{m^3+1} + 4$. In particular, we consider the case where the exponents are integer-valued polynomials with coefficients in $\mathbb{Q}$. One could imagine other models for integer-valued expressions, but this seems sufficiently general for a number of purposes. We call these "symbolic polynomials." Symbolic polynomials can be related to exponential polynomials [1] and to families of polynomials with parametric exponents [2, 3, 4].

To date, computer algebra systems have only been able to do simple ring operations on symbolic polynomials. They can add and multiply symbolic polynomials, but not much else. In earlier work, we have given a technical definition of symbolic polynomials, have shown that these symbolic polynomials over the integers form a UFD, and have given algorithms to compute GCDs and factor them [5, 6]. These algorithms fall into two families: *extension methods*, based on the algebraic independence of variables to different monomial powers (*e.g.* x, x^n, x^{n^2},...), and *homomorphism methods*, based on the evaluation and interpolation of exponent polynomials.

There is a problem with these earlier algorithms, however: they become impractical when the exponent polynomials are sparse. Extension methods introduce an exponential number of new variables and homomorphism methods require an exponential number of images. We have attempted to address this by performing sparse interpolation of exponents [7, 8], but this leads to impractical factorizations in the image polynomial domain.

This paper presents solves these problems. We show a substitution for the extension method that introduces only a linear number of new variables. The resulting polynomials are super-sparse and may be factored by taking images using Fermat's little theorem, as done by Giesbrecht and Roche [9]. (Indeed, Fermat's little theorem can be used in a second stage of projection for our homomorphism method, but there combining images is more complicated.)

The remainder of the paper is organized as follows: Section 2 recalls a few elementary facts about integer-valued polynomials and fixed divisors. Section 3 summarizes the extension algorithm that we have presented earlier for dense exponents. Section 4 explains why this algorithm is not suitable for the situation when the exponent polynomials are sparse and shows how to deal with this problem. Section 5 presents the extension algorithms adapted to sparse exponents and Section 6 concludes the paper.

2 Preliminaries

We recall the definitions of *integer-valued* polynomial and *fixed divisor*, and note some of their elementary properties.

Definition 1 (Integer-valued polynomial)**.** For an integral domain D with quotient field K, the (univariate) integer-valued polynomials over D, denoted $\mathrm{Int}(D)$, are defined as

$$\mathrm{Int}(D) = \{f \mid f \in K[X] \text{ and } f(a) \in D, \text{ for all } a \in D\}$$

For example, $\frac{1}{2}n^2 - \frac{1}{2}n \in \mathrm{Int}(\mathbb{Z})$ because if $n \in \mathbb{Z}$, either n or $n-1$ is even. Integer-valued polynomials have been studied for many years, with classic papers dating back 90 years [10, 11]. We make the obvious generalization to multivariate polynomials.

Definition 2 (Multivariate integer-valued polynomial)**.** For an integral domain D with quotient field K, the (multivariate) integer-valued polynomials over D in variables $X_1, \ldots, X_n$, denoted $\mathrm{Int}_{[X_1,\ldots,X_n]}(D)$, are defined as

$$\mathrm{Int}_{[X_1,\ldots,X_n]}(D) = \{f \mid f \in K[X_1, \ldots, X_n] \text{ and } f(a) \in D, \text{ for all } a \in D^n\}$$

For consistency we will use the notation $\mathrm{Int}_{[X]}(D)$ for univariate integer-valued polynomials.

When written in the binomial basis, integer-valued polynomials have the following useful property:

Property 1. *If f is a polynomial in $\mathrm{Int}_{[n_1,\ldots,n_p]}(\mathbb{Z}) \subset \mathbb{Q}[n_1, \ldots n_p]$, then when f is written in the basis $\binom{n_1}{i_1} \cdots \binom{n_p}{i_p}$, its coefficients are integers.*

If a polynomial is integer-valued, then there may be a non-trivial common divisor of all its integer evaluations.

Definition 3 (Fixed divisor)**.** A fixed divisor of an integer-valued polynomial $f \in \mathrm{Int}(D)$ is a value $q \in D$ such that $q|f(a)$ for all $a \in D$.

Given the following result, it is easy to compute the largest fixed divisor of a multivariate integer-valued polynomial.

Property 2. *If f is a polynomial in $Z[n_1, \ldots, n_p]$, then the largest fixed divisor of f may be computed as the* gcd *of the coefficients of f when written in the binomial basis.*

3 Algorithms for Dense Exponents

Following earlier work [5, 6] we define the ring of symbolic polynomials as follows:

Definition 4 (Ring of symbolic polynomials)**.** The ring of symbolic polynomials in $x_1, \ldots, x_v$ with exponents in $n_1, \ldots, n_p$ over the coefficient ring R is the ring consisting of finite sums of the form

$$\sum_i c_i x_1^{e_{i1}} x_2^{e_{i2}} \cdots x_n^{e_{in}}$$

where $c_i \in R$ and $e_{ij} \in \mathrm{Int}_{[n_1,n_2,...,n_p]}(\mathbb{Z})$. Multiplication is defined by

$$c_1 x_1^{e_{11}} \cdots x_n^{e_{1n}} \quad \times \quad c_2 x_1^{e_{21}} \cdots x_n^{e_{2n}} = c_1 c_2 x_1^{e_{11}+e_{21}} \cdots x_n^{e_{1n}+e_{2n}}$$

We denote this ring $R[n_1, ..., n_p; x_1, ..., x_v]$.

A more elaborate definition is available that allows symbolic exponents on constants from the coefficient ring and everything we say here can be carried over.

We have already shown [5, 6] that symbolic polynomials with integer coefficients form a UFD. The first ingredient of the proof is that x^n and x^{n^2} are algebraically independent. The second ingredient is that fixed divisors become explicit when integer-valued polynomials are written in a binomial basis. The conversion to the binomial basis detects fixed divisors. For example, $n^2 + n$ is even for any integer n so we must detect that $x^{n^2+n} - 1$ is a difference of squares:

$$x^{n^2+n} - 1 = (x^{\frac{1}{2}n^2+\frac{1}{2}n} + 1)(x^{\frac{1}{2}n^2+\frac{1}{2}n} - 1)$$

This leads to the extension algorithms. For example, for factorization we have:

Dense Extension Algorithm for Symbolic Polynomial Factorization

Input: A symbolic polynomial $f \in \mathbb{Z}[n_1, ...n_p; x_1, ..., x_v]$.

Output: The factors $g_1, ..., g_n$ such that $\prod_i g_i = f$, unique up to units.

1. Put the exponent polynomials of f in the basis $\binom{n_i}{j}$.
2. Construct polynomial $F \in \mathbb{Z}[X_{10...0}, ..., X_{vd_1...d_p}]$, where d_i is the maximum degree of n_i in any exponent of f, using the correspondence
$$\gamma : x_k^{\binom{n_1}{i_1} \cdots \binom{n_p}{i_p}} \mapsto X_{ki_1...i_p}.$$
3. Compute the factors G_i of F.
4. Compute $g_i = \gamma^{-1}(G_i)$.

Under any evaluation map on the exponents, $\phi : \mathrm{Int}_{[n_1,...,n_p]}(\mathbb{Z}) \to \mathbb{Z}$, if $\phi(f)$ factors into $f_{\phi 1}, ..., f_{\phi r}$ these factors may be grouped to give the factors $\phi(g_i)$. That is, there is a partition of $\{1, ..., r\}$ into subsets I_i such that $\phi(g_i) = \prod_{j \in I_i} f_{\phi j}$. This factorization into g_i is the maximal uniform factorization in the sense that any other factorization g_i' has $\forall_i \exists_j g_i \mid g_j'$.

4 Sparse Exponents

The problem with the previous algorithm is that the change to the binomial basis makes the exponent polynomials dense. If all exponent variables are of degree d or less, the new factorization involves $v \times (d+1)^p$ indeterminates. If the number of exponent variables or their degree is large, then the problem becomes difficult. We solve this by introducing a different substitution.

In factorization and related algorithms, the reason we have transformed the exponents of the input symbolic polynomial to a binomial basis is so that all factors will have exponent polynomials with integer coefficients. Then, because $x^{cb_i} = \left(x^{b_i}\right)^c$, we can treat the algebraically independent x^{b_i} as new polynomial variables.

The only thing that matters, really, is that the coefficients of the factored symbolic polynomials' exponents be integers.

We can achieve the same effect by scaling the original variables and using a Pochhammer basis for the exponent polynomials. Any polynomial in variables z_i may be written in terms of the basis $(z_i)_{(j)}$ and *vice versa*, in the same coefficient ring. The binomial coefficients and the Pochhammer symbols are related by $\binom{x}{j} = (x)_{(j)}/j!$ so multiplying the exponent polynomials by a suitable constant will make them integer-valued. To see this, we make use of the following result.

Lemma 1. *If $h \in \mathrm{Int}_{[n_1,...,n_p]}(\mathbb{Z})$ is of degree at most d in each of the n_i, then $d!^p \times h \in \mathbb{Z}[n_1, ..., n_p]$.*

Proof. Because h is integer-valued, we can write

$$d!^p \times h = \sum_{0 \le i_1,...,i_p \le d} h_{i_1,...,i_p} d!^p \binom{n_1}{i_1} \cdots \binom{n_p}{i_p} \qquad h_{i_1,...,i_p} \in \mathbb{Z}.$$

If $0 \le i \le d$, then $d!\binom{w}{i} = (d \times \cdots \times (d-i+1)) \times (w \times \cdots \times (w-i+1)) \in \mathbb{Z}[w]$ and the result is immediate. □

We now use this to avoid having to make a change of basis.

Theorem 1. *If $f \in \mathcal{P} = R[n_1, ..., n_p; x_1, ..., x_v]$ has factors $g_i \in \mathcal{P}$ with exponents in $\mathrm{Int}_{[n_1,...,n_p]}(\mathbb{Z})$ with each exponent of degree at most d in any n_i, then making the substitution $x_i \mapsto X_i^{d!^p}$ gives factors in $R[n_1, ..., n_p, X_1, ..., X_v]$ with exponents in $\mathbb{Z}[n_1, ..., n_p]$.*

Proof. Let $\mathbf{exdeg}_n f$ denote the maximum degree in n of any exponent polynomial in f. By hypothesis, we have $\max_i \mathbf{exdeg}_{n_i} f = d$. Then for all g_i and n_j we have $\mathbf{exdeg}_{n_j} g_i \le \mathbf{exdeg}_{n_j} f$. Therefore all exponent polynomials occurring in any g_i are elements of $\mathrm{Int}_{[n_1,...,n_p]}(\mathbb{Z})$ of degree at most d in any n_i. By Lemma 1, multiplying all the exponent polynomials by $d!^p$ will give exponent polynomials in $\mathbb{Z}[n_1, ..., n_p]$. Making the substitution $x_i \mapsto X_i^{d!^p}$ multiplies the exponent polynomials in exactly this way. □

The exponent multiplier given by the change of variables $x_i \mapsto X_i^{d!^p}$ may be larger than required to give integer coefficients in the exponents of the factors. This may lead to factors whose exponents are not integer-valued polynomials when the change of variables is inverted. It is easy to give an example of such an "over factorization" resulting from too large a multiplier. Suppose we wish to factor

$$f = x^{n^3+n^2} - x^{n^3} + x^{n^2} - 1.$$

The substitution from the theorem is $x \mapsto X^{3!}$ and this gives

$$\begin{aligned} f &= X^{6n^3+6n^2} - X^{6n^3} + X^{6n^2} - 1 \\ &= (X^{n^3})^6 (X^{n^2})^6 - (X^{n^3})^6 + (X^{n^2})^6 - 1. \end{aligned}$$

This then factors as

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

The these factors do not have integer-valued polynomials as exponents. Combinations of these factors, however, do:

$$\left(x^{\frac{1}{3}n^2}+1\right)\times\left(x^{\frac{2}{3}n^2}-x^{\frac{1}{3}n^2}+1\right)=x^{n^2}+1$$
$$\left(x^{\frac{1}{6}n^3}-1\right)\times\left(x^{\frac{1}{3}n^3}+x^{\frac{1}{6}n^3}+1\right)\times\left(x^{\frac{1}{6}n^3}+1\right)\times\left(x^{\frac{1}{3}n^3}-x^{\frac{1}{6}n^3}+1\right)=x^{n^3}-1$$

Because $\mathbb{Z}[n_1,...,n_p;x_1,...,x_v]$ is a UFD, there will be a grouping of factors that leads to a unique fullest factorization, up to units.

5 Algorithms for Sparse Exponents

The transformation given in Section 4 allows us to adapt the dense exponent algorithms for symbolic polynomial factorization, GCD, *etc* to sparse exponents. In each case we substitute the variables for a suitable power, compute the result, combine factors and substitute back. We show the algorithm for factorization of symbolic polynomials in more detail:

Sparse Extension Algorithm for Symbolic Polynomial Factorization

INPUT: A symbolic polynomial $f \in \mathcal{P} = \mathbb{Z}[n_1,...n_p;x_1,...,x_v]$.

OUTPUT: The factors $g_1,...,g_n$ such that $\prod_i g_i = f$, unique up to units.

1. Construct $E = \rho f \in \mathbb{Z}[n_1,...,n_p;X_1,...X_v]$, using the substitution
$$\rho : x_i \mapsto X_i^{d!^p}.$$
2. Construct $F = \gamma E \in \mathbb{Z}[X_{10...0},...,X_{vd...d}]$, using the correspondence
$$\gamma : X_k^{{n_1}^{i_1}\cdots {n_p}^{i_p}} \mapsto X_{ki_1...i_p}.$$
3. Compute the factors G_j of F.
4. Compute $H_j = \gamma^{-1}(G_j)$.
5. Find the finest partition $\mathcal{H}_1 \cup \cdots \cup \mathcal{H}_N$ of $\{H_j\}$ such that for all $\mathcal{H}_i$ we have $g_i = \rho^{-1}\left(\prod_{G\in\mathcal{H}_i} G\right) \in \mathcal{P}$.

This gives the maximal uniform factorization of the symbolic polynomial f. We may compute the GCD and related quantities similarly.

We make a few general observations:

In Step 1, we need not necessarily substitute all variables with $x_i \mapsto X_i^{d!^p}$. The exponents of each x_i form independent spaces so we may calculate separate bounds b_i and substitute $x_i \mapsto X_i^{b_i}$. If any x_i has fewer than all p exponent variables or if some exponent variables have lower degrees, then the corresponding b_i will be lower.

In Step 2, if the original exponent polynomials are sparse, then most of the variables will not appear in F. In particular, the number of variables in F is at most linear in the size of the input polynomial.

The polynomial F will be supersparse: We have replaced the problem of having a number of new variables exponential in d with the problem of increasing the number of bits in the exponent coefficients by $p\log(d!)/\log 2 = O(pd\log d)$. In general, factoring super-sparse polynomials is intractable in a complexity theoretic sense as there may be dense factors of high degree. Likewise, the corresponding GCD problem can be reduced to an NP-complete problem [12]. In our problem, however, the symbolic polynomial factorization must be valid for all values of the exponent variables n_i. In particular, the symbolic polynomial polynomial factorization $\prod_i g_i$ evaluated with $\phi : \{n_1, ..., n_p\} \to 0$ will be a (possibly incomplete) factorization of the polynomial ϕf. The number of terms in the final symbolic polynomial factorization is therefore unaffected by the multiplication of the exponent polynomials by a large constant.

In Step 3, we may reduce the size of the exponents that occur in the factorization of F by taking several images using Fermat's little theorem for small primes. That is, if a variable x is going to be evaluated by a homomorphism to give an image problem, then reduce first using $x^p \equiv x \pmod{p}$. This has idea been observed by other authors (for example [9]).

In Step 5 we can limit the combinations that need be considered by examining only those for which the sum of asymptotically leading (and, separately, trailing) exponents give integer-valued polynomials.

6 Conclusions

We have shown how to preserve the sparsity of exponents in problems related to the factorization of of symbolic polynomials. We do this by making a change of variables that guarantees the exponents of the output polynomials will have integer coefficients.

We have implemented this method in Maple and have found it to allow factorizations of symbolic polynomials far larger than any we have been able to achieve using other methods. For the first time we appear to have an algorithm of reasonable practical complexity for computing the factorization of symbolic polynomials.

References

[1] C.W. Henson, L. Rubel and M. Singer, *Algebraic properties of the ring of general exponential polynomials.* Complex Variables Theory and Applications, **13** (1989) 1-20.

[2] V. Weispfenning, *Gröbner bases for binomials with parametric exponents.* Technical report, Universität Passau, Germany, 2004.

[3] K. Yokoyama, *On systems of algebraic equations with parametric exponents.* Proc. ISSAC 2004, July 4-7, 2004, Santander, Spain, ACM Press, 312-319.

[4] W. Pan and D. Wang. *Uniform Gröbner bases for ideals generated by polynomials with parametric exponents.* Proc. ISSAC 2006, ACM Press, 269–276.

[5] S.M. Watt, *Making computer algebra more symbolic.* Proc. Transgressive Computing 2006: A conference in honor of Jean Della Dora, April 24–26, 2006, Granada, Spain, 44–49.

[6] S.M. Watt, *Two families of algorithms for symbolic polynomials.* Computer Algebra 2006: Latest Advances in Symbolic Algorithms — Proceedings of the Waterloo Workshop I. Kotsireas, E. Zima (editors), World Scientific 2007, 193–210.

[7] M. Malenfant and S.M. Watt, *Sparse exponents in symbolic polynomials.* Symposium on Algebraic Geometry and Its Applications: In honor of the 60th birthday of Gilles Lachaud (SAGA 2007) (Abstracts), May 7–11 2007, Papeete, Tahiti.

[8] M. Malenfant, *A comparison of two families of algorithms for symbolic polynomials.* MSc.Thesis, Dept of Computer Science, University of Western Ontario, December 2007.

[9] M. Giesbrecht and D. Roche, *Interpolation of shifted-lacunary polynomials*. Proc. Mathematical Aspects of Computer and Information Sciences (MACIS), 2007.

[10] A. Ostrowski, *Über ganzwertige Polynome in algebraischen Zahlköpern*. J. Reine Angew. Math., **149** (1919), 117–124.

[11] G. Pólya, *Über ganzwertige Polynome in algebraischen Zahlköpern*. J. Reine Angew. Math., **149** (1919), 97–116.

[12] D. A. Plaisted, *New NP-hard and NP-complete polynomial and integer divisibility problems*. Theoret. Comput. Sci., **31** (1984), 125–138.

Barycentric Birkhoff Interpolation

Extended Abstract

D.A. Aruliah
University of Ontario Institute of Technology
Canada

J. C. Butcher
University of Auckland
New Zealand

R. M. Corless, A. Shakoori
University of Western Ontario
Canada

L. Gonzalez–Vega[*]
Universidad de Cantabria
Spain

Introduction

The problem of interpolating an unknown function $f : \mathbb{R} \to \mathbb{R}$ by an univariate polynomial with the knowledge of the values of f and some of its derivatives at some points in $\mathbb{R}$ is one of the main problems in Numerical Analysis and Approximation Theory.

Let n and r be two integer numbers such that $n \geq 1$ and $r \geq 0$ and

$$\mathcal{E} = \begin{pmatrix} e_{1,0} & \dots & e_{1,r} \\ \vdots & & \vdots \\ e_{n,0} & \dots & e_{n,r} \end{pmatrix}$$

with $e_{i,j} = 0$ or $e_{i,j} = 1$, for every i and j. The matrix $\mathcal{E}$, usually known as the incidence matrix, is going to codify the information we have about f (and its derivatives).

Let $\mathcal{X} = \{x_1, \dots, x_n\}$ be a set of real numbers such that $x_1 < \dots < x_n$ (the nodes) and $\mathcal{F}$ be a matrix of given real numbers (the known values of f and its derivatives) with the same dimensions as $\mathcal{E}$ whose elements are denoted by $f_{i,j}$. The problem of determining a polynomial P in $\mathbb{R}[x]$ with degree smaller or equal than r which interpolates $\mathcal{F}$ at $(\mathcal{X}, \mathcal{E})$, i.e. which satisfies the conditions:

$$P^{(j)}(x_i) = f_{i,j} \qquad \text{iff} \qquad e_{i,j} = 1$$

is known as the *Birkhoff Interpolation Problem.*

The Birkhoff Interpolation scheme given by $\mathcal{E}$ is said to be poised if for any matrix $\mathcal{F}$ there exists an unique P in $\mathbb{R}[x]$ with degree smaller or equal than r interpolating $\mathcal{F}$ at $(\mathcal{X}, \mathcal{E})$.

To clarify the previous formalism, next we present one example. The Birkhoff Interpolation scheme given by the matrix

$$\mathcal{E} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

is poised (see [3]). This means that for any given real numbers $x_1 < x_2 < x_3$ and a, b, c, d there exist one and only one degree 3 polynomial $P(x)$ in $\mathbb{R}[x]$ such that:

$$P(x_1) = a,\ P^{(2)}(x_1) = b,\ P^{(1)}(x_2) = c,\ P^{(3)}(x_3) = d.$$

[*]Partially supported by the Spanish Ministerio de Educacion y Ciencia grant MTM2005-08690-C02-02.

This polynomial is:

$$P(x) = \frac{d}{6}x^3 + \frac{b - dx_1}{2}x^2 + \left(-\frac{d}{2}x_2^2 + dx_1x_2 - x_2b + c\right)x + \frac{d}{3}{x_1}^3 - \frac{b}{2}x_1^2 + \frac{d}{2}x_2^2x_1 - dx_1^2x_2 + bx_1x_2 - cx_1 + a$$

In [4] the so called barycentric representation for Hermite interpolants was introduced bringing as main advantages a better numerical stability and the possibility of treating simultaneously, both, polynomial and rational interpolation. And, in [2], these interpolants were used for event location in initial value problems through a new companion matrix pencil whose generalized eigenvalues provided such events.

In this paper, by using some of the techniques in [1], we extend this formalism to the Birkhoff Interpolation problem by showing that we can produce the corresponding barycentric interpolant if and only if the considered Birkhoff Interpolation Scheme is poised. If, in the Hermite case, the coefficients of the barycentric interpolant were found by solving a partial fraction problem, for the Birkhoff case, a further solution step is required in order to get the basic polynomials needed to represent conveniently the searched interpolant.

A companion matrix pencil for the Birkhoff case like the one derived in [2] for the Hermite case can be easily derived from this one.

1 Deriving the Hermite barycentric interpolant

Specifying a univariate polynomial P by its values $\rho_{i,0}$ on nodes τ_i for $1 \leq i \leq n$, or by an unbroken sequence of local Taylor coefficients $\rho_{i,j}$, for $1 \leq i \leq n$, $0 \leq j \leq s_i - 1$, where the integers s_i are the confluencies at each node τ_i and

$$\rho_{i,j} = \frac{P^{(j)}(\tau_i)}{j!}$$

has been shown to be useful.

The main tools in using polynomials in this way, without changing basis, are the so called barycentric forms, e.g.

$$P(t) = \omega(t) \sum_{i=1}^{n} \sum_{j=0}^{s_i-1} \frac{\gamma_{i,j}}{(t-\tau_i)^{j+i}} \sum_{k=0}^{j} \rho_{i,k}(t-\tau_i)^k \tag{1}$$

where

$$\omega(t) = \prod_{i=1}^{n} (t-\tau_i)^{s_i} \ ,$$

and the companion matrix pencil associated with this form, which enables zero–finding without (unstable) conversion to the monomial on Newton basis.

This extended abstract outlines an extension of the barycentric form idea to the Birkhoff interpolation case, that is, the case where there is missing data in the sequence of local Taylor coefficients $\rho_{i,j}$. Moreover in this case we have an easy way to get a companion matrix pencil for performing root–finding tasks: use the obtained interpolant to fill in the missing values, and then just use the companion matrix pencil for the Hermite Interpolation Scheme introduced in [2].

We begin by rederiving equation (1) using a contour integral technique from [1]. Define

$$\Phi(z;t) = -\frac{1}{z-t} + \sum_{i=1}^{n} \sum_{j=0}^{s_i-1} \frac{\gamma_{i,j}\omega(t)}{(z-\tau_i)^{j+i}}$$

for the barycentric weights $\gamma_{i,j}$, $1 \leq i \leq n$ and $0 \leq j \leq s_i - 1$ (see [2]). Put

$$d = -1 + \sum_{i=1}^{n} s_i$$

ans suppose $d \geq 0$. Then, for all polynomials $P(z)$ with $\deg P \leq d$, we have, for a large enough contour C,

$$\frac{1}{2\pi i}\oint_C \Phi(z)P(z)\mathrm{d}z = 0$$

because (it can be shown that) the degree of the denominator is $d+2$, while the degree of the numerator (in z) is d. Expansion of the integral by residues gives equation (1).

2 Deriving the Birkhoff barycentric interpolant

Now we consider the Birkhoff case where not all $\rho_{i,j}$ are known. Let J_i be the index set for which $\rho_{i,j}$ is known and

$$J_i^c = \{0, 1, \ldots, s_i - 1\} - J_i$$

the set for which $\rho_{i,j}$ is not known. Put

$$m = \sum_{i=1}^{n} \# J_i^c \ ,$$

the number of missing pieces.

Define

$$\phi(z;t) = -\frac{1}{z-t} + \sum_{i=1}^{n}\sum_{j \in J_i} \frac{\alpha_{i,j}(t)}{(z-\tau_i)^{j+i}}$$

for some as–yet unspecified $\alpha_{i,j}(t)$. Contour integration as before gives

$$0 = -P(t) + \sum_{i=1}^{n}\sum_{j\in J_i} \alpha_{i,j}(t)\rho_{i,j} \tag{2}$$

if the z–degree of the numerator of $\phi(z;t)$ is small enough so that the denominator is $O(1/|z^2|)$ as z goes to ∞. In fact we will see that the z–degree of the numerator of ϕ is equal to m, and so equation (2) will be valid for polynomials $P(t)$ with t–degree less than or equal to $d-m$. Note that equation (2) is not quite in barycentric form, but is similar.

We know show how to construct $\phi(z)$ and find the $\alpha_{i,j}(t)$. Simply define

$$\phi(z;t) := \frac{\displaystyle\sum_{k=0}^{m} a_k(t) z^k}{\displaystyle (z-t)\prod_{i=1}^{n}(z-\tau_i)^{s_i}}$$

and we see that

$$\oint_C \phi(z;t)P(z)\mathrm{d}z = 0$$

for all $P(z)$ with $\deg_z P \leq d-m$, as claimed. It remains to be seen that we may find $a_k(t)$ such that the residue of ϕ at $z=t$ is -1 and the residues at $z=\tau_i$ of order $\#(J_i^c)+1$ are zero.

Proposition 2.1. The Birkhoff Interpolation Problem is poised if and only if the $m+1$ equations in the $m+1$ polynomial unknowns $a_k(t)$

$$\mathrm{residue}_{z=t}\phi(z;t) = -1$$

$$\left[(z-\tau_i)^{(-j-1)}\right]\mathrm{series}(\phi(z), z=\tau_i) = 0$$

($1 \leq i \leq n$, $j \in J_i^c$) have a solution.

We will supply a proof in the full version of this paper, but note that this idea goes back to 1967 (see [1]). Instead of giving a proof we show examples of how to use this proposition, and how to convert the results to a barycentric–like form.

Example 2.1.
Suppose we know $P(t)$ and $P'(t)$ at $t = 0$, and $P'(t)$ at $t = 1$. In tabular form

τ	P	P'
0	y_0	y_0'
1		y_1'

So $s_1 = s_2 = 2$, $J_1 = \{0, 1\}$, $J_1^c = \emptyset$, $J_2 = \{1\}$ and $J_2^c = \{0\}$. Computation gives

$$\phi(z;t) = \frac{a_0(t) + a_1(t)z}{(z-t)z^2(z-1)^2} = -\frac{1}{z-t} + \frac{1}{z} + \frac{\frac{1}{2}t(2-t)}{z^2} + \frac{\frac{1}{2}t^2}{(z-1)^2}$$

after symbolic expansion in partial fractions and solution of the two equations

$$\frac{a_0(t) + ta_1(t)}{t^2(t-1)^2} = -1$$

and

$$-3a_0(t) - 2a_1(t) + t(2a_0(t) + a_1(t)) = 0$$

which force the coefficient of $\frac{1}{z-t}$ to be -1 and the coefficient of $\frac{1}{z-1}$ to be zero, respectively. This gives

$$P(t) = 1 \cdot \rho_{1,0} + \frac{1}{2}t(2-t)\rho_{1,1} + \frac{1}{2}t^2\rho_{2,1} = y_0 + \frac{1}{2}t(2-t)y_0' + \frac{1}{2}t^2y_1'$$

which is the correct Birkhoff interpolant for the data, not in barycentric form.

Example 2.2.
The known information about $P(t)$ is the following

τ	P	P'
0	y_0	y_0'
$\frac{1}{2}$		$y_{\frac{1}{2}}'$
1	y_1	y_1'

So $J_1 = J_3 = \{0, 1\}$, $J_2 = \{1\}$, $J_2^c = \{0\}$, $m = 1$ and

$$\omega(t) = t^2\left(t - \frac{1}{2}\right)^2(t-1)^2 .$$

Setting up the system of equations for $a_0(t)$ and $a_1(t)$ is easy, but we find that they are inconsistent: thus the considered Borkhoff Interpolation Scheme is not poised (which is obvious in hindsight: specifying a degree 4 polynomial with, say, $y_0 = y_1 = 1$, $y_0' = 1$, and $y_1' = -1$ forces $y_{\frac{1}{2}}' = 0$).

Example 2.3.
The known information about $P(t)$ is the following

τ	P	P'	$P''/2$
0	y_0		$y_0''/2$
1		y_1'	

Hence $m = 3$, $J_1^c = \{1\}$ and $J_2^c = \{0, 2\}$ (but, in a more efficient way, we could take $s_2 = 2$ and $J_2^c = \{0\}$). Thus

$$\phi(z;t) = \frac{a_0 + a_1 z + a_2 z^2 + a_3 z^3}{(z-t)z^3(z-1)^3}$$

and the residue computations give

$$\begin{aligned} a_0(t) &= t^2(t-2) \\ a_1(t) &= -t(3t+1) \\ a_2(t) &= t(3t^2-3t-5) \\ a_3(t) &= -t(t^2-3) \end{aligned}$$

Note that $z - 1$ must divide the numerator of ϕ, because we could take $s_2 = 2$ instead.

3 Concluding Remarks

The Hermite barycentric form is

$$\begin{aligned} P(t) &= \omega(t) \sum_{i=1}^{n} \sum_{j=0}^{s_i-1} \frac{\gamma_{i,j}}{(t-\tau_i)^{j+i}} \sum_{k=0}^{j} \rho_{i,k}(t-\tau_i)^k = \\ &= \omega(t) \sum_{i=1}^{n} \sum_{j=0}^{s_i-1} \sum_{k=0}^{j} \gamma_{i,j}\rho_{i,k}(t-\tau_i)^{k-j-1} = \\ &= \omega(t) \sum_{i=1}^{n} \sum_{k=0}^{s_i-1} \sum_{j=k}^{s_i-1} \gamma_{i,j}\rho_{i,k}(t-\tau_i)^{k-j-1} = \\ &= \omega(t) \sum_{i=1}^{n} \sum_{k=0}^{s_i-1} \rho_{i,k} \left(\sum_{\ell=0}^{s_i-k-1} \gamma_{i,\ell+k}(t-\tau_i)^{-\ell-1} \right) \end{aligned}$$

whereas the Birkhoff form we have is

$$P(t) = \sum_{i=1}^{n} \sum_{j \in J_i} \alpha_{i,j}(t)\rho_{i,j} \ .$$

In the Hermite case, the $\gamma_{i,j}$ are found by solving a partial fraction problem; for the Birkhoff case, a further solution step is needed for $m + 1$ polynomials to get the polynomials coefficients $\alpha_{i,j}(t)$.

A companion matrix pencil for the Birkhoff case, to be used when dealing with root finding problems and like the one derived in [2] for the Hermite case, can be easily derived from this one. It is enough to use the Birkhoff interpolant to fill the missing data and use the companion matrix pencil for the Hermite case.

References

[1] J. C. Butcher: *A Multistep Generalization of Runge-Kutta Methods With Four or Five Stages.* J. ACM 14, 84–99, 1967.

[2] R. M. Corless, A. Shakoori, D.A. Aruliah, L. Gonzalez–Vega: *Barycentric Hermite Interpolants for Event Location in Initial-Value Problems.* Special issue on numerical computing in problem solving environments with emphasis on differential equations, Journal of Numerical Analysis, Industrial and Applied Mathematics (in print), 2008.

[3] L. Gonzalez–Vega,: *Applying quantifier elimination to the Birkhoff interpolation problem*. Jorunal of Symbolic Computation 22, 83–103, 1996.

[4] W. Werner and C. Schneider: *Hermite interpolation: The barycentric approach*. Computing 46, 35–51, 1990.

On the Representation of Constructible Sets

Changbo Chen, Liyun Li, Marc Moreno Maza, Wei Pan and Yuzhen Xie
{cchen, liyun, moreno, panwei, yxie}@orcca.on.ca

Abstract

The occurrence of redundant components is a natural phenomenon when computing with constructible sets. We present different algorithms for computing an irredundant representation of a constructible set or a family thereof. We provide a complexity analysis and report on an experimental comparison.

1 Introduction

Constructible sets appear naturally when solving systems of equations, in particular in presence of parameters. Our Maple implementation of comprehensive triangular decompositions has led us to dedicate a module of the `RegularChains` library, `ConstructibleSetTools` [4], to computing with constructible sets. In this paper, we discuss the representation used in our software and the implementation of fundamental operations, such as the set theoretical operations of difference, union and intersection. The problems faced there are representative of the usual dilemma of symbolic computation: choosing between canonical representation and lazy evaluation.

We represent a constructible set C by a list $[[T_1, h_1], \ldots, [T_e, h_e]]$ of so-called regular systems, where a regular system is a pair $[T, h]$ consisting of a regular chain T and a polynomial h regular w.r.t. the saturated ideal of T. Then the points of C are formed by the points that belong to at least one quasi-component $W(T_i)$ without canceling the associated polynomial h_i.

Example 1 *The constructible set C given by the conjunction of the conditions $s-(y+1)x = 0, s-(x+1)y = 0, s-1 \neq 0$ can be represented by two regular systems $R_1 = [T_1, h_1]$ and $R_2 = [T_2, h_2]$, where*

$$\begin{cases} T_1 &= [(y+1)x - s, y^2 + y - s] \\ h_1 &= s - 1 \end{cases}, \quad \begin{cases} T_2 &= [x+1, y+1, s] \\ h_2 &= 1 \end{cases}$$

and $x > y > s$; recall that $W(T_1)$ is defined by $(y+1)x - s = y^2 + y - s = 0$ and $y + 1 \neq 0$ whereas $W(T_2)$ is defined by $x + 1 = y + 1 = s = 0$.

In the representation of constructible sets, two levels of redundancy need to be considered. A first one appears in representing a single constructible set with regular systems. This problem arises when computing the complement of a constructible set, the union, the intersection or the difference of two constructible sets. For instance, one of the central operations is the union of two constructible sets C_1 and C_2. The lazy evaluation point of view suggests to represent $C_1 \cup C_2$ by concatenating the lists of regular systems representing C_1 and C_2. Of course, a `simplify` function is needed, at least in order to remove duplicated regular systems. A canonical representation could be achieved via a decomposition into irreducible components as in [11], but this could be very expensive for our usage of the union of two constructible sets. Alternatively, we remove the redundancy by making the zero sets of these regular systems pairwise disjoint (MPD). In the `ConstructibleSetTools` module of the `RegularChains` library in Maple 12, this operation is implemented in the function `MakePairwiseDisjoint`. The fundamental algorithm in support of MPD is the `Difference` of the zero sets of two regular systems $[T, h]$ and $[T', h']$, which will be described in Section 2.

A second level of redundancy can occur in a family of constructible sets. Our point of view is to provide an *intersection-free* representation of these constructible sets at an acceptable cost and to remove the redundancy as well. More precisely, let $\mathcal{C} = \{C_1, \ldots, C_m\}$ be a set of constructible sets, each of which is represented by a series of regular systems. For $\mathcal{C}$, redundancy occurs while some C_i intersects a C_j for $i \neq j$. Like the coprime factorization for integers, $\mathcal{C}$ can be refined to an intersection-free basis $\mathcal{D} = \{D_1, \ldots, D_n\}$, that is, $\mathcal{D}$ is a set of constructible sets such that

(1) $D_i \cap D_j = \emptyset$ for $1 \leq i \neq j \leq n$,

(2) each C_i can be uniquely written as a finite union of some of the D_j's.

This simplification operation is called *Symmetrically Make Pairwise Disjoint* (SMPD) and it has been implemented as `RefiningPartition` in the `ConstructibleSetTools` module. The input constructible sets of `RefiningPartition` are assumed to be represented by regular systems. For any other forms, one should use triangular decompositions [3] to obtain such a representation. The work in [8] suggests that multiple-valued logic minimization method could help with simplifying the problems before triangular decomposition.

Relying on the traditional Euclidean algorithm for computing GCDs [7] and the augment refinement method by Bach, Driscoll and Shallit in [1], this paper introduces efficient algorithms for MPD and SMPD by exploiting the triangular structure of the regular system representation. Then, we give a complexity analysis of our algorithms under some realistic assumptions. We also report an experimental comparison among the implementations of three algorithms for SMPD: the one following the approach of Bach et al. (BachSMPD), one using a divide-and-conquer approach (DCSMPD) and the one of [3] (OldSMPD) where we introduced this operation. All the tested examples are well known problems on parametric polynomial systems [3].

2 Background

The starting point of our work is an algorithm for computing the set theoretical difference of the zero sets of two regular systems $Z([T,h])$ and $Z([T',h'])$. We introduced this algorithm in [3] as a building block for simplifying the representations of constructible sets by means of MPD and SMPD, defined in the Introduction.

To be brief, we restrict ourselves to the case where $h = h' = 1$ and the initials of T and T' are all equal to 1, too. The complete algorithm has a similar structure but with more branches for case discussion. A sketch of this algorithm is given below and is illustrated by Figure 1.

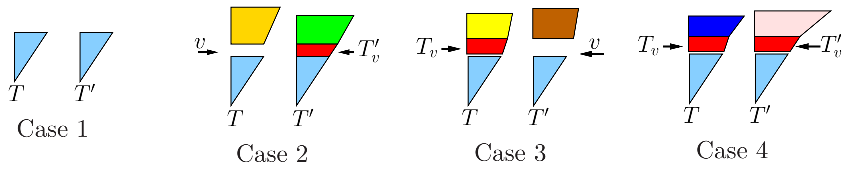


Figure 1: Compute $Z([T,h]) \setminus Z([T',h'])$ with $h = h' = 1$ by exploiting the triangular structure level by level.

Case 1: If T and T' generate the same ideal, which can be tested by pseudo-division, then the difference is empty.

If not, there exists a variable v such that below v the two sets generate the same ideal while at the level of v they disagree. This leads to the following case discussion.

Case 2: Assume that there is a polynomial in T' with main variable v and no such a polynomial in T. We then have two groups of points:

- those from $V(T)$ (the zero set of T) that do not cancel T'_v.
- those from $V(T)$ that cancel T'_v but which are outside of $V(T')$, which leads to a recursive call.

Case 3: Assume that there is a polynomial in T with main variable v and no such a polynomial in T'. Then, it suffices to exclude from $V(T)$ the points of $V(T')$ that cancel T_v, leading to a recursive call.

Case 4: Now we assume that both T_v and T'_v exist. By assumption, they are different modulo $T_{<v}$, which is the regular chain below the level v. Let g be their GCD modulo $T_{<v}$. To be simple, we assume that no splitting is needed and that the initial of g is 1. Three sub-cases arise:

Case 4.1: If g is a constant then the ideals generated by T and T' are relatively prime, hence $V(T)$ and $V(T')$ are disjoint and we just return $[T, 1]$.

Case 4.2: If g is non-constant but its main variable is less than v we have two groups of solution points:

- those from $V(T)$ that do not cancel g,
- those from $V(T)$ that cancel g but are still outside of $V(T')$, which leads to a recursive call.

Case 4.3: Finally, if g has main variable v, we just split T following the D5 principle philosophy [6] and we make two recursive calls.

From the above algorithm, we can see that the main cost comes from GCD computation modulo regular chains. By means of evaluation/interpolation techniques, these GCDs can be performed modulo zero-dimensional regular chains [9]. Thus if all the regular chains in the regular systems representing a constructible set have the same dimension and the same set of algebraic variables, one can reduce all computations to dimension zero; see Proposition 1.12 in [2] for a justification of this point. Therefore, in the present study, we restrict ourselves to regular systems $[T, h]$ such that the saturated ideal of T is zero-dimensional. We shall relax this restriction in future work. Under this zero-dimensional assumption, the saturated ideal of T is equal to the ideal generated by T; moreover h is invertible modulo T and thus can be assumed to be 1, or equivalently, can be ignored. Finally, we shall assume that the base field $\mathbb{K}$ is perfect and that $\langle T \rangle$ is radical. The latter assumption is easily achieved by squarefree factorization, since $\langle T \rangle$ is zero-dimensional.

Based on the above approach, we employ the augment refinement method in [1] in order to implement MPD for a list of regular systems or SMPD for a list of constructible sets. According to this method, given a set C_r and a list of pairwise disjoint sets $[C_1, \ldots, C_{r-1}]$, an intersection-free basis of $C_1, \ldots, C_{r-1}, C_r$ is computed by the following natural principle. First, we consider the pair of C_r and C_1; let $G_{C_rC_1}$ be their intersection. Second, we put $G_{C_rC_1}$ and $C_1 \setminus G_{C_rC_1}$ in the result list. Third, we need to consider the pair of $C_r \setminus G_{C_rC_1}$ and C_2 and continue similarly. In broad terms and in summary, only the "remaining part" of C_r from the first "pair refinement" needs to be considered for the rest of the "original refinement".

3 A complexity analysis

Our objective is to analyze the complexity of algorithms implementing MPD and SMPD operations. We rely on classical, thus quadratic, algorithms for computing GCDs modulo zero-dimensional regular chains [10]. Our motivation is practical: we aim at handling problem sizes to which the asymptotically fast GCDs of [5] are not likely to apply. Even if they would, we do not have yet implementations for these fast GCDs.

Let $T = [T_1, \ldots, T_n]$ be a zero-dimensional regular chain in $\mathbb{K}[X_1 < \cdots < X_n]$. We assume that T generates a radical ideal. The residue class ring $\mathbb{K}(T) := \mathbb{K}[X_1, \ldots, X_n]/\langle T \rangle$ is thus a direct product of fields (DPF). We denote by $\deg_i T$ the degree of T_i in X_i for $1 \leq i \leq n$. The degree of T is defined to be $\prod_{i=1}^{n} \deg_i T$.

We first adapt the extended Euclidean algorithm with coefficients in a field [7] to coefficients in a DPF defined by a regular chain. Then we use the augment refinement method of [1] to compute a polynomial GCD-free basis over a DPF. Following the inductive process applied in [5], we achieve the complexity result of Theorem 1. Recall that an arithmetic time $T \mapsto \mathsf{A}_n(\deg_1 T, \ldots, \deg_n T)$ is an asymptotic upper bound for the cost of basic polynomial arithmetic operations in $\mathbb{K}(T)$, counting operations in $\mathbb{K}$; see [5] for details.

Theorem 1 *There exists a constant C such that, writing*

$$\mathsf{A}_n(d_1, \ldots, d_n) = C^n (d_1 \times \cdots \times d_n)^2,$$

the function $T \mapsto \mathsf{A}_n(\deg_1 T, \ldots, \deg_n T)$ is an arithmetic time for regular chains T in n variables, for all n.

Therefore, an extended GCD of f_1 and f_2 in $\mathbb{K}(T)[y]$ with degrees $d_1 \geq d_2$, can be computed in $O(d_1 d_2)\mathsf{A}_n(T)$ operations in $\mathbb{K}$. Moreover, for a family of monic squarefree polynomials $F = \{f_1, \ldots, f_m\}$ in $\mathbb{K}(T)[y]$ with degrees $d_1, \ldots, d_m$, we extend the augment refinement method in [1] to compute a GCD-free basis of F modulo T in $O(\sum_{1 \leq i < j \leq m} (d_i d_j))\mathsf{A}_n(T)$ operations in $\mathbb{K}$.

To estimate the time complexity of MPD and SMPD in zero-dimensional case where a regular system can be regarded as a regular chain, we start from the base operation RCPairRefine, presented in Algorithm 1. Given two zero-dimensional, monic and squarefree regular chains T and T' in $\mathbb{K}[X_1, \ldots, X_n]$, RCPairRefine produces three constructible sets D, I and D' such that $Z(D) = V(T) \setminus V(T')$, $Z(I) = V(T) \cap V(T')$ and $Z(D') = V(T') \setminus V(T)$. In other words, $\{D, I, D'\}$ is an intersection-free basis of the zero sets defined by T and T'. In the worst case, for each v in $X_1, \ldots, X_n$, a GCD of T_v and T'_v modulo $T_{<v}$ and two divisions are performed. The GCD operation used in Algorithm 1 is specified in [10]. If the degrees of regular chains T and T' are d and d' respectively, then RCPairRefine costs $O(C^{n-1}dd')$ operations in $\mathbb{K}$.

Algorithm 1 RCPairRefine

Input: two monic squarefree zero-dimensional regular chains T and T'
Output: three constructible sets D, I and D', such that
$V(T) \setminus V(T') = Z(D), V(T) \cap V(T') = Z(I)$ and $V(T') \setminus V(T) = Z(D')$

```
1: if $T = T'$ then
2:   return $\emptyset$, $[T]$, $\emptyset$
3: else
4:   $D \leftarrow \emptyset$; $I \leftarrow \emptyset$; $D' \leftarrow \emptyset$
5:   Let $v$ be the largest variable s.t. $T_{<v} = T'_{<v}$
6:   for $(g, G) \in \mathsf{GCD}(T_v, T'_v, T_{<v})$ do
7:     if $g \in \mathbb{K}$ or $\mathrm{mvar}(g) < v$ then
8:       $T_q \leftarrow G \cup \{T_v\} \cup T_{>v}$; $T'_q \leftarrow G \cup \{T'_v\} \cup T'_{>v}$; $D \leftarrow D \cup T_q$; $D' \leftarrow D' \cup T'_q$
9:     else
10:      $q \leftarrow \mathrm{pquo}(T_v, g, G)$; $q' \leftarrow \mathrm{pquo}(T'_v, g, G)$; $E \leftarrow G \cup \{g\} \cup T_{>v}$; $E' \leftarrow G \cup \{g\} \cup T'_{>v}$
11:      if $\mathrm{mvar}(q) = v$ then
12:        $T_q \leftarrow G \cup \{q\} \cup T_{>v}$; $D \leftarrow D \cup T_q$
13:      end if
14:      if $\mathrm{mvar}(q') = v$ then
15:        $T'_q \leftarrow G \cup \{q'\} \cup T'_{>v}$; $D' \leftarrow D' \cup T'_q$
16:      end if
17:      $W, J, W' \leftarrow$ RCPairRefine$(E, E')$; $D \leftarrow D \cup W$; $I \leftarrow I \cup J$; $D' \leftarrow D' \cup W'$
18:    end if
19:  end for
20:  return $D, I, D'$
21: end if
```

Note that Algorithm 1 suffices to compute solely `Difference`(T, T'), i.e. $V(T) \setminus V(T')$ when removing the lines for computing I and D'. Thus, the cost of `Difference`(T, T') is also bounded by $O(C^{n-1}dd')$.

Using RCPairRefine and adapting the augment refinement method in [1] to a list of regular systems or constructible sets, the operation CSPairRefine for computing an intersection-free basis of a pair of constructible sets can be deduced naturally. Based on these operations, we build Algorithms 2 and 3 to implement MPD and SMPD respectively. Their complexity results are stated in the theorems below.

Theorem 2 *Let $L = \{U_1, \ldots, U_m\}$ be a set of monic and square free regular chains in dimension zero and the degree of U_i be d_i for $1 \le i \le m$. Then a pairwise disjoint representation of L (that is, regular chains $S_1, \ldots, S_q$ such that $\{V(S_1), \ldots, V(S_q)\}$ forms a partition of the union of $V(U_1), \ldots, V(U_m)$) can be computed in $O(C^{n-1} \sum_{1 \le i < j \le m} d_i d_j)$ operations in $\mathbb{K}$.*

Theorem 3 *Given a set $L = \{C_1, \ldots, C_m\}$ of constructible sets, each of which is given by some monic squarefree and pairwise disjoint regular chains in dimension zero. Let D_i be the number of points in C_i for $1 \le i \le m$. An intersection-free basis of L can be computed in $O(C^{n-1} \sum_{1 \le i < j \le m} D_i D_j)$ operations in $\mathbb{K}$.*

Algorithm 2 MPD

Input: a list L of monic squarefree zero-dimensional regular chains
Output: a pairwise disjoint representation of L

```
1: $n \leftarrow |L|$
2: if $n < 2$ then
3:    return $L$
4: else
5:    $d \leftarrow L[n]$
6:    $L^* \leftarrow$MPD$(L[1, \ldots, n-1])$
7:    for $l' \in L^*$ do
8:       $d \leftarrow \mathsf{Difference}(d, l')$
9:    end for
10:   return $d \cup L^*$
11: end if
```

Algorithm 3 BachSMPD

Input: a list L of constructible sets with each consisting of a family of monic squarefree zero-dimensional regular chains
Output: an intersection-free basis of L

```
1: $n \leftarrow |L|$
2: if $n < 2$ then
3:    return $L$
4: else
5:    $I \leftarrow \emptyset$; $D' \leftarrow \emptyset$; $d \leftarrow L[n]$
6:    $L^* \leftarrow \mathsf{BachSMPD}(L[1, \ldots, n-1])$
7:    for $l' \in L^*$ do
8:       $d, i, d' \leftarrow \mathsf{CSPairRefine}(d, l')$
9:       $I \leftarrow I \cup i$; $D' \leftarrow D' \cup d'$
10:   end for
11:   return $d \cup I \cup D'$
12: end if
```

We have also combined a divide-and-conquer approach with the augment refinement method, leading to another algorithm, called DCSMPD, for the operation SMPD. Our analysis shows that its worst case complexity is the same as that of BachSMPD; however it performs better for some tested examples.

The main operation in our divide-and-conquer algorithm merges two lists of pairwise disjoint constructible sets $[A_1, \ldots, A_s]$ and $[B_1, \ldots, B_t]$. We first consider A_1 and $[B_1, \ldots, B_t]$ following the principle of the augment refinement method described in Section 2. This will result in three parts: $[G_{A_1B_1}, \ldots, G_{A_1B_t}]$, $[B_1 \backslash G_{A_1B_1}, \ldots, B_t \backslash G_{A_1B_t}]$ and $A_1 \backslash G_{A_1B_1} \backslash \cdots \backslash G_{A_1B_t}$, where $G_{A_1B_1}, \ldots, G_{A_1B_t}$ are the respective intersections of A_1 and B_i for $1 \leq i \leq t$. Next we only need to consider A_2 with respect to $[B_1 \backslash G_{A_1B_1}, \ldots, B_t \backslash G_{A_1B_t}]$ since A_2 is disjoint from $G_{A_1B_i}$ for $1 \leq i \leq t$. The same rule applies to each of $A_3, \ldots, A_s$.

4 An experimental comparison

In this section we provide benchmarks on the implementation of three different algorithms for realizing the SMPD operation, respectively OldSMPD, BachSMPD and DCSMPD.

Given a list $\mathcal{C}$ of constructible sets, the algorithm OldSMPD first collects all their defining regular systems into a list, then computes its intersection-free basis $\mathcal{G}$ which consists of regular systems, and finally one can easily group $\mathcal{G}$ into an intersection-free basis of $\mathcal{C}$. In this manner the defining regular systems of each constructible set are made (symmetrically) pairwise disjoint, though sometimes this is unnecessary. As reported in [3], OldSMPD is expensive and sometimes can be a bottleneck. After replacing OldSMPD in the comprehensive triangular decomposition (CTD) algorithm respectively by BachSMPD and DCSMPD, we rerun the CTD algorithm for twenty examples selected from [3] (all examples are in positive dimension).

The second column named OldSMPD in Table 1 is the timing from [3] where SMPD was first implemented. The third column OldSMPD (improved) extends the RCPairRefine algorithm to positive dimension and manages to compute the difference and the intersection in one pass, whereas in OldSMPD the set theoretical differences and the intersections are computed separately. The fourth column and the sixth column present the timings of computing an SMPD with BachSMPD and DCSMPD respectively. The fifth column and the seventh column show the timings for cleaning each constructible set with an MPD operation on each of the constructible set in the output. In this way, we remove the redundancy both among a series of constructible sets and in their defining regular systems.

Table 1 shows that BachSMPD and DCSMPD are more efficient than OldSMPD. We know from the previous section that Algorithms BachSMPD and DCSMPD have the same complexity in the worst case. However, experimentation shows that DCSMPD performs more than 3 times faster than BachSMPD for some examples, which need to be investigated in the future.

Sys	OldSMPD	OldSMPD (improved)	BachSMPD	BachSMPD (+MPD)	DCSMPD	DCSMPD (+MPD)
9	3.817	0.871	0.818	0.877	1.112	1.435
10	1.138	0.154	0.223	0.223	0.281	0.344
11	12.302	3.949	3.494	3.766	0.786	0.914
12	10.114	0.551	0.383	0.383	0.318	0.318
13	1.268	0.348	0.318	0.318	0.362	0.363
14	0.303	0.118	0.103	0.103	0.062	0.062
15	1.123	0.271	0.259	0.259	0.271	0.271
16	2.407	1.442	1.184	1.449	0.703	0.927
17	0.574	0.116	0.091	0.100	0.159	0.173
18	0.548	0.257	0.293	0.300	0.283	0.290
19	0.733	0.460	0.444	0.444	0.211	0.211
20	0.020	0.013	0.013	0.013	0.013	0.013
21	3.430	0.607	0.584	0.584	0.633	0.633
22	25.413	9.291	8.292	8.347	9.530	9.592
23	1097.291	95.690	82.468	82.795	122.575	125.286
24	11.828	0.912	0.930	0.930	0.985	1.784
25	54.197	12.330	1.934	1.934	1.778	2.900
26	0.530	0.065	0.047	0.047	0.064	0.065
27	27.180	16.792	13.705	16.280	4.626	6.323
28	–	2272.550	1838.927	1876.061	592.554	624.679

Table 1 Timing(s) of 20 examples computed by 3 algorithms

Acknowledgement
All authors acknowledge the continuing support of Waterloo Maple Inc., the Mathematics of Information Technology and Complex Systems (MITACS) and the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] E. Bach, J. R. Driscoll, and J. Shallit. Factor refinement. In *SODA*, pages 201–211, 1990.

[2] F. Boulier, F. Lemaire, and M. Moreno Maza. Well known theorems on triangular systems and the D5 principle. In *Proc. of* Transgressive Computing, Spain, 2006.

[3] C. Chen, O. Golubitsky, F. Lemaire, M. Moreno Maza, and W. Pan. *Comprehensive Triangular Decomposition*, volume 4770 of *Lecture Notes in Computer Science*, pages 73–101. Springer Verlag, 2007.

[4] C. Chen, F. Lemaire, L. Li, M. Moreno Maza, W. Pan and Y. Xie. The `ConstructibleSetTools` and `ParametricSystemTools` modules of the `RegularChains` library in MAPLE. In *Proc. CASA'08.*

[5] X. Dahan, M. Moreno Maza, É. Schost, and Y. Xie. On the complexity of the D5 principle. In *Proc. of* Transgressive Computing, Spain, 2006.

[6] J. Della Dora, C. Dicrescenzo, and D. Duval. About a new method for computing in algebraic number fields. In *Proc. EUROCAL 85 Vol. 2*, volume 204 of *LNCS*, pages 289–290. Springer-Verlag, 1985.

[7] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra.* Cambridge University Press, 1999.

[8] H. Hong. Simple solution formula construction in cylindrical algebraic decomposition based quantifier elimination. In *Proc. ISSAC'92*, pages 177–188.

[9] X. Li, M. Moreno Maza, and R. Rasheed. Fast arithmetic and modular techniques for polynomial gcds modulo regular chains, 2008. Preprint.

[10] R. Rioboo and M. Moreno Maza. Polynomial GCD computations over towers of algebraic extensions. In *Proc. AAECC-11*, pages 365–382, 1995.

[11] M. Manubens and A. Montes. Minimal canonical comprehensive Gröbner system, 01-12-06. arXiv:math.AC/0611948.

A Survey of Recent Advancements of Multivariate Hensel Construction and Applications

Tateaki Sasaki

Institute of Mathematics, University of Tsukuba
Tsukuba-shi, Ibaraki 305-8571, Japan
sasaki@math.tsukuba.ac.jp

The generalized Hensel construction is a very important tool in computer algebra. It plays essential roles in multivariate GCD computation, multivariate factorization, and so on; see [4] for details. In early 1970's, important advancements in computational technique were made: formulation by interpolation polynomials by Moses and Yun [13], and parallel Hensel construction by Yun [25] and Wang [24]. Twenty years later, two important advancements have been made: in 1993, Sasaki and Kako [20] formulated the multivariate Hensel construction at singular points, which they called "extended Hensel construction (EHC)"; in 2008, Sasaki and Inaba [19] expressed the multivariate Hensel factors in the roots of initial factors. These advancements lead us to wide applications, theoretically as well as practically. In this article, we describe these advancements and survey representative applications attained so far or being attained now.

A. Extended Hensel construction and its applications

Let $F(x,\boldsymbol{u})$ be a given multivariate irreducible polynomial in $\mathbf{C}[x,u_1,\dots,u_\ell]$, where $(\boldsymbol{u})=(u_1,\dots,u_\ell)$ and we consider the case of $\ell \geq 2$ mostly. Let $R(\boldsymbol{u}) = \mathrm{resultant}_x(F, \partial F/\partial x)$. The point $(\boldsymbol{s}) \in \mathbf{C}^\ell$ is called a *singular point for Hensel construction* or *singular point* in short if $R(\boldsymbol{s})=0$. Let $F(x,\boldsymbol{s}) = \hat{F}^{(0)}(x)\check{F}^{(0)}(x)$, where $\hat{F}^{(0)}(x) = (x-\alpha)^m$, with $m \geq 2$ and $\check{F}^{(0)}(\alpha) \neq 0$. Therefore, $(\boldsymbol{s})$ is a singular point for Hensel construction of $F(x,\boldsymbol{u})$. Using the generalized Hensel construction, we can factor $F(x,\boldsymbol{u})$ as $F(x,\boldsymbol{u}) \equiv \hat{F}^{(k)}(x,\boldsymbol{u})\check{F}^{(k)}(x,\boldsymbol{u}) \pmod{(\boldsymbol{u}-\boldsymbol{s})^{k+1}}$, where $\hat{F}^{(k)}(x,\boldsymbol{s}) = (x-\alpha)^m$. The generalized Hensel construction breaks down for $\hat{F}^{(k)}(x,\boldsymbol{u})$ and EHC is the Hensel construction for such polynomials as $\hat{F}^{(k)}(x,\boldsymbol{u})$. The EHC for bivariate polynomials was conceived by Kuo [10] in customizing a method of Abhyankar and Moh [3] for analytic factorization, and Sasaki and Kako [20] formulated EHC for multivariate polynomials independently in 1993.

A key concept in the EHC is *Newton polynomial* defined as follows for monic polynomials; see [17] for non-monic case. Below, we put $n = \deg_x(F(x,\boldsymbol{u}))$.

Definition 1 (Newton line and Newton polynomial) *For each nonzero monomial $c\,x^{e_x}u_1^{e_1}\cdots u_\ell^{e_\ell}$ of $F(x,\boldsymbol{u})$, plot a dot at the point (e_x,e_t), where $e_t = e_1+\cdots+e_\ell$, in (e_x,e_t)-plane. Let $\mathcal{L}$ be a straight line such that it passes through the point $(n,0)$ as well as another dot plotted and that no dot is plotted below $\mathcal{L}$. The line $\mathcal{L}$ is called* Newton line *for $F(x,\boldsymbol{u})$. The sum of all the monomials plotted on $\mathcal{L}$ is called* Newton polynomial.

In EHC, factors of the Newton polynomial are chosen to be the initial factors of the Hensel construction. The modulus J_k ($k \in \mathbf{N}$) is determined as follows. Let $(0,e)$ be the intersecting point between $\mathcal{L}$ and e_t-axis in the (e_x,e_t)-plane, and let $\hat{n}$ and $\hat{e}$ be relatively prime positive integers satisfying $\hat{e}/\hat{n} = e/n$. Let $\mathcal{L}_k$ be the line obtained by shifting $\mathcal{L}$ upwardly by $k/\hat{n}$. Then, define J_k so that it contains all the monomials on and above $\mathcal{L}_k$ but no monomial below $\mathcal{L}_k$. For details of EHC, see [20] and [17]. In this article, we assume that the origin $(\boldsymbol{u}) = (0,\dots,0)$ is a singular point, and we consider EHC at the origin.

It is well known that if the expansion point is not singular then we can compute the Taylor-series roots of $F(x,\boldsymbol{u})$ by the parallel Hensel construction. Similarly, if the expansion point is singular, a repeated application of EHC allows us to factor $F(x, \boldsymbol{u})$ as

$$F(x, \boldsymbol{u}) \;\equiv\; f_n(\boldsymbol{u})\,(x - \phi_1^{(k)}(\boldsymbol{u})) \cdots (x - \phi_n^{(k)}(\boldsymbol{u})) \quad (\mathrm{mod}\ J_{k+1}). \tag{1}$$

We call $\phi_1^{(\infty)}(\boldsymbol{u}), \ldots, \phi_n^{(\infty)}(\boldsymbol{u})$ *Hensel-series roots* because they are computed by the Hensel construction. In the case of bivariate polynomial $F(x, u_1)$, the Hensel-series roots are Puiseux series in general. Compared with classical Newton-Puiseux's method, the EHC method computes the roots parallelly. Furthermore, if the coefficients of $F(x, u_1)$ are floating-point numbers, Newton-Puiseux's method often becomes very unstable, but the EHC method is quite stable; the reason will be explained in **A-2** below.

In the multivariate case ($\ell \geq 2$), the Hensel factors are very characteristic expressions: homogeneous rational functions in $u_1, \ldots, u_\ell$ appear in the coefficients of Hensel factors.

Example 1 Extended Hensel construction and Hensel-series roots. Let

$$\begin{aligned} F(x, u, v) \;\; = \;\; & (u^2 - v^2)\,x^3 - (u^3 + 3u^2v - uv^2 - v^3)\,x^2 \\ & + (2u^3v + 3u^2v^2)\,x \\ & - (u^3v^2 + u^2v^3 - u^6 - v^6). \end{aligned}$$

The Newton polynomial for $F(x, u, v)$ is $F(x, u, v) - (u^6 + v^6)$. The Newton polynomial is factored as $(x - u - v) \cdot [(u+v)x - uv] \cdot [(u-v)x - uv]$. The EHC with initial factors $(x - u - v)$, $[(u+v)x - uv]$ and $[(u-v)x - uv]$ gives the Hensel construction $F(x, u, v) = (x - \chi_1^{(\infty)}) \cdot [(u+v)(x - \chi_2^{(\infty)})] \cdot [(u-v)(x - \chi_3^{(\infty)})]$, where

$$\begin{aligned} \chi_1^{(\infty)}(u, v) \;&=\; (u + v) - \frac{u^6 + v^6}{(u^2 + uv + v^2)(u^2 - uv - v^2)} + \cdots, \\ \chi_2^{(\infty)}(u, v) \;&=\; \frac{1}{u + v}\Big[uv - \frac{(u + v)(u^6 + v^6)}{2uv^2\,(u^2 + uv + v^2)} + \cdots\Big], \\ \chi_3^{(\infty)}(u, v) \;&=\; \frac{1}{u - v}\Big[uv + \frac{(u - v)(u^6 + v^6)}{2uv^2\,(u^2 - uv - v^2)} + \cdots\Big]. \end{aligned}$$

Resultants of factors of Newton polynomial appear in the denominators. $\diamondsuit$

One may think that the appearance of rational functions in the Hensel series makes the situation complicated, but the denominators of rational functions have a very important meaning; near the expansion point, Hensel-series roots intersect each other at the zero-points of denominators [18]. We can expand the multivariate algebraic function into multivariate Puiseux series (fractional-power series in each variable), see [12]. However, the multivariate Puiseux series is quite messy to compute and the resulting series is not suited for seeing the behavior of the roots. Hence, Hensel series obtained by EHC will be much more useful than multivariate Puiseux series.

The EHC has been applied successfully to the following issues so far: 1) analytic continuation of algebraic functions via singular points, 2) error analysis of symbolic Newton's method for computing Taylor-series roots of multivariate polynomials, 3) solving the *nonzero substitution problem* in multivariate polynomial factorization, 4) constructing algorithms of multivariate analytic factorization, and so on.

A-1. In analytic continuation, we usually continue an analytic function along a path surrounding one or several singular points, which is considerably time-consuming. Shiihara and Sasaki [23] proposed to perform the analytic continuation of algebraic functions via singular points, then the computation is speeded up largely. This method requires power-series roots expanded at singular points, and we can use Hensel series, multivariate as well as univariate. Recently, similar approach for univariate algebraic functions has been done by Poteaux [14].

A-2. Given a multivariate polynomial with floating-point number coefficients, we consider computing series roots of the polynomial. When the expansion point is a singular point, Newton-Puiseux's method often causes fully erroneous terms. Thus, if we do not eliminate the fully erroneous terms, the computation will lead us to a completely wrong result. With EHC, the fully erroneous terms are systematically discarded by the change of modulus, hence EHC is quite stable. When the expansion point is close to a singular point, Sasaki, Kitamoto and Kako [21] clarified that Newton's method may cause large numerical errors; let d be the distance between the expansion point and the nearest singular point, hence $d \ll 1$, then the errors in the k-th order expansion may be as large as $O((1/d)^k)\,\varepsilon_{\rm m}$, where $\varepsilon_{\rm m}$ is the machine epsilon. In [21], authors proposed a method to compute the series roots accurately: first, compute Hensel-series roots at the nearest singular point, then continue them to the expansion point. See also [22] for error analysis of the generalized Hensel construction with floating-point numbers.

A-3. In the multivariate factorization with Hensel construction, we usually choose the origin as the expansion point. However, if the polynomial is not square-free at the origin or the leading coefficient vanishes at the origin, we must shift the origin. Many of the actual polynomials to be factored are sparse and contain high degree terms, then the origin shifting increases the number of terms drastically, making the computation very time-consuming. This is the nonzero substitution problem, and it has been unsolved for many years. With EHC, we need not shift the origin, so the problem was solved completely. The actual algorithm is as follows; see [5] for details. First, we factor $F(x,\boldsymbol{u})$ modulo J_{k+1}, with polynomial initial factors, where $k > \mathrm{tdeg}_{\boldsymbol{u}}(F)$, obtaining Hensel factors in $\mathbf{C}(\boldsymbol{u})[x]$. Then, we combine Hensel factors so that not only denominators in the coefficients but also higher order terms of Hensel factors are eliminated. Combining Hensel factors can be done efficiently by the zero-sum algorithm [15]. See [5] for how effective EHC is for the nonzero substitution problem.

A-4. Analytic factorization is the factorization in $\mathbf{C}\{\boldsymbol{u}\}[x]$, of a given polynomial in $\mathbf{C}[x,\boldsymbol{u}]$, where $\mathbf{C}\{\boldsymbol{u}\}$ denotes the power series ring over $\mathbf{C}$; see [2]. If the expansion point is non-singular then result of the generalized Hensel construction is nothing but the analytic factorization. Hence, in analytic factorization, we usually assume that the expansion point is a singular point. For bivariate polynomials, Abhyankar proposed an algorithm named "expansion base method" [1], see also [10] and [11]. There are only a few works for multivariate polynomials. In 2003, Iwami [7, 8] proposed an algorithm based on EHC. Her algorithm is as follows: first, perform the EHC and obtain Hensel factors which are in $\mathbf{C}(\boldsymbol{u})[x]$, then eliminate the denominators successively from the lowest order to some high order by combining the extended Hensel factors.

B. Hensel construction in roots, and ongoing applications

The resultant can be expressed in the roots of given polynomials, and the expression obtained manifests the meaning of resultant clearly. Similarly, if the Hensel factors can be expressed in the roots of initial factors, the resulting expression will be quite useful. Recently, the present author and Inaba have formulated the multivariate Hensel construction, both the generalized and extended ones, so that the Hensel factors are expressed in the roots of initial factors [19]. Furthermore, we have formulated the construction so that the higher order terms of the given polynomial $F(x,\boldsymbol{u})$ are treated as a mass: we introduce an auxiliary variable t and split $F(x,\boldsymbol{u})$ into two parts as

$$\tilde{F}(x,\boldsymbol{u},t) \ \stackrel{\rm def}{=}\ F_0(x) + t\,F_{\rm u}(x,\boldsymbol{u}), \qquad \begin{cases} F_0(x) = F(x,\boldsymbol{0}), \\ F_{\rm u}(x,\boldsymbol{u}) = F(x,\boldsymbol{u}) - F_0(x), \end{cases} \tag{2}$$

then we perform the Hensel construction with respect to moduli t^k $(k = 1,2,3,\ldots)$.

Example 2 Generalized Hensel construction in roots. Let $F(x,u,v)$, $F_0(x)$, $G_0(x)$ and $H_0(x)$ be as follows; γ_1, γ_2 and η_1, η_2 are the roots of $G_0(x)$ and $H_0(x)$, respectively.

$$\begin{array}{rcl} F(x,u,v) &=& (x-2)(x-1)(x)(x+2) + F_{\rm u}(x,u,v), \\ && \text{where}\ \ F_{\rm u}(x,u,v) = ux^2 + v^2x + u^2v^2, \\ F_0(x) &=& (x-2)(x-1)(x)(x+2), \\ G_0(x) &=& (x-2)(x-1), \qquad \gamma_1 = 2, \ \ \gamma_2 = 1, \\ H_0(x) &=& \quad (x)(x+2), \qquad\quad \eta_1 = 0, \ \ \eta_2 = -2. \end{array}$$

Note that $F_0(x)$ is square-free. Let vectors $\vec{G} = (G_1, G_2)$ and $\vec{H} = (H_1, H_2)$ be defined to be $G_i = F_{\mathrm{u}}(\gamma_i, u, v)/F_0(\gamma_i)$ and $H_i = F_{\mathrm{u}}(\eta_i, u, v)/F_0(\eta_i)$ $(i = 1, 2)$:

$$\vec{G} = \Big(\frac{u^2v^2 + 2v^2 + 4u}{8},\ \frac{u^2v^2 + v^2 + u}{-3}\Big), \qquad \vec{H} = \Big(\frac{u^2v^2}{4},\ \frac{u^2v^2 - 2v^2 + 4u}{-24}\Big).$$

Our method gives the Hensel construction up to t^2, for example, as follows.

$$\begin{aligned}
\tilde{F}(x,u,v,t) \ \equiv\ & \Big\{G_0(x) + \frac{G_0(x)}{x-2}\Big[\, t\,G_1 - t^2\Big(\frac{G_1H_1}{2} + \frac{G_1H_2}{4}\Big)\Big] \\
& \quad + \frac{G_0(x)}{x-1}\Big[\, t\,G_2 - t^2\Big(\frac{G_2H_1}{1} + \frac{G_2H_2}{3}\Big)\Big]\Big\} \\
\times\ & \Big\{H_0(x) + \frac{H_0(x)}{x}\Big[\, t\,H_1 + t^2\Big(\frac{G_1H_1}{2} + \frac{G_2H_1}{1}\Big)\Big] \\
& \quad + \frac{H_0(x)}{x+2}\Big[\, t\,H_2 + t^2\Big(\frac{G_1H_2}{4} + \frac{G_2H_2}{3}\Big)\Big]\Big\} \quad (\mathrm{mod}\ t^3).
\end{aligned}$$

Here, denominators $2, 4$, etc. are systematically determined by the root-differences $\gamma_i - \eta_j$ $(1 \le i, j \le 2)$. As this expression shows, the Hensel factors up to any order can be expressed in terms of G_1, G_2, H_1, H_2 and root-differences. ♢

We have so far applied the new formulation to the following topics: 1) finding a sufficient condition on Iwami's algorithm of analytic factorization, 2) deriving a formula of convergence domain of multivariate Taylor-series and Hensel-series roots. Currently, we are challenging to the monodromy computation.

B-1. Iwami's algorithm mentioned in **A-4** above is incomplete in that the stopping condition on the denominator elimination is not given. Determination of the stopping condition seems to be quite hard from the conventional approach. Recently, we have succeeded in determining one sufficient condition by using the extended Hensel factors expressed in roots [19].

B-2. The convergence domain of Taylor expansion of univariate algebraic function is well-known: the domain is a circle whose center is at the expansion point and whose radius is the distance from the expansion point to the nearest singular point. In the case of multivariate algebraic functions, no book describes the convergence domain so the author thinks that there is no general formula on the convergence domain. One exception is the case of $\deg_x(F) = 2$ (hence, we have two conjugate roots). In this case, the convergence domain can be determined easily from the root formula, which shows that the domain is never a circle but a complicated domain in general.

In [6], we investigated properties of multivariate Hensel series numerically, which reveals that the Hensel series has a very characteristic convergence domain. We surely have a convergence domain of multivariate Hensel series, however, the expansion point is contained in neither convergence nor divergence domains. The convergence and divergence domains coexist in any small neighborhood of the expansion point, in such a way that a short line segment whose one edge is on the expansion point is in the convergence domain in one direction but in the divergence domain in another direction. Using a technique developed in [19], we are now succeeding in deriving a formula which describes the convergence domain of both Taylor and Hensel expansions of multivariate algebraic function.

B-3. In [6], we investigated many-valuedness of Hensel series and found a very interesting and astonishing behavior. Since $F(x, \boldsymbol{u})$ is irreducible over $\mathbf{C}$, all the n roots of $F(x, \boldsymbol{u})$ are conjugate one another. However, conjugateness of Hensel series are determined by irreducible factors of the Newton polynomial $F_0(x, \boldsymbol{u})$: only Hensel series generated from one irreducible factor of $F_0(x, \boldsymbol{u})$ are conjugate each other. Tracing a Hensel-series root numerically along a path which starts from a point in the convergence domain, passes through the divergence domain, and arrive at another point in the convergence domain, we found that the series root goes to infinity in the divergence domain and "jumps" to another Hensel-series root in the convergence domain. However, n Hensel series as a whole reproduce well the n algebraic functions numerically in the convergence domain. We are now challenging to clarify this behavior theoretically.

References

[1] S.S. Abhyankar. Irreducibility criterion for germs of analytic functions of two complex variables. *Adv. in Math.* **74** (1989), 190-267.

[2] S.S. Abhyankar. *Algebraic Geometry for Scientists and Engineers.* Number 35 in Mathematical Surveys and Monographs, American Mathematical Society, 1990.

[3] S.S. Abhyankar and T.M. Moh. Newton-Puiseux expansion and generalized Tschirnhausen transformation II. J. Reine Angew. Math., **261** (1973), 29-54.

[4] K.O. Geddes, S.R. Czapor and G. Labahn. *Algorithms for Computer Algebra.* Kluwer Academic Publishers, 1992.

[5] D. Inaba. Factorization of multivariate polynomials by extended Hensel construction. *ACM SIGSAM Bulletin* **39** (2005), 142-154.

[6] D. Inaba and T. Sasaki. A numerical study of extended Hensel series. *Proc. SNC'2007 (Symbolic-Numeric Computation)*, J. Verchelde and S. Watt (Eds.), ACM, **ISBN:** 978-1-59593-744-5, 103-109, 2007.

[7] M. Iwami. Analytic factorization of the multivariate polynomial. *Proc. CASC 2003 (Computer Algebra in Scientific Computing)*, V.G. Ganzha, E.W. Mayr and E.V. Vorozhtsov (Eds.), Technishe Universität München Press, 213-225, 2003.

[8] M. Iwami. Extension of expansion base algorithm to multivariate analytic factorization. *Proc. CASC 2004 (Computer Algebra in Scientific Computing)*, V.G. Ganzha, E.W. Mayr and E.V. Vorozhtsov (Eds.), Technishe Universität München Press, 269-282, 2004.

[9] M. Iwami. A unified algorithm for multivariate analytic factorization. *Proc. CASC 2007 (Computer Algebra in Scientific Computing)*, V.G. Ganzha, E.W. Mayr and E.V. Vorozhtsov (Eds.); Lect. Notes Comp. Sci., **4770**, 211-223, 2007.

[10] T.-C. Kuo. Generalized Newton-Puiseux theory and Hensel's lemma in $\mathbf{C}[[x, y]]$. *Canad. J. Math.* **XLI** (1989), 1101-1116.

[11] S. McCallum. On testing a bivariate polynomial for analytic reducibility. *J. Symb. Comput.* **24** (1997), 509-535.

[12] J. McDonald. Fiber polytopes and fractional power series. *J. Pure Appl. Algebra* **104** (1995), 213-233.

[13] J. Moses and D.Y.Y. Yun. The EZGCD algorithm. *Proc. ACM Annual Conference*, Atlanta, 159-166, 1973.

[14] A. Poteaux. Computing monodromy groups defined by plane algebraic curves. *Proc. SNC'2007 (Symbolic-Numeric Computation)*, J. Verchelde and S. Watt (Eds.), ACM, **ISBN:** 978-1-59593-744-5, 36-45, 2007.

[15] T. Sasaki. Approximate multivariate polynomial factorization based on zero-sum relations. *Proc. ISSAC 2001 (Intern'l Symp. on Symbolic and Algebraic Computation)*, B. Mourrain (Ed.), ACM, **ISBN:** 1-58113-417-7, 284-291, 2001.

[16] T. Sasaki. Approximately singular multivariate polynomials. *Proc. CASC 2004 (Computer Algebra in Scientific Computing)*, V.G. Ganzha, E.W. Mayr and E.V. Vorozhtsov (Eds.), Technishe Universität München Press, 399-408, 2004.

[17] T. Sasaki and D. Inaba. Hensel construction of $F(x, u_1, \ldots, u_\ell)$, $\ell \geq 2$, at a singular point and its applications. *ACM SIGSAM Bulletin* **34** (2000), 9-17.

[18] T. Sasaki and D. Inaba. Extended Hensel construction and multivariate algebraic functions. Preprint of Univ. Tsukuba (18 pages), 2007, submitted.

[19] T. Sasaki and D. Inaba. Convergence domains of series expansions of multivariate algebraic functions. Preprint of Univ. Tsukuba (in preparation).

[20] T. Sasaki and F. Kako. Solving multivariate algebraic equation by Hensel construction. *Japan J. Indust. Appl. Math.* **16** (1999), 257-285. (This paper has been written in 1993; the publication was delayed by a very slow refereeing procedure.)

[21] T. Sasaki, T. Kitamoto and F. Kako. Error analysis of power-series roots of multivariate algebraic equation. Preprint of Univ. Tsukuba, March 1994.

[22] T. Sasaki and S. Yamaguchi. An analysis of cancellation error in multivariate Hensel construction with floating-point number arithmetic. *Proc. ISSAC'98 (Intern'l Symp. on Symbolic and Algebraic Computation)*, O. Gloor (Ed.), ACM Press, 1-8, 1998.

[23] K. Shiihara and T. Sasaki. Analytic continuation and Riemann surface determination of algebraic functions by computer. *Japan J. Indust. Appl. Math.* **13** (1996), 107-116.

[24] P.S. Wang. An improved multivariate polynomial factoring algorithm. *Math. Comp.* **32** (1978), 1215-1231.

[25] D.Y.Y. Yun. *The Hensel lemma in algebraic manipulation.* Ph. D. Thesis, Dept. Math., M.I.T., Nov. 1973.

DIFFERENTIATION OF KALTOFEN'S DIVISION-FREE DETERMINANT ALGORITHM
Abstract

Gilles Villard
CNRS, Université de Lyon
Laboratoire LIP, CNRS-ENSL-INRIA-UCBL
46, Allée d'Italie, 69364 Lyon Cedex 07, France
`http://perso.ens-lyon.fr/gilles.villard`

Kaltofen has proposed a new approach in [8] for computing matrix determinants. The algorithm is based on a baby steps/giant steps construction of Krylov subspaces, and computes the determinant as the constant term of a characteristic polynomial. For matrices over an abstract field and by the results of Baur and Strassen [1], the determinant algorithm, actually a straight-line program, leads to an algorithm with the same complexity for computing the adjoint of a matrix [8]. However, the latter is obtained by the reverse mode of automatic differentiation and somehow is not "explicit". We study this adjoint algorithm, show how it can be implemented (without resorting to an automatic transformation), and demonstrate its use on polynomial matrices.

Kaltofen has proposed in [8] a new approach for computing matrix determinants. This approach has brought breakthrough ideas for improving the complexity estimate for the problem of computing the determinant without divisions over an abstract ring [8, 11]. The same ideas also lead to the currently best known bit complexity estimates for some problems on integer matrices such as the problem of computing the characteristic polynomial [11].

We consider the straigth-line programs of [8] for computing the determinant over abstract fields or rings (with or without divisions). Using the reverse mode of automatic differentiation (see [12, 13, 14]), a straight-line program for computing the determinant of a matrix A can be (automatically) transformed into a program for computing the adjoint matrix A^* of A [1] (see the application in [8, §1.2] and [11, Theorem 5.1]). Since the latter program is derived by an automatic process, few is known about the way it computes the adjoint. The only available information seems to be the determinant program itself and the knowledge we have on the differentiation process. In this paper we study the adjoint programs that would be automatically generated by differentiation from Kaltofen's determinant programs. We show how they can be implemented with and without divisions, and study their behaviour on univariate polynomial matrices.

Our motivation for studying the differentiation and resulting adjoint algorithms is the importance of the determinant approach of [8, 11] for various complexity estimates. Recent advances around the determinant of polynomial or integer matrices [5, 11, 15, 16], and the adjoint of a univariate polynomial matrix in the generic case [7], also justify the study of the general adjoint problem.

1 Kaltofen's determinant algorithm

Let K be a commutative field. We consider $A \in \mathsf{K}^{n\times n}$, $u \in \mathsf{K}^{n\times 1}$, and $v \in \mathsf{K}^{n\times 1}$. Kaltofen's approach extends the Krylov-based methods of [18, 9, 10]. We introduce the Hankel matrix $H = (uA^{i+j-2}v)_{ij} \in \mathsf{K}^{n\times n}$, and let $h_k = uA^k v$ for $0 \leq k \leq 2n-1$. We assume that H is non-singular. In the applications the latter is ensured either by construction of A, u, and v [8, 11], or by randomization (see [11] and references therein).

With baby steps/giant steps parameters $r = \lceil 2n/s \rceil$ and $s = \lceil \sqrt{n} \rceil$ $(rs \geq 2n)$ we consider the following algorithm (the algorithm without divisions will be described in Section 3).

```
Algorithm Det [8]
step 1. For $i = 0, 1, \ldots, r-1$ Do $v_i := A^i v$;
step 2. $B = A^r$;
step 3. For $j = 0, 1, \ldots, s-1$ Do $u_j := uB^j$;
step 4. For $i = 0, 1, \ldots, r-1$ Do
            For $j = 0, 1, \ldots, s-1$ Do $h_{i+jr} := u_j v_i$;
step 5. Compute the minimum polynomial $f(\lambda)$ of the sequence $\{h_k\}_{0 \leq k \leq 2n-1}$;
Return $f(0)$.
```

2 The ajoint algorithm

The determinant of A is a polynomial in $\mathsf{K}[a_{11}, \ldots, a_{ij}, \ldots, a_{nn}]$ of the entries of A. If we denote the adjoint matrix by A^* such that $AA^* = A^*A = (\det A)I$, then the entries of A^* satisfy [1]:

$$a^*_{j,i} = \frac{\partial \Delta}{\partial a_{i,j}}, 1 \leq i, j \leq n. \tag{1}$$

The reverse mode of automatic differentiation (see [1, 12, 13, 14]) allows to transform a program which computes Δ into a program which computes all the partial derivatives in (1). We apply the transformation process to Algorithm Det.

The flow of computation for the adjoint is reversed compared to the flow of Algorithm Det. Hence we start with the differentiation of Step 5. Consider the $n \times n$ Hankel matrices $H = (uA^{i+j-2}v)_{ij}$ and $H_A = (uA^{i+j-1}v)_{ij}$. Then the determinant $f(0)$ is computed as

$$\Delta = (\det H_A)/(\det H).$$

Viewing Δ as a function Δ_5 of the h_k's, we show that

$$\frac{\partial \Delta_5}{\partial h_k} = (\varphi_{k-1}(H_A^{-1}) - \varphi_k(H^{-1}))\Delta \tag{2}$$

where for a matrix $M = (m_{ij})$ we define $\varphi_k(M) = 0 + \sum_{i+j-2=k} m_{ij}$ for $1 \leq k \leq 2n-1$. Identity (2) gives the first step of the adjoint algorithm. Over an abstract field, and using intermediate data from Algorithm Det, its costs is essentially the cost of a Hankel matrix inversion.

For differentiating Step 4, Δ is seen as a function Δ_4 of the v_i's and u_j's. The entries of v_i are involved in the computation of the s scalars $h_i, h_{i+r}, \ldots, h_{i+(s-1)r}$. The entries of u_j are used for computing the r scalars $h_{jr}, h_{1+jr}, \ldots, h_{(r-1)+jr}$. Let ∂v_i be the $1 \times n$ vector, respectively the $n \times 1$ vector ∂u_j, whose entries are the derivatives of Δ_4 with respect to the entries of v_i, respectively u_j. We show that

$$\begin{bmatrix} \partial v_0 \\ \partial v_1 \\ \vdots \\ \partial v_{r-1} \end{bmatrix} = H^v \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{s-1} \end{bmatrix} \tag{3}$$

and

$$\begin{bmatrix} \partial u_0, \partial u_1, \ldots \partial u_{s-1} \end{bmatrix} = \begin{bmatrix} v_0, v_1, \ldots v_{r-1} \end{bmatrix} H^u \tag{4}$$

where H^v and H^u are $r \times s$ matrices whose entries are selected $\partial\Delta_5/\partial h_k$'s. Identities (3) and (4) give the second step of the adjoint algorithm. Its costs is essentially the cost of two $n \times \sqrt{n}$ by $\sqrt{n} \times \sqrt{n}$ (unstructured) matrix products.

Note that (2), (3) and (4) somehow call to mind the matrix factorizations [3, (3.5)] (our objectives are similar to Eberly's ones) and [4, (3.1)].

Steps 3-1 of Det may then be differentiated. For differentiating Step 3 we recursively compute an $n \times n$ matrix ∂B from the δu_j's. The matrix ∂B gives the derivatives of Δ_3 (the determinant seen as a function of B and the v_i's) with respect to the entries of B.

For Step 2 we recursively compute from δB an $n \times n$ matrix δA that gives the derivatives of Δ_2 (the determinant seen as a function of v_i's).

Then the differentiation of Step 1 computes from δA and the δv_i's an update of δA that gives the derivatives of $\Delta_1 = \Delta$. From (1) we know that $A^* = (\delta A)^T$.

The recursive process for differentiating Step 3 to Step 1 may be written in terms of the differentiation of the basic operation (or its transposed operation)

$$q := p \times M \tag{5}$$

where p and q are row vectors of dimension n and M is an $n \times n$ matrix. We assume at this point (recursive process) that column vectors δp and δq of derivatives with respect to the entries of p and q are available. We also assume that an $n \times n$ matrix δM that gives the derivatives with respect to the m_{ij}'s has been computed. We show that differentiating (5) amounts to updating δp and δM as follows:

$$\begin{cases} \delta p := \delta p + M \times \delta q, \\ \delta M := \delta M + p^T \times (\delta q)^T. \end{cases} \tag{6}$$

We see that the complexity is essentially preserved between (5) and (6) and corresponds to a matrix by vector product. In particular, if Step 2 of Algorithm Det is implemented in $O(\log r)$ matrix products, then Step 2 differentiation will cost $O(n^3 \log r)$ operations (by decomposing the $O(n^3)$ matrix product).

Let us call Adjoint the algorithm just described for computing A^*.

3 Application to computing the adjoint without divisions

Now let A be an $n \times n$ matrix over an abstract ring R. Kaltofen's method for computing the determinant of A without divisions applies Algorithm Det on a well chosen univariate polynomial matrix $Z(z) = C + z(A - C)$ where $C \in \mathbb{Z}^{n \times n}$. The choice of C as well as a dedicated choice for the projections u and v allow the use of Strassen's general method of avoiding divisions [17, 8]. The determinant is a polynomial Δ of degree n, the arithmetic operations in Det are replaced by operations on power series modulo z^{n+1}. Once the determinant of $Z(z)$ is computed, $(\det Z)(1) = \det(C + 1 \times (A - C))$ gives the determinant of A.

In Step 1 and Step 2 in Algorithm Det applied to $Z(z)$ the matrix entries are actually polynomials of degree at most $\sqrt{n}$. This is a key point for reducing the overall complexity estimate of the problem. Since the adjoint algorithm has a reversed flow, this key point does not seem to be relevant for Adjoint. For computing $\det A$ without divisions, Kaltofen's algorithm goes through the computation of $\det Z(z)$. Adjoint applied to $Z(z)$ computes A^* but does not seem to compute $Z^*(z)$ with the same complexity. In particular, differentiation of Step 3 using (6) leads to products $A^l(\delta B)^T$ that are more expensive over power series (one computes $A(z)^l(\delta B(z))^T$) than the initial computation in Det A^r ($A(z)^r$ on series).

For computing A^* without divisions only $Z^*(1)$ needs to be computed. We extend algorithm Adjoint with input $Z(z)$ by evaluating polynomials (truncated power series) partially. With a final evaluation at $z = 1$ in mind, a polynomial $p(z) = p_0 + p_1 z + \ldots + p_{n-1}z^{n-1} + p_n z^n$ may typically be replaced by

$(p_0 + p_1 + \ldots + p_m) + p_{m+1}x^{m+1} + \ldots + p_{n-1}z^{n-1} + p_n z^n$ as soon as any subsequent use of $p(z)$ will not require its coefficients of degree less than m.

4 Fast matrix product and application to polynomial matrices

We show how to integrate asymptotically fast matrix products in Algorithm Ajoint. On univariate polynomial matrices $A(z)$ with power series operations modulo z^n, Algorithm Adjoint leads to intermediary square matrix products where one of the operand has a degree much smaller than the other. In this case we show how to use fast rectangular matrix products [2, 6] for a (tiny) improvement of the complexity estimate of general polynomial matrix inversion.

Concluding remarks

Our understanding of the differentiation of Kaltofen's determinant algorithm has to be improved. We have proposed an implementation whose mathematical explanation remains to be given. Our work also has to be generalized to the block algorithm of [11].

Acknoledgements. We thank Erich Kaltofen who has brought reference [14] to our attention.

References

[1] W. Baur and V. Strassen. The complexity of partial derivatives. *Theor. Comp. Sc.*, 22:317–330, 1983.

[2] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. of Symbolic Computations*, 9(3):251–280, 1990.

[3] W. Eberly. Processor-efficient parallel matrix inversion over abstract fields: two extensions. In *Proc. Second International Symposium on Parallel Symbolic Computation, Maui, Hawaii, USA*, pages 38–45. ACM Press, Jul 1997.

[4] W. Eberly, M. Giesbrecht, P. Giorgi, A. Storjohann, and G. Villard. Faster inversion and other black box matrix computation using efficient block projections. In *Proc. International Symposium on Symbolic and Algebraic Computation, Waterloo, Canada*, pages 143–150. ACM Press, August 2007.

[5] M. Giesbrecht, W. Eberly, and G. Villard. Fast computations of integer determinants. In *The 6th International IMACS Conference on Applications of Computer Algebra, St. Petersburg, Russia*, June 2000.

[6] X. Huang and V.Y. Pan. Fast rectangular matrix multiplications and improving parallel matrix computations. In *Proc. Second International Symposium on Parallel Symbolic Computation, Maui, Hawaii, USA*, pages 11–23, Jul 1997.

[7] C.P. Jeannerod and G. Villard. Asymptotically fast polynomial matrix algorithms for multivariable systems. *Int. J. Control*, 79(11):1359–1367, 2006.

[8] E. Kaltofen. On computing determinants without divisions. In *International Symposium on Symbolic and Algebraic Computation, Berkeley, California USA*, pages 342–349. ACM Press, July 1992.

[9] E. Kaltofen and V.Y. Pan. Processor efficient parallel solution of linear systems over an abstract field. In *Proc. 3rd Annual ACM Symposium on Parallel Algorithms and Architecture*, pages 180–191. ACM-Press, 1991.

[10] E. Kaltofen and B.D. Saunders. On Wiedemann's method of solving sparse linear systems. In *Proc. AAECC-9*, LNCS 539, Springer Verlag, pages 29–38, 1991.

[11] E. Kaltofen and G. Villard. On the complexity of computing determinants. *Computational Complexity*, 13:91–130, 2004.

[12] S. Linnainmaa. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors (in Finnish). Master's thesis, University of Helsinki, Dpt of Computer Science, 1970.

[13] S. Linnainmaa. Taylor expansion of the accumulated rounding errors. *BIT*, 16:146–160, 1976.

[14] G. M. Ostrowski, Ju. M. Wolin, and W. W. Borisow. Über die Berechnung von Ableitungen (in German). *Wissenschaftliche Zeitschrift der Technischen Hochschule für Chemie, Leuna-Merseburg*, 13(4):382–384, 1971.

[15] A. Storjohann. High-order lifting and integrality certification. *Journal of Symbolic Computation*, 36(3-4):613–648, 2003. Special issue International Symposium on Symbolic and Algebraic Computation (ISSAC'2002). Guest editors: M. Giusti & L. M. Pardo.

[16] A. Storjohann. The shifted number system for fast linear algebra on integer matrices. *Journal of Complexity*, 21(4):609–650, 2005.

[17] V. Strassen. Vermeidung von Divisionen. *J. Reine Angew. Math.*, 264:182–202, 1973.

[18] D. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Transf. Inform. Theory*, IT-32:54–62, 1986.

Summation of Linear Recurrence Sequences

Robert A. Ravenscroft, Jr.[*]
Edmund A. Lamagna
Department of Computer Science and Statistics
University of Rhode Island
Kingston, Rhode Island 02881 USA

Abstract

We describe an simple and efficient algorithm for evaluating summations of sequences defined by linear recurrences with constant coefficients. The result is expressed in finite terms using the sequence name. The algorithm can evaluate all homogeneous recurrences and a large variety of inhomogeneous recurrences.

1 Introduction

The literature contains many examples of summations involving symbolic function names that express the results of the sums using the symbolic names. For example, the sum of the first $n+1$ Fibonacci numbers is given by $\sum_{k=0}^{n} F_k = F_{n+2} - 1$ and the sum of the first n harmonic numbers is given by $\sum_{k=1}^{n} H_k = (n+1)H_n - n$. The sequences $\langle F_k \rangle$ and $\langle H_k \rangle$ are examples of what we call *linear recurrence sequences*. These sequences are defined by linear recurrences with constant coefficients. In this paper we develop an algorithm that evaluates indefinite summations involving symbolic names of linear recurrence sequences and expresses its result using the symbolic sequence names. This method is a formalization of an ad hoc technique that is used to evaluate the summation of the Fibonacci numbers. It leads to a procedure that is able to evaluate the indefinite sum of all homogeneous linear recurrence sequences and many inhomogeneous linear recurrence sequences.

Indefinite summation involves finding S_n for the equation $S_n = \sum_{k=0}^{n} a_k$, where a_k depends on k but not on n. When a_k is an algebraic function of k, we want to find algebraic closed forms for S_n. Algorithms such as those of Gosper [1], Karr [3], and Moenck [4], as well as generating function techniques [6, 7], can be used to do so. However, when a_k involves symbolic function names, there are two choices of how to express S_n. We can apply the definition or closed form of the special function and attempt to find a closed form for S_n. Unfortunately, this is not always possible. For example, $\sum_{k=1}^{n} H_n$ cannot be expressed in closed form as a function of n with only elementary functions. The second choice is to express the summation in finite terms using the symbolic function names that are used in the summand. Using this approach we have $\sum_{k=1}^{n} H_k = (n+1)H_n - n$.

Even though a closed form may exist for an indefinite sum, it may be more informative to use symbolic names to evaluate a sum. Using symbolic names in the expression for S_n may reveal interesting information about the structure of the expression, whereas the complexity of the closed form may well obscure that information. For example, summations involving the Fibonacci numbers appear frequently in the literature. In most of these cases, including $\sum_{k=0}^{n} F_k = F_{n+2} - 1$, the sums are expressed using F_n, the symbolic name of the Fibonacci numbers, rather than the closed form for the Fibonacci numbers. This clearly expresses the role that the Fibonacci numbers play in the results.

[*]Also an adjunct instructor at the Department of Mathematics and Computer Science, Rhode Island College, Providence, Rhode Island 02908 USA.

There are also practical, computational advantages to computing with the symbolic function names rather than their closed forms. Typically, expressions involving symbolic function names are rational functions with rational coefficients of the symbols in the expression. On the other hand, the closed form of many sequences involve radicals and complex numbers. Unfortunately, computer algebra systems are not as fast when computing with radicals and complex numbers as they are when computing with rational numbers and rational functions. From a system programming standpoint, we find it desirable to maximize the computation done with symbolic function names. Then other forms of the result can be obtained from the finite expression. Closed forms can be found by substituting for the symbolic function names. Asymptotics can be found by applying asymptotic techniques to the sequences. Numerical results can be obtained by substituting actual values for the sequences.

Several algorithms have been developed to evaluate indefinite summations that involve symbolic function names and to express their results in finite terms using symbolic names. Karr's algorithm uses the theory of difference fields to solve first order difference equations, of which indefinite summation is a special case [3]. This algorithm can determine if there is an expression using the symbols in an extension field that solves the difference equation. Russell's method evaluates indefinite sums whose summands involve term-wise products and linear indexing of homogeneous linear recurrence sequences [8]. Greene and Wilf also consider summations whose summands involve term-wise products and linear indexing of homogenous linear recurrences [2]. Savio, Lamagna and Liu have implemented algorithms for evaluating indefinite sums that involve harmonic numbers [9]. The authors have previously implemented algorithms using rational generating functions to evaluate indefinite summations that involve symbolic names of homogeneous linear recurrence sequences, term-wise products of homogeneous linear recurrence sequences, and linear indexing of homogeneous linear recurrence sequences [5, 6, 7]. Each of these algorithms can evaluate some of the summations that the algorithm described here is able to evaluate. However, none of them can handle all of the summations that the method we develop can.

2 Simple Sums

The Fibonacci numbers $\langle F_0, F_1, F_2, F_3, F_4, F_5, \ldots\rangle = \langle 0, 1, 1, 2, 3, 5, \ldots\rangle$ form a sequence generated by the constant coefficient linear recurrence relation $F_k = F_{k-1} + F_{k-2}$, with initial conditions $F_0 = 0$ and $F_1 = 1$. A well known result in combinatorial mathematics expresses the indefinite sum of the numbers in this sequence in terms of a Fibonacci number,

$$\sum_{k=0}^{n} F_k \;=\; F_{n+2} - 1.$$

It is instructive to begin our investigation by deriving this relationship. We start by writing

$$\begin{aligned} F_n &= F_{n-1} + F_{n-2} \\ F_{n-1} &= F_{n-2} + F_{n-3} \\ &\;\;\vdots \\ F_2 &= F_1 + F_0 \end{aligned}$$

Adding these equations together and using the sum of interest to express the result, we have

$$\sum_{k=0}^{n} F_k - F_1 - F_0 \;=\; \sum_{k=0}^{n} F_k - F_n - F_0 + \sum_{k=0}^{n} F_k - F_n - F_{n-1}.$$

Substituting the initial conditions and isolating the sum, we obtain the desired result,

$$\sum_{k=0}^{n} F_k \;=\; 2F_n + F_{n-1} - 1 \;=\; F_{n+1} + F_n - 1 \;=\; F_{n+2} - 1.$$

This same technique can be applied to more complicated linear recurrences, as seen in the next example.

Example 1 The inhomogeneous recurrence $G_k = G_{k-1} + 2G_{k-2} + 1$, with initial conditions $G_0 = 0$ and $G_1 = 1$, gives the number of code words in a k-digit binary Gray code. Find $S_n = \sum_{k=0}^{n} G_k$.

Proceeding in the same manner as above, we have

$$\begin{aligned} G_n &= G_{n-1} + 2G_{n-2} + 1 \\ G_{n-1} &= G_{n-2} + 2G_{n-3} + 1 \\ &\vdots \\ G_2 &= G_1 + 2G_0 + 1 \end{aligned}$$

Summing these equations and expressing the result in terms of S_n, we obtain

$$S_n - G_1 - G_0 \;=\; S_n - G_n - G_0 + 2(S_n - G_n - G_{n-1}) + n - 1.$$

Thus, we can express the sum as

$$S_n \;=\; \sum_{k=0}^{n} G_k \;=\; \frac{3}{2}G_n + G_{n-1} - \frac{n}{2}. \qquad \blacksquare$$

We are now ready to consider the general case. Consider a linear recurrence with constant coefficients of order d having inhomogeneous part t_k,

$$T_k \;=\; \alpha_1 T_{k-1} + \alpha_2 T_{k-2} + \cdots + \alpha_d T_{k-d} + t_k.$$

In this case, d terms of the sequence, T_n, T_{n-1}, $\ldots$, T_{n-d+1}, and d initial conditions, T_0, T_1, $\ldots$, T_{d-1}, will be required to express the sum. Using the technique described above, it is straightforward to prove the following result.

Theorem 1 *The indefinite sum of the elements in the sequence $\langle T_0, T_1, T_2, \ldots \rangle$ generated by the linear recurrence $T_k = \sum_{i=1}^{d} \alpha_i T_{k-i} + t_k$ is given*

$$\sum_{k=0}^{n} T_k \;=\; \frac{1}{1 - \sum_{i=1}^{d} \alpha_i} \left(\sum_{j=0}^{d-1} T_j - \sum_{i=1}^{d} \alpha_i \left(\sum_{j=0}^{d-i-1} T_j + \sum_{j=n-i+1}^{n} T_j \right) + \sum_{k=d}^{n} t_k \right),$$

provided that $\sum_{i=1}^{d} \alpha_i \neq 1$.

The proof follows by direct algebraic manipulation of the relation

$$\sum_{k=d}^{n} T_k \;=\; \sum_{k=d}^{n} \left(\sum_{i=1}^{d} \alpha_i T_{k-i} + t_k \right).$$

One difficulty is that this theorem cannot be applied when $\sum_{i=1}^{d} \alpha_i = 1$. This is equivalent to the condition that the characteristic polynomial of the homogeneous part of the recurrence

$$p(x) \;=\; x^d - \alpha_1 x^{d-1} - \alpha_2 x^{d-2} - \cdots - \alpha_d$$

has as a factor $x - 1$ raised to a positive integer power. The value $p(1)$ is the coefficient of the sum when we isolate it on the left side of the equation. In this case, we find that $p(1) = 0$ because the indefinite sums on the left and right sides cancel when we gather terms. We extend our algorithmic technique to deal with this case in the next section.

A second possible difficulty concerns the sum of the inhomogeneous part, $\sum_{k=d}^{n} t_k$. This problem is solved if a closed form can be obtained by using an existing summation algorithm such as those of Karr, Gosper, or Moenck. Alternatively, if t_k can be described as a linear recurrence with constant coefficients, then the techniques developed here can be used to express the sum in finite terms using the symbolic sequence name t_n.

3 Summation with Multiplication Factors

As shown in the previous section, we cannot use Theorem 1 if $\sum_{i=1}^{d} \alpha_i = 1$. However, the technique described there can be extended by choosing an appropriate multiplication factor and multiplying both sides of the recurrence by this factor before summing. The following examples suggest how this is done.

Example 2 The first order recurrence $H_n = H_{n-1} + 1/n$ with $H_1 = 1$ defines the harmonic number $H_n = \sum_{k=1}^{n} 1/k$. Express $S_n = \sum_{k=1}^{n} H_k$ in terms of H_n.

Instead of summing directly, we first multiply both sides of the recurrence for H_n by n.

$$\begin{aligned} nH_n &= nH_{n-1} + 1 \\ (n-1)H_{n-1} &= (n-1)H_{n-2} + 1 \\ &\vdots \\ 2H_2 &= 2H_1 + 1 \end{aligned}$$

Summing these equations gives

$$\sum_{k=2}^{n} kH_k = \sum_{k=1}^{n-1} (k+1)H_k + n - 1.$$

Distributing the sum on the right, adjusting the limits of summation and using the definition of S_n, we have

$$\sum_{k=1}^{n} kH_k - H_1 = \sum_{k=1}^{n} kH_k - nH_n + S_n - H_n + n - 1.$$

The multiplication factor causes the sums $\sum_{k=1}^{n} kH_k$ to cancel, allowing us to isolate S_n,

$$S_n = (n+1)H_n - n. \quad ∎$$

Example 3 Evaluate $S_n = \sum_{k=0}^{n} A_k$, where A_k is given by the second order linear recurrence $A_k = 3A_{k-1} - 2A_{k-2}$ with $A_0 = 0$ and $A_1 = 1$.

As in Example 2, multiply both sides of the recurrence by k, sum over $2 \le k \le n$, and adjust the limits of the summation.

$$\sum_{k=2}^{n} kA_k = 3\sum_{k=2}^{n} kA_{k-1} - 2\sum_{k=2}^{n} kA_{k-2} = 3\sum_{k=1}^{n-1} (k+1)A_k - 2\sum_{k=0}^{n-2} (k+2)A_k.$$

Now expand the sums on the right, adjust the limits of summation to $0 \le k \le n$, and apply the definition of S_n. Letting $C_n = \sum_{k=0}^{n} kA_k$, this gives

$$C_n - A_1 = 3\left(C_n + S_n - (n+1)A_n - A_0\right) - 2\left(C_n + 2S_n - (n+1)A_{n-1} - (n+2)A_n\right).$$

Isolating S_n, the C_n terms disappear and we have

$$S_n = 2(n+1)A_{n-1} - (n-1)A_n + 1. \quad ∎$$

Example 4 Consider the recurrence $B_k = 2B_{k-1} - B_{k-2} + 1$ with initial conditions $B_0 = 0$ and $B_1 = 1$. Evaluate $S_n = \sum_{k=0}^{n} B_k$.

Once again, multiply the recurrence by k and sum over $2 \le k \le n$.

$$\sum_{k=2}^{n} kB_k = 2\sum_{k=2}^{n} kB_{k-1} - \sum_{k=2}^{n} kB_{k-2} + \sum_{k=2}^{n} k$$

Proceeding as in the previous examples and letting $C_n = \sum_{k=0}^{n} kB_k$, we have

$$\begin{aligned} C_n - B_1 &= 2\left(C_n + S_n - 2(n+1)B_n - 2B_0\right) \\ &\quad - \left(C_n + 2S_n - (n+1)B_{n-1} - (n+2)B_n\right) + \frac{(n+2)(n-1)}{2}. \end{aligned}$$

This time when we try to isolate S_n, both C_n and S_n disappear! However, if we multiply the recurrence by k^2 instead of k before summing, we can remedy this difficulty.

$$\begin{aligned} \sum_{k=2}^{n} k^2 B_k &= 2\sum_{k=2}^{n} k^2 B_{k-1} - \sum_{k=2}^{n} k^2 B_{k-2} + \sum_{k=2}^{n} k^2 \\ &= 2\sum_{k=1}^{n-1} (k+1)^2 B_{k-1} - \sum_{k=0}^{n-2} (k+2)^2 B_{k-2} + \sum_{k=2}^{n} k^2. \end{aligned}$$

Now, expanding the sums involving B_k, letting $C_n = \sum_{k=0}^{n} kB_k$, and letting $D_n = \sum_{k=0}^{n} k^2 B_k$, we have

$$D_n - B_1 = 2\left(D_n + 2C_n + S_n - (n+1)^2 B_n - B_0\right) - \left(D_n + 4C_n + 4S_n - (n+1)^2 B_{n-1} - (n+2)B_n\right) + \sum_{k=2}^{n} k^2.$$

When we isolate S_n, both the C_n and D_n terms disappear, leaving

$$S_n = \frac{-(n^2-2)}{2} B_n + \frac{(n+1)^2}{2} B_{n-1} - \frac{n(n+1)(2n+1)}{12}. \qquad \blacksquare$$

4 The General Algorithm

How do we choose the multiplication factor, in general? The characteristic polynomial of the homogeneous part provides the key. The following table summarizes the results of the previous section.

recurrence	characteristic polynomial	factor
$H_k = H_{k-1}$	$x - 1$	k
$A_k = 3A_{k-1} - 2A_{k-2}$	$x^2 - 3x + 2 = (x-1)(x-2)$	k
$B_k = 2B_{k-1} - B_{k-2} + 1$	$x^2 - 2x + 1 = (x-1)^2$	k^2

The multiplication factor depends not on the order of the recurrence but rather on the largest m such that $(x-1)^m$ is a factor of the characteristic polynomial. If $m = 0$, then Theorem 1 applies. Otherwise, multiply both sides of the recurrence by k^m before summing. Actually, any polynomial in k of degree m can be used as the multiplication factor. It will prove convenient to use falling powers, $k^{\underline{m}} = k(k-1)\cdots(k-m+1)$, instead of ordinary powers, in what follows.

We now formalize the techniques we have developed into an algorithm for summing linear recurrence sequences. Given the recurrence $T_k = \sum_{i=1}^{d} \alpha_i T_{k-i} + t_k$ and d initial conditions T_i for $0 \le i \le d-1$, we can find $S_n = \sum_{k=0}^{n} T_k$ by performing the following steps:

1. Find $p(x) = x^d - \sum_{i=1}^{m} \alpha_i x^{d-i}$, the characteristic polynomial of the recurrence.
2. Determine m, the largest integer such that $(x-1)^m$ is a factor of $p(x)$.
3. If $m = 0$, use Theorem 1 to evaluate S_n. Otherwise, perform steps 4 through 8.
4. Multiply both sides of the recurrence by $k^{\underline{m}}$.
5. Sum both sides of the equation from step 4 over $d \le k \le n$.
6. Perform the following steps for $1 \le i \le d$:

(a) Let $k^{\underline{m}} = \sum_{j=0}^{m} m^{\underline{j}} \binom{i}{j} (k-i)^{\underline{m-j}}$.

(b) Substitute this value for $k^{\underline{m}}$ in the coefficient of T_{k-i} in the equation from step 5.

7. Let $C_{j,n} = \sum_{k=0}^{n} k^{\underline{j}} T_k$, for $1 \le j \le m$. Express the equation from step 6 in terms of S_n and the values $C_{j,n}$.

8. Substitute initial conditions into the equation from step 7, cancel terms and solve for S_n.

This algorithm can be applied successfully to any homogeneous linear recurrence sequence. The procedure will also succeed for inhomogeneous linear recurrence sequences if the sum $\sum_{k=d}^{n} k^{\underline{m}} t_k$ resulting from the inhomogeneous part can be evaluated in closed form or finite terms.

The following theorem states the correctness of our algorithm. We present a brief sketch of the significant portions of the proof, assuming a summation factor of $k^{\underline{m}}$. Different summation factors give different polynomial coefficients in the result.

Theorem 2 *Given the linear recurrence relation*

$$T_k = \sum_{i=1}^{d} \alpha_i T_{k-i} + t_k$$

and d initial conditions, the algorithm described above will find an expression in finite terms to all summations of the form $S_n = \sum_{k=0}^{n} T_k$ provided $\sum_{k=d}^{n} k^{\underline{m}} t_k$ can be evaluated.

In step 3 of the algorithm, if $m = 0$, the sum S_n is found by applying Theorem 1. All homogeneous linear recurrence sequences for which $p(1) \ne 0$ can be solved using this theorem.

The next major part of the proof is the identity used in step 6a. This can be proved by induction on i. The important step in the inductive part of the proof involves applying the equation $(k-i)^{\underline{m-j}} = (k-(i+1))^{\underline{m-j}} + (m-j)(k-(i+1))^{\underline{m-j-1}}$ and then algebraically rearranging the resulting sum to obtain the desired result.

The last major part of the proof is step 8. In order to solve for S_n, we must show that the coefficients of the values $C_{j,n}$ cancel and that the coefficient of S_n is nonzero. If we collect the desired terms on the left side of the equation, we find that the coefficient of $C_{m,n}$ is $c_m = 1 - \sum_{i=1}^{d} \alpha_i$, the coefficient of $C_{j,n}$ for $1 \le j < m$ is $c_j = -\sum_{i=m-j}^{d} m^{\underline{m-j}} \binom{i}{m-j} \alpha_i$, and the coefficient of S_n is $s = \sum_{i=m}^{d} m! \binom{i}{m} \alpha_i$. We then show that $c_m = p(1)$, $c_j = m^{\underline{m-j}} p^{(m-j)}(1)/(m-j)!$ for $1 \le j < m$, and $s = p^{(m)}(1)$, where $p^{(j)}(x) = d^j p(x)/dx^j$. Since $p(x) = (x-1)^m q(x)$ and $x-1$ is not a factor of $q(x)$, we can prove that $p^{(j)}(1) = 0$ for $1 \le j \le m$ and that $p^{(m)}(1) \ne 0$. As a result, we find that $c_j = 0$ for $1 \le j \le m$ and $s \ne 0$.

5 Application to Some Special Sequences

The algorithm presented in this paper has proven to be a useful tool for the evaluation of indefinite summations of linear recurrence sequences. It is able to evaluate all sums of homogeneous linear recurrence sequences and many sums of inhomogeneous linear recurrence sequences. Moreover, the techniques discussed in the previous section enable us to use the algorithm to sum sequences generated by linear indexing and term-wise products of homogeneous linear recurrence sequences.

Linear indexing uses a linear polynomial of the summation variable to index the sequence, as in $F_{a \cdot k+b}$ where k is the summation index, a and b are integers and $a > 0$. Greene and Wilf generalize linear indexing to use both the summation index and limit, as in $F_{a \cdot n+b \cdot k+c}$ where n is the summation limit, k is the sum index, a, b, and c are integers, $a > 0$, and $a+b > 0$. When $b < 0$, the sum is a generalized form of convolution. Term-wise products multiply terms of two or more linear recurrence sequences, as in $\langle F_k G_k \rangle$. While it is beyond the scope of this paper, it can be shown that any sequences generated by use of linear indexing and term-wise products have linear recurrences with constant coefficients [6, 7].

We have implemented the algorithm presented in Section 4 using Maple. The following examples show results the procedure is capable of producing on some of these special sequences.

Example 5 Evaluate $S_n = \sum_{k=0}^{n} F_{2k}$, where the summand uses linear indexing of the Fibonacci numbers F_k. The summand F_{2k} has the recurrence $F_{2k} = F_{2(k-1)} - F_{2(k-2)}$. Application of our procedure gives

$$S_n \;=\; \sum_{k=0}^{n} F_{2k} \;=\; 2F_{2n} - F_{2(n-1)} - 1.$$

Using the Fibonacci recurrence we can rewrite this as

$$S_n \;=\; F_{2n+1} - 1. \qquad \blacksquare$$

Example 6 Evaluate $S_n = \sum_{k=0}^{n} F_k^2$, where the summand is the term-wise product of the Fibonacci numbers with themselves. The summand F_k^2 has the recurrence $F_k^2 = 2F_{k-1}^2 + 2F_{k-2}^2 - F_{k-3}^2$. Application of our procedure gives

$$S_n \;=\; \frac{3}{2}F_n^2 + \frac{1}{2}F_{n-1}^2 - \frac{1}{2}F_{n-2}^2.$$

Using the Fibonacci recurrence, this can be rewritten as

$$S_n \;=\; F_n^2 + F_n F_{n-1}. \qquad \blacksquare$$

Example 7 Evaluate $S_n = \sum_{k=1}^{n} kH_k$. The summand is a term-wise product and has the recurrence $kH_k = 2(k-1)H_{k-1} - (k-2)H_{k-2} + 1/(k-1)$. Application of the procedure gives

$$S_n \;=\; \frac{-(n^2-n-2)}{2}nH_n + \frac{(n^2+n)}{2}(n-1)H_{n-1} + 1 + \frac{1}{2}\sum_{k=3}^{n} k.$$

The summation results from the inhomogeneous part of the recurrence and is easily evaluated using our algorithm. Thus we find that

$$\begin{aligned} S_n \;&=\; \frac{-(n^2-n-2)}{2}nH_n + \frac{(n^2+n)}{2}(n-1)H_{n-1} + \frac{(n^2+n-2)}{4} \\ &=\; \frac{(n^2+n)}{2}H_n - \frac{n(n-1)}{4}. \qquad \blacksquare \end{aligned}$$

As a final example, we show how the algorithm can be used with summations involving binomial coefficients.

Example 8 Evaluate the summation $S_n = \sum_{k=0}^{n} \binom{k}{m}$. We observe that $\binom{k}{m}$ is a polynomial in k of degree m. Thus we could find an order $m+2$ linear recurrence for $\binom{k}{m}$. Instead, we will use the recurrence $\binom{k}{m} = \binom{k-1}{m} + \binom{k-1}{m-1}$. We treat $\binom{k-1}{m-1}$ as the inhomogeneous part of the recurrence. The recurrence has a characteristic polynomial of $p(x) = x - 1$. Thus we need to use a multiplication factor of k. Multiplying by k and summing over $1 \le k \le n$ gives

$$\sum_{k=1}^{n} k\binom{k}{m} \;=\; \sum_{k=1}^{n}(k-1)\binom{k-1}{m} + \sum_{k=1}^{n}\binom{k-1}{m} + \sum_{k=1}^{n} k\binom{k-1}{m-1}.$$

We can absorb the factor of k into the binomial coefficient in the last sum and substitute S_n and $C_{1,n} = \sum_{k=0}^{n} k\binom{k}{m}$ to find

$$C_{1,n} \;=\; C_{1,n} - n\binom{n}{m} + S_n - \binom{n}{m} + mS_n.$$

Solving for S_n gives

$$S_n \;=\; \frac{n+1}{m+1}\binom{n}{m} \;=\; \binom{n+1}{m+1}. \qquad \blacksquare$$

6 Conclusion

Our algorithm has proven to be a simple and effective method for evaluating summations involving sequences defined by linear recurrences with constant coefficients and expressing the result in finite terms involving the symbolic sequence name. It is able to solve summations that none of the other summation algorithms can handle. Moenck's algorithm and Gosper's algorithm do not deal with summands defined by recurrences. Karr's algorithm works on sequences defined by first order difference equations. Russell's method can evaluate the homogeneous case of these sums using a symbolic sequence name. However, it cannot handle recurrences where the characteristic polynomial $p(x)$ has $x - 1$ as a factor. Greene and Wilf can handle all of the cases that Russell's method does not. In addition, they handle convolutions involving homogeneous linear recurrence sequences. Our generating function technique can evaluate these sums using the symbolic sequence name. However, that approach requires a paradigm shift to and from generating functions and involves extensive use of partial fraction techniques. In comparison, the algorithm in this paper involves a simple and straightforward manipulation of the recurrence that is easier to implement and runs more efficiently. In addition, the authors' generating function techniques are more limited in the types of inhomogeneous terms that can be handled.

Acknowledgement

The first author would like to thank Dr. Keith Geddes and the Symbolic Computation Group at the University of Waterloo. Some of the ideas and techniques used in this paper resulted from work he did while he was a postdoc with the Symbolic Computation Group in 1991–1992. Even this many years later, his time spent at Waterloo is having an impact on his professional career.

References

[1] R. W. Gosper, Jr., "Indefinite Hypergeometric Sums in MACSYMA," *Proceedings of the MACSYMA User's Conference*, Berkeley, CA., pp. 237–251, 1977.

[2] C. Greene and H. S. Wilf, "Closed Form Summation of C-Finite Sequences," *Transactions of the American Mathematical Society*, Vol. 359, No. 3, pp. 1161–1189, 2007.

[3] M. Karr, "Summation in Finite Terms," *Journal of the ACM*, Vol. 28, No. 2, pp. 305-350, 1981.

[4] R. Moenck, "On Computing Closed Forms for Summation," *Proceedings of the MACSYMA User's Conference*, Berkeley, CA, pp. 225–236, 1977.

[5] R. A. Ravenscroft, Jr. and E. A. Lamagna, "Symbolic Summation with Generating Functions," *Proceedings of the 1989 International Symposium on Symbolic and Algebraic Computation*, ACM Press, pp. 228–233, 1989.

[6] R. A. Ravenscroft, Jr., "Generating Function Algorithms for Symbolic Computation," Ph. D. Thesis, Department of Computer Science, Brown University, 1991.

[7] R. A. Ravenscroft, Jr., "Rational Generating Function Applications in Maple," *Maple V: Mathematics and Its Application, Proceedings of the Maple Summer Workshop and Symposium*, R. J. Lopez (ed.), Birkhäuser, pp. 122–128, 1994.

[8] D. L. Russell, "Sums of Recurrences of Terms from Linear Recurrence Sequences," *Discrete Mathematics*, Vol. 28, No. 1, pp. 65–79, 1979.

[9] D. Y. Savio, E. A. Lamagna and S. Liu, "Summation of Harmonic Numbers," *Computers and Mathematics*, E. Kaltofin and S. M. Watt (eds.), Springer-Verlag, pp. 12–20, 1989.

Compressed Modular Matrix Multiplication

Jean-Guillaume Dumas[*] Laurent Fousse[*] Bruno Salvy[†]

October 22, 2008

Abstract

Matrices of integers modulo a small prime can be compressed by storing several entries into a single machine word. Modular addition is performed by addition and possibly subtraction of a word containing several times the modulus. We show how modular multiplication can also be performed. In terms of arithmetic operations, the gain over classical matrix multiplication is equal to the number of integers that are stored inside a machine word. The gain in actual speed is also close to that number.

First, modular dot product can be performed via an integer multiplication by the reverse integer. Modular multiplication by a word containing a single residue is also possible. We give bounds on the sizes of primes and matrices for which such a compression is possible. We also make explicit the details of the required compressed arithmetic routines and show some practical performance.

Keywords : Kronecker substitution ; Finite field ; Modular Polynomial Multiplication ; REDQ (simultaneous modular reduction) ; DQT (Discrete Q-adic Transform) ; FQT (Fast Q-adic Transform).

1 Introduction

Compression of matrices over fields of characteristic 2 is classically made via the binary representation of machine integers and has numerous uses in number theory [1, 10]. The need for efficient matrix computations over *very small* finite fields of characteristic other than 2 arises in particular in graph theory (adjacency matrices), see, e.g., [11] or [12].

The FFLAS/FFPACK project has demonstrated the efficiency that is gained by wrapping cache-aware BLAS routines for efficient linear algebra over small finite fields [4, 5, 2]. The conversion between a modular representation of prime fields of any (small) characteristic and floating points can be performed via the homomorphism to the integers. For extension fields, the elements are naturally represented as polynomials over prime fields. In [3] it is proposed to transform these polynomials into a Q-adic representation where Q is an integer larger than the characteristic of the field. This transformation is called DQT for Discrete Q-adic Transform, it is a form of Kronecker substitution [7, §8.4]. With some care, in particular on the size of Q, it is possible to map the polynomial operations into the floating point arithmetic realization of this Q-adic representation and convert back using an inverse DQT.

In this work, we propose to use this fast polynomial arithmetic within machine words to compress matrices over very small finite fields. This is achieved by storing groups of $d+1$ entries of the matrix into one floating point number each, where d is a parameter to be maximized depending on the cardinality of the finite field and the the size of the matrices. First, we show in Section 2 how a dot product of vectors of size $d+1$ can be recovered from a single machine word multiplication. This extends to matrix multiplication by compressing both matrices first. Then we propose in Section 3 an alternative matrix multiplication using multiplication

[*]Laboratoire J. Kuntzmann, Université de Grenoble, umr CNRS 5224. BP 53X, 51, rue des Mathématiques, F38041 Grenoble, France. {Jean-Guillaume.Dumas,Laurent.Fousse}@imag.fr. This work is supported in part by the French *Agence Nationale pour la Recherche* (ANR Safescale).

[†]Algorithms Project, INRIA Rocquencourt, 78153 Le Chesnay. France. Bruno.Salvy@inria.fr. This work is supported in part by the French *Agence Nationale pour la Recherche* (ANR Gecko).

of a compressed word by a single residue. This operation also requires a simultaneous modular reduction, which is called REDQ in [3] where its efficient implementation is described.

In general, the prime field, the size of matrices and the available mantissa are given. This gives some constraints on the possible choices of Q and d. In both cases anyway, we show that these compression techniques represent a speed-up factor of up to the number $d+1$ of residues stored in the compressed format. We conclude in Section 5 with a comparison of the techniques.

2 Q-adic compression or Dot product via polynomial multiplication

2.1 Modular dot product via machine word multiplication

If $a = [a_0, \dots, a_d]$ and $b = [b_0, \dots, b_d]$ are two vectors with entries in $\mathbb{Z}/p\mathbb{Z}$, their dot product $\sum a_i b_i$ is the coefficient of degree d in the product of polynomials $a(X) = \sum_{i=0}^{d} a_{d-i} X^i$ and $b(X) = \sum_{i=0}^{d} b_i X^i$. The idea here, as in [3], is to replace X by an integer Q, usually a power of 2 in order to speed up conversions. Thus the vectors of residues a and b are stored respectively as $\bar{b} = \sum_{i=0}^{d} b_i Q^i$ and the *reverse* $\bar{a} = \sum_{i=0}^{d} a_{d-i} Q^i$.

For instance, for $d = 2$, the conversion is performed by the following compression:

```
double& init3( double& r, const double u, const double v, const double w) {
        // _dQ is a floating point storage of Q
        r=u; r*=_dQ; r+=v; r*=_dQ; return r+=w;
}
```

2.2 Compressed Matrix Multiplication

We first illustrate the idea for 2×2 matrices and $d = 1$. The product

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

is recovered from

$$\begin{bmatrix} Qa+b \\ Qc+d \end{bmatrix} \times \begin{bmatrix} e+Qg & f+Qh \end{bmatrix} = \begin{bmatrix} *+(ae+bg)Q+*Q^2 & *+(af+bh)Q+*Q^2 \\ *+(ce+dg)Q+*Q^2 & *+(cf+dh)Q+*Q^2 \end{bmatrix},$$

where the character $*$ denotes other coefficients.

In general, A is an $m \times k$ matrix to be multiplied by a $k \times n$ matrix B, the matrix A is first compressed into a $m \times \left\lceil \frac{k}{d+1} \right\rceil$ `CompressedRowMatrix`, CA, and B is transformed into a $\left\lceil \frac{k}{d+1} \right\rceil \times n$ `CompressedColumnMatrix`, CB. The compressed matrices are then multiplied and the result can be extracted from there. This is depicted on Fig. 1

In terms of number of arithmetic operations, the matrix multiplication $CA \times CB$ can save *a factor of* $d+1$ over the multiplication of $A \times B$ as shown on the 2×2 case above.

The computation has three stages: compression, multiplication and extraction of the result. The compression and extraction are less demanding in terms of asymptotic complexity, but can still be noticeable for moderate sizes. For this reason, compressed matrices are often reused and it is more informative to distinguish the three phases in an analysis. This is done in Section 5 (Table 2), where the actual matrix multiplication algorithm is also taken into account.

Partial compression Note that the last column of CA and the last row of B might not have $d+1$ elements if $d+1$ does not divide k. Thus one has to artificially append some zeroes to the converted values. On $\bar{b}$ this means just do nothing. On the reversed $\bar{a}$ this means multiplying by Q several times.

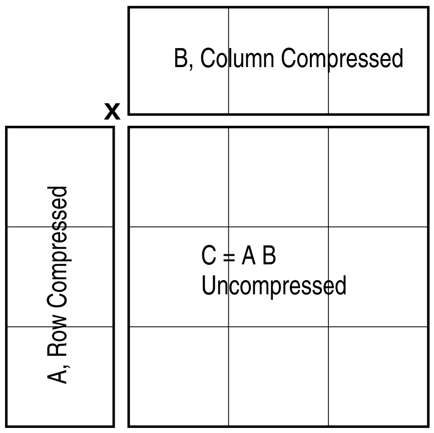


Figure 1: Compressed Matrix Multiplication (CMM)

2.3 Delayed reduction and lower bound on Q

For the results to be correct the inner dot product must not exceed Q. With a positive modular representation mod p (i.e. integers from 0 to $p-1$), this means that we demand that the inequality $(d+1)(p-1)^2 < Q$ holds. Moreover, it is possible to save more time by using delayed reductions on the intermediate results, i.e., accumulating several products $\bar{a}\bar{b}$ before any modular reduction. It is thus possible to perform matrix multiplications with common dimension k as long as:

$$\frac{k}{d+1}(d+1)(p-1)^2 = k(p-1)^2 < Q. \tag{1}$$

2.4 Available mantissa and upper bound on Q

If the product $\bar{a}\bar{b}$ is performed with floating point arithmetic we just need that the coefficient of degree d fits in the β bits of the mantissa. Writing $\bar{a}\bar{b} = c_H Q^d + c_L$, we see that this implies that c_H, *and only* c_H, must remain smaller than 2^β. It can then be recovered exactly by multiplication of $\bar{a}\bar{b}$ with the correctly precomputed and rounded inverse of Q^d as shown e.g., in [3, Lemma 2].

With delayed reduction this means that

$$\sum_{i=0}^{d} \frac{k}{d+1}(i+1)(p-1)^2 Q^{d-i} < 2^\beta.$$

Using Eq. (1) shows that this is ensured if

$$Q^{d+1} < 2^\beta. \tag{2}$$

Thus a single reduction has to be made at the end of the dot product as follows:

```
Element& init( Element& rem, const double dp) const {
        double r = dp;
        // Multiply by the inverse of Q^d with correct rounding
        r *= _inverseQto_d;
        // Now we just need the part less than Q=2^t
        unsigned long rl( static_cast<unsigned long>(r) );
        rl &= _QMINUSONE;
        // And we finally perform a single modular reduction
        rl %= _modulus;
        return rem = static_cast<Element>(rl);
}
```

Note that one can avoid the multiplication by the inverse of Q when Q is a power of 2, say 2^t: by adding Q^{2d+1} to the final result one is guaranteed that the $t(d+1)$ high bits represent exactly the $d+1$ high coefficients. On the one hand, the floating point multiplication is then replaced by an addition. On the other hand, this doubles the size of the dot product and thus reduces by a factor of $\sqrt[d+1]{2}$ the largest possible dot product size k.

2.5 Results

On Figure 2 we compare our compression algorithm to the numerical double floating point matrix multiplication `dgemm` of GotoBlas [8] and to the `fgemm` modular matrix multiplication of the FFLAS-LinBox library [4]. For the latter we show timings using `dgemm` and also `sgemm` over single floating points.

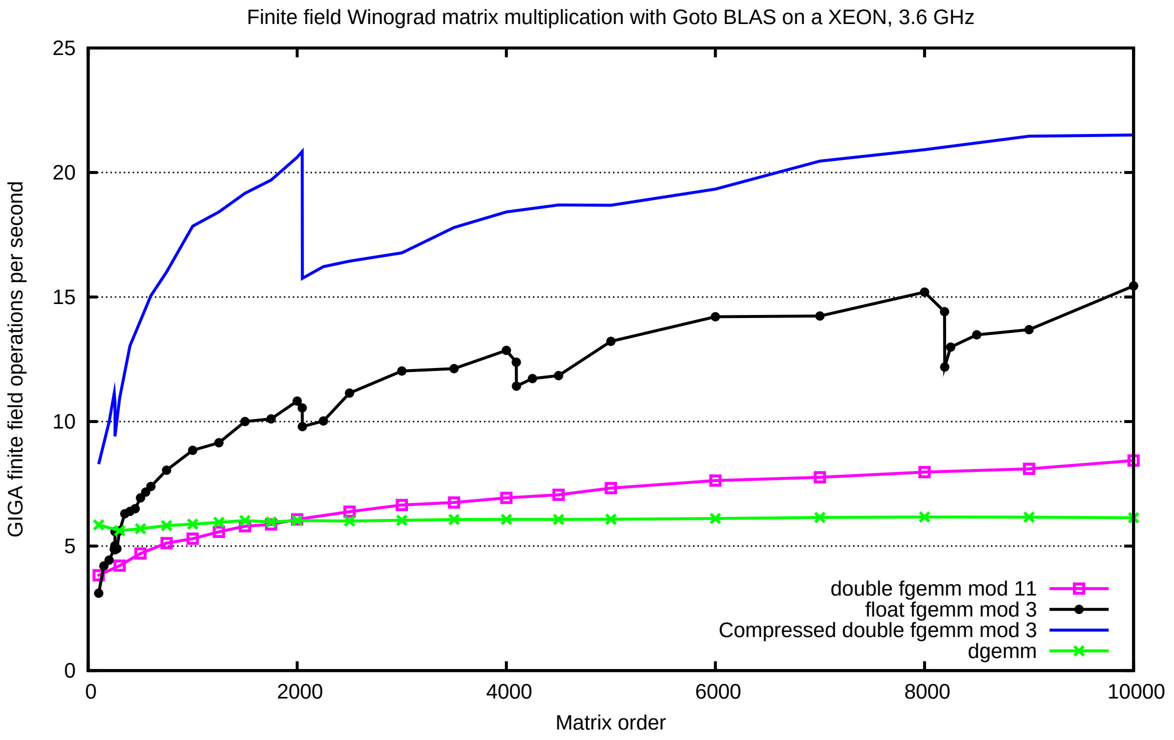


Figure 2: Compressed matrix multiplication compared with `dgemm` (the floating point double precision matrix multiplication of GotoBlas) and `fgemm` (the exact routine of FFLAS) with double or single precision.

This figure shows that the compression $(d+1)$ is very effective for small primes: the gain over the double floating point routine is quite close to d.

Observe that the curve of `fgemm` with underlying arithmetic on single floats oscillates and drops sometimes. Indeed, the matrix begins to be too large and modular reductions are now required between the recursive matrix multiplication steps. Then the floating point BLAS[1] routines are used only when the submatrices are small enough. One can see the subsequent increase in the number of classical arithmetic steps on the drops around 2048, 4096 and 8192.

[1]http://www.tacc.utexas.edu/resources/software/

Compression	2	3..4	5..8	8	7	6	5	4	3
Degree d	1	5	9	7	6	5	4	3	2
Q-adic	2^3	2^4	2^5	2^6	2^7	2^8	2^{10}	2^{13}	2^{17}
Dimensions	2	≤ 4	≤ 8	≤ 16	≤ 32	≤ 64	≤ 256	≤ 2048	≤ 32768

Table 1: Compression factors for different common matrix dimensions modulo 3, with 53 bits of mantissa and Q a power of 2.

On Table 1, we show the compression factors modulo 3, with Q a power of 2 to speed up conversions. For a dimension $n \leq 256$ the compression is at a factor of five and the time to perform a matrix multiplication is less than a hundredth of a second. Then from dimensions from 257 to 2048 one has a factor of 4 and the times are roughly 16 times the time of the four times smaller matrix. The next stage, from 2048 to 32768 is the one that shows on Figure 1.

Figure 2 shows the dramatic impact of the compression dropping from 4 to 3 between $n = 2048$ and $n = 2049$. It would be interesting to compare the multiplication of 3-compressed matrices of size 2049 with a decomposition of the same matrix into matrices of sizes 1024 and 1025, thus enabling 4-compression also for matrices larger than 2048, but with more modular reductions.

3 Right or Left Compressed Matrix Multiplication

Another way of performing compressed matrix multiplication is to multiply an uncompressed $m \times k$ matrix to the right by a row-compressed $k \times \frac{n}{d+1}$ matrix. We illustrate the idea on 2×2 matrices:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e + Qf \\ g + Qh \end{bmatrix} = \begin{bmatrix} (ae + bg) + Q(af + bh) \\ (ce + dg) + Q(cf + dh) \end{bmatrix}$$

The general case is depicted on Fig. 3, center. This is called Right Compressed Matrix Multiplication. Left Compressed Matrix Multiplication is obtained by transposition.

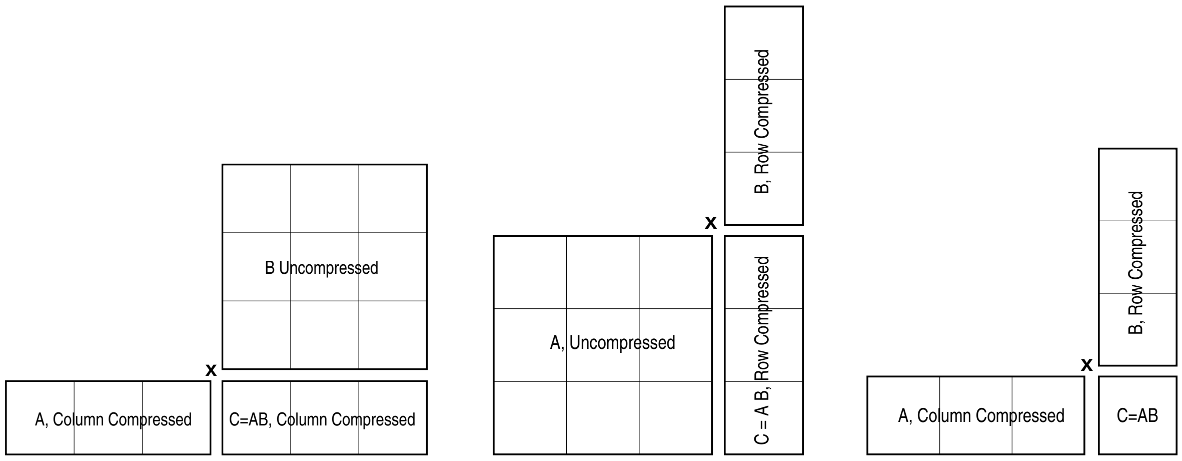


Figure 3: Left, Right and Full Compressions

Here also Q and d must satisfy Eqs. (1) and (2).

The major difference with the Compressed Matrix Multiplication lies in the reductions. Indeed, now one needs to reduce simultaneously the $d + 1$ coefficients of the polynomial in Q in order to get the results. This simultaneous reduction can be made by the REDQ algorithm of [3, Algorithm 2].

When working over compressed matrices CA and CB, a first step is to uncompress CA, which has to be taken into account when comparing methods. Thus the whole right compressed matrix multiplication is the following algorithm

$$A = \text{Uncompress}(CA); CC = A \times CB; \text{REDQ}(CC) \tag{3}$$

4 Full Compression

It is also possible to compress simultaneously both dimensions of the matrix product (see Fig. 3, right). This is achieved by using polynomial multiplication with two variables Q and Θ. Again, we start by an example in dimension 2:

$$\begin{bmatrix} a + Qc & b + Qd \end{bmatrix} \times \begin{bmatrix} e + \Theta f \\ g + \Theta h \end{bmatrix} = \begin{bmatrix} (ae + bg) + Q(ce + dg) + \Theta(af + bh) + Q\Theta(cf + dh) \end{bmatrix}$$

More generally, let d_q be the degree in Q and d_θ be the degree in Θ. Then, the dot product is:

$$\begin{aligned} a \cdot b &= [\sum_{i=0}^{d_q} a_{i0} Q^i, \ldots, \sum_{i=0}^{d_q} a_{in} Q^i] \times [\sum_{j=0}^{d_\theta} b_{0j} \Theta^j, \ldots, \sum_{j=0}^{d_\theta} b_{nj} \Theta^j], \\ &= \sum_{l=0}^{k} (\sum_{i=0}^{d_q} a_{il}) (\sum_{j=0}^{d_\theta} b_{lj}) Q^i \Theta^j = \sum_{i=0}^{d_q} \sum_{j=0}^{d_\theta} (\sum_{l=0}^{k} a_{il} b_{lj}) Q^i \Theta^j. \end{aligned}$$

In order to guarantee that all the coefficients can be recovered independently, Q must still satisfy Eq. (1) but then Θ must satisfy an additional constraint:

$$Q^{d_q+1} \leq \Theta \tag{4}$$

This imposes restrictions on d_q and d_θ:

$$Q^{(d_q+1)(d_\theta+1)} < 2^\beta \tag{5}$$

5 Comparison

In Table 2, we summarize the differences of the algorithms presented on Figures 1 and 3. As usual, the exponent ω denotes the exponent of matrix multiplication. Thus, $\omega = 3$ for the classical matrix multiplication, while $\omega < 3$ for faster matrix multiplications, as used in [6, §3.2]. For products of rectangular matrices, we use the classical technique of first decomposing the matrices into square blocks and then using fast matrix multiplication on those blocks.

Compression Factor The costs in Table 2 are expressed in terms of a *compression factor* e, that we define as

$$e := \left\lfloor \frac{\beta}{\log_2(Q)} \right\rfloor,$$

where, as above, β is the size of the mantissa and Q is the integer chosen according to Eqs. (1) and (2), except for Full Compression where the more constrained Eq. (5) is used.

Thus the degree of compression for the first three algorithms is just $d = e - 1$, while it becomes only $d = \sqrt{e} - 1$ for the full compression algorithm (with equal degrees $d_q = d_\theta = d$ for both variables Q and Θ).

Algorithm	Operations	Reductions	Conversions
CMM	$\mathcal{O}\left(mn\left(\frac{k}{e}\right)^{\omega-2}\right)$	$m \times n$ REDC	$\frac{1}{e}mn\ \mathrm{INIT}_e$
Right Comp.	$\mathcal{O}\left(mk\left(\frac{n}{e}\right)^{\omega-2}\right)$	$m \times \frac{n}{e}\ \mathrm{REDQ}_e$	$\frac{1}{e}mn\ \mathrm{EXTRACT}_e$
Left Comp.	$\mathcal{O}\left(nk\left(\frac{m}{e}\right)^{\omega-2}\right)$	$\frac{m}{e} \times n\ \mathrm{REDQ}_e$	$\frac{1}{e}mn\ \mathrm{EXTRACT}_e$
Full Comp.	$\mathcal{O}\left(k\left(\frac{mn}{e}\right)^{\frac{\omega-1}{2}}\right)$	$\frac{m}{\sqrt{e}} \times \frac{n}{\sqrt{e}}\ \mathrm{REDQ}_e$	$\frac{1}{e}mn\ \mathrm{INIT}_e$

Table 2: Number of arithmetic operations for the different algorithms

Analysis In terms of asymptotic complexity, the cost in number of arithmetic operations is dominated by that of the product (column Operations in the table), while reductions and conversions are linear in the dimensions. This is well reflected in practice. For example, with algorithm CMM on matrices of sizes $10,000 \times 10,000$ it took 92.75 seconds to perform the matrix multiplication modulo 3 and 0.25 seconds to convert the resulting matrix. This is less than 0.3%. For 250×250 matrices it takes less than 0.0028 seconds to perform the multiplication and roughly 0.00008 seconds for the conversions. There, the conversions account for 3% of the time.

In the case of rectangular matrices, the second column of Table 2 shows that one should choose the algorithm depending on the largest dimension: CMM if the common dimension k is the largest, Right Compression if n if the largest and Left Compression if m dominates. The gain in terms of arithmetic operations is $e^{\omega-2}$ for the first three variants and $e^{\frac{\omega-1}{2}}$ for full compression. This is not only of theoretical interest but also of practical value, since the compressed matrices are then less rectangular. This enables more locality for the matrix computations and usually results in better performance. Thus, even if $\omega = 3$, i.e., classical multiplication is used, these considerations point to a source of speed improvement.

The full compression algorithm seems to be the best candidate for locality and use of fast matrix multiplication; however the compression factor is an integer, depending on the flooring of either $\frac{\beta}{\log_2(Q)}$ or $\sqrt{\frac{\beta}{\log_2(Q)}}$. Thus there are matrix dimensions for which the compression factor of e.g., the right compression will be larger than the square of the compression factor of the full compression. There the right compression will have some advantage over the full compression.

If the matrices are square ($m = n = k$) or if $\omega = 3$, the products all become the same, with similar constants implied in the $O()$, so that apart from locality considerations, the difference between them lies in the time spent in reductions and conversions. Since the REDQ_e reduction is faster than e classical reductions [3], and since INIT_e and $\mathrm{EXTRACT}_e$ are roughly the same operations, the best algorithm would then be one of the Left, Right or Full compression. Further work would include implementing the Right or Full compression and comparing the actual timings of conversion overhead with that of algorithm CMM.

References

[1] Don Coppersmith. Solving linear equations over $GF(2)$: block Lanczos algorithm. *Linear Algebra and its Applications*, 192:33–60, October 1993.

[2] Jean-Guillaume Dumas. Efficient dot product over finite fields. In Victor G. Ganzha, Ernst W. Mayr, and Evgenii V. Vorozhtsov, editors, *Proceedings of the seventh International Workshop on Computer Algebra in Scientific Computing, Yalta, Ukraine*, pages 139–154. Technische Universität München, Germany, July 2004.

[3] Jean-Guillaume Dumas. Q-adic transform revisited. In David Jeffrey, editor, *Proceedings of the 2008 International Symposium on Symbolic and Algebraic Computation, Hagenberg, Austria.* ACM Press, New York, July 2008.

[4] Jean-Guillaume Dumas, Thierry Gautier, and Clément Pernet. Finite field linear algebra subroutines. In Teo Mora, editor, *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation, Lille, France*, pages 63–74. ACM Press, New York, July 2002.

[5] Jean-Guillaume Dumas, Pascal Giorgi, and Clément Pernet. FFPACK: Finite field linear algebra package. In Jaime Gutierrez, editor, *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation, Santander, Spain*, pages 119–126. ACM Press, New York, July 2004.

[6] Jean-Guillaume Dumas, Pascal Giorgi, and Clément Pernet. Dense linear algebra over prime fields. *ACM Transactions on Mathematical Software*, 2008. to appear.

[7] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 1999.

[8] Kazushige Goto and Robert van de Geijn. On reducing TLB misses in matrix multiplication. Technical Report TR-2002-55, University of Texas, November 2002. FLAME working note #9.

[9] Chao H. Huang and Fred J. Taylor. A memory compression scheme for modular arithmetic. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 27(6):608–611, December 1979.

[10] Erich Kaltofen and Austin Lobo. Distributed matrix-free solution of large sparse linear systems over finite fields. In A.M. Tentner, editor, *Proceedings of High Performance Computing 1996, San Diego, California*. Society for Computer Simulation, Simulation Councils, Inc., April 1996.

[11] John P. May, David Saunders, and Zhendong Wan. Efficient matrix rank computation with application to the study of strongly regular graphs. In Christopher W. Brown, editor, *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation, Waterloo, Canada*, pages 277–284. ACM Press, New York, July 29 – August 1 2007.

[12] Guobiao Weng, Weisheng Qiu, Zeying Wang, and Qing Xiang. Pseudo-Paley graphs and skew Hadamard difference sets from presemifields. *Designs, Codes and Cryptography*, 44(1-3):49–62, 2007.

Black Box Matrix Computations: Two Improvements

Wayne Eberly
Department of Computer Science, University of Calgary

Abstract

Two enhancements for black box matrix computations are described.

1 Introduction

Inspired by and working from Wiedemann's algorithm [8], as well as applications of an algorithm of Lanczos [5] to number-theoretic computations (as described, for example, by LaMacchia and Odlyzko [4]), research in black box matrix computations has been in progress for approximately the last decade. The LinBox library [2] is a notable result of this work; see `http://www.linalg.org` for additional documentation as well as the current version of this library.

Two extensions or enhancements of black-box matrix computations, that might be useful additions to LinBox or any similar library, are described below. The first allows alternative techniques that are considerably faster for various structured matrix calculations to be included in a library. It is argued below that these techniques can be selected automatically, when appropriate, without requiring user expertise. The second extends scalar algorithms, by adding an additional step at the point when current algorithms terminate, in order to make more effective use of the information that has been computed. This would reduce the expected number of matrix-vector multiplications needed to solve nonsingular systems of linear equations using these techniques as well as the expected number of matrix-vector multiplications needed to compute the minimal polynomial of a matrix.

This is in work in progress. Certainly a considerable amount of experimental work and tuning of algorithms is required before these modifications are incorporated in any library that is available for general use. There is also additional analytical work to be done; questions arising from the current results are mentioned at the end of the sections that follow.

2 Improved Support for Structured Computations

While less general than the algorithms presently included in the LinBox library, "superfast" algorithms for various structured matrix computations are asymptotically faster. Bitmead and Anderson [1] contributed one of the first such algorithms, namely, an algorithm that could be used to solve nonsingular systems of linear equations when the input matrix is "Toeplitz-like." The more recent text of Pan [6], and the references therein, document several other classes of structured matrices along with superfast algorithms for various structured matrix computations.

A library that supports and effectively combines both types of algorithms would be more useful than any library that supports only one of the above families of algorithms. Of course, one could simply include implementations of superfast algorithms in LinBox and allow a user to choose from the algorithms that are available.

However this is only effective if a library user has considerable expertise as well as knowledge about the problem that is to be solved. Under other circumstances it would be helpful if the system could do the work of intelligently choosing from these algorithms. In particular it would be useful for the system to be able to detect various kinds of structured matrices automatically.

This motivates the following.

Lemma 2.1. *Let $C \in \mathsf{F}^{n\times n}$ be a matrix and let m be a positive integer such that $0 \le m < n$. Let S be a finite subset of F with size $s > m$.*

It is possible to decide whether the rank of C is less than m, compute the rank ℓ of C if this is less than m, and generate matrices $G, H \in \mathsf{F}^{n\times\ell}$ such that $C = G\cdot H^T$ (in this case) using a Monte Carlo algorithm. This algorithm selects at most $\min(\mathrm{rank}(C), m) + 1$ vectors uniformly and independently from $S^{n\times 1}$ and is otherwise deterministic. The algorithm also computes the product of C and at most $\min(\mathrm{rank}(C), m)+1$ vectors, the product of C^T and at most $\min(\mathrm{rank}(C), m)$ vectors, and performs $O(nm^2)$ additional operations over F. The algorithm fails (by producing an estimate of the rank that is too small) with probability at most $1-\frac{m}{s}$; the output provided by the algorithm is correct in all other cases.

Proof. Consider an algorithm that begins by generating vectors $v_1, v_2, \ldots, v_h$ uniformly and independently from $S^{n\times 1}$, such that either $h = m+1$ and $Cv_1, Cv_2, \ldots, Cv_h$ are linearly independent, or $h \le m+1$, $Cv_1, Cv_2, \ldots, Cv_{h-1}$ are linearly independent, and Cv_h is a linear combination of $Cv_1, Cv_2, \ldots, Cv_{h-1}$. In the former case the algorithm reports that the rank of C is greater than m and stops. In the latter case the algorithm should set ℓ to be $h-1$ and it should set G to be the matrix obtained by using Gaussian Elimination to triangularize the matrix

$$\widehat{G} = \begin{bmatrix} Cv_1 & Cv_2 & \ldots & Cv_\ell \end{bmatrix}$$

whose columns have been generated above. In other words $G = \widehat{G}X$ for an invertible matrix $X \in \mathsf{F}^{\ell\times 1}$ (so that the columns of G form a basis for the column space of C, if ℓ is indeed equal to the rank of C) and so that $P\cdot G$ is lower triangular for some $n\times n$ permutation matrix P. It is clear from the above conditions that $\ell \le \min(\mathrm{rank}(C), m)$, so that this step requires the generation of at most $\min(\mathrm{rank}(C), m)+1$ vectors uniformly and independently from $S^{n\times 1}$ and at most this number of matrix-vector products by C. The matrix G is generated one column at a time, essentially by applying Gaussian elimination to an $n\times\ell$ matrix, so that the number of additional operations required for this stage of the algorithm is indeed in $O(nm^2)$.

To continue, suppose that the rank of C is less than or equal to m and that ℓ has correctly been set to be the rank of C. Note that (as part of the application of Gaussian Elimination that has been described above) as set of ℓ rows included in an invertible $\ell\times\ell$ submatrix of G, has now been identified. In other words, a matrix $K \in \mathsf{F}^{n\times\ell}$ whose columns each have exactly one nonzero entry (namely, 1) is available such that the matrix

$$B = G^T\cdot K \in \mathsf{F}^{\ell\times\ell} \tag{1}$$

is invertible. Let

$$L = C^T K \in \mathsf{F}^{n\times\ell}, \tag{2}$$

noting that the entries of L can be computed using ℓ multiplications of C^T by vectors (namely, the columns of K). Set

$$H = L\cdot B^{-1} \in \mathsf{F}^{n\times\ell}. \tag{3}$$

To see that G and H have the desired properties if ℓ is equal to the rank of C, note that

$$C = G\cdot \widehat{H}^T \tag{4}$$

for some matrix $\widehat{H} \in \mathsf{F}^{n\times\ell}$ because the columns of G span the column space of C. However, it now follows that

$$\begin{aligned}
H &= L\cdot B^{-1} && \text{(by the choice of } H \text{ at line (3), above)}\\
&= C^T\cdot K\cdot B^{-1} && \text{(by the choice of } L \text{ at line (2))}\\
&= \widehat{H}\cdot G^T\cdot K\cdot B^{-1} && \text{(using the decomposition of } C \text{ at line (4))}\\
&= \widehat{H}\cdot B\cdot B^{-1} = \widehat{H} && \text{(since } B = G^T\cdot K\text{, as shown at line (1)).}
\end{aligned}$$

It should be clear that the number of operations used by this last stage of the algorithm is within the bounds given in the statement of the lemma. It therefore remains only to characterize the failure of this algorithm and bound its probability. Since the matrix G always has full rank and has columns in the column space of C the algorithm can only fail by reporting a value for the rank that is too small (producing matrices $G, H \in \mathsf{F}^{n\times \ell}$ whose product is different from C at the same time). In order to bound the likelihood of this, let $r = \min(m, \operatorname{rank}(C))$ and consider a slightly different situation — in particular, suppose that r vectors $v_1, v_2, \dots v_r$ are initially chosen uniformly and independently from $S^{n\times 1}$ and that these vectors are subsequently used (if needed) by the algorithm as the first r vectors that are to be randomly selected. Note, in this case, that if

$$\widehat{G}_r = \begin{bmatrix} Cv_1 & Cv_2 & \dots & Cv_r \end{bmatrix} \in \mathsf{F}^{n\times r}$$

then the algorithm only fails if $\widehat{G}_r$ is rank-deficient. Now, there certainly does exist a set of vectors $w_1, w_2, \dots, w_r \in \mathsf{F}^{n\times 1}$ such that the matrix

$$\overline{G}_r = \begin{bmatrix} Cw_1 & Cw_2 & \dots Cw_r \end{bmatrix} \in \mathsf{F}^{n\times r}$$

has full rank, so that there is a nonsingular $r \times r$ submatrix of $\overline{G}_r$ corresponding to some choice of r rows. The determinant of this submatrix is then nonzero and a polynomial function of the entries of the vectors $w_1, w_2, \dots, w_r$ with total degree $r \leq m$. The Schwartz-Zippel lemma [7, 9] can now be applied to establish that $\widehat{G}$ is rank-deficient with probability at most $\frac{m}{s}$, establishing the bound on the probability of failure given in the statement of the claim. □

The algorithm described here can be implemented to use the storage space needed to store a sequence of at most m integers between 1 and n (indicating the position of nonzero entries in the columns of the matrix L) along with the space needed to store $O(nm^2)$ elements of the field F. Note, however, that the matrix H can be constructed by computing one column of L at a time, and using each column with the corresponding row of B^{-1} to produce a matrix with rank one whose entries should be added to a previous estimate; this process can be carried out using the storage space that will eventually be used to store the output matrix H along with $O(n + m^2)$ additional storage locations. Indeed, the algorithm can be implemented in such a way that the only storage space used is that which is eventually used to store the output, along with storage needed to represent $O(n + m^2)$ entries of F and $O(m)$ integers between 1 and m.

A similar algorithm can be used to solve the above problem reliably when F is a small finite field. In order to ensure that the probability of failure is at most ϵ for a given real number $\epsilon > 0$, it suffices to consider up to $1 + \lceil \log_{|\mathsf{F}|}(1/\epsilon) \rceil$ candidates for each vector v_i (when searching for a vector such that Cv_i is not a linear combination of $Cv_1, Cv_2, \dots, Cv_{i-1}$) during the first phase of the algorithm. It can be shown that the expected number of additional vectors that must be generated and considered (and multiplied by C) is then in $O(\log_{|\mathsf{F}|}(1/\epsilon))$. The expected amount of additional operations over F to be performed is in $O(nm^2 \log_{|\mathsf{F}|}(1/\epsilon))$.

Now let us recall that a matrix $A \in \mathsf{F}^{n\times n}$ (with entries in a field F) is *Toeplitz-like* if $F(A)$ has small rank, where

$$F(A) = A - ZAZ^T, \tag{5}$$

and where $Z \in \mathsf{F}^{n\times n}$ is the displacement matrix with ones on the band below the diagonal and zeroes everywhere else. The rank of $F(A)$ is called the *displacement rank* of A (and if A is a Toeplitz matrix then $F(A) \leq 2$), and vectors $g_1, g_2, \dots, g_k \in \mathsf{F}^{n\times 1}$ and $h_1, h_2, \dots, h_k \in \mathsf{F}^{n\times 1}$ are *displacement generators* for A if

$$F(A) = GH^T,$$

where

$$G = \begin{bmatrix} g_1 & g_2 & \dots & g_k \end{bmatrix} \in \mathsf{F}^{n\times k} \quad \text{and} \quad H = \begin{bmatrix} h_1 & h_2 & \dots & h_k \end{bmatrix} \in \mathsf{F}^{n\times k}.$$

Consequently the following is a direct corollary of the result that has been established above.

Theorem 2.2. *Let $A \in \mathsf{F}^{n\times n}$, let m be a positive integer such that $0 \le m \le n$, and let S be a finite subset of F with size $s > m$. Then it is possible to determine whether A is Toeplitz-like with displacement rank at most m. Furthermore, if this is the case, then the displacement rank and a set of displacement generators for A can also be computed. These computations can be carried out using a Monte Carlo algorithm that uses at $2m + 2$ multiplications by A, at most $2m$ multiplications of vectors by A^T, $O(nm^2)$ additional operations in F, and which fails with probability at most $\frac{m}{s}$ if the algorithm selects vectors uniformly and independently from $S^{n\times 1}$.*

Proof. The computation can be performed by applying the algorithm described in the proof of Lemma 2.1 above, using the matrix $C = F(A) = A - ZAZ^T$ as the input matrix. A matrix-vector multiplication by C can be implemented using a pair of matrix-vector multiplications by A, along with $O(n)$ additional operations in F. Thus the bounds on running time and the probability of failure given above follow immediately from Lemma 2.1. □

If $m \in o(\sqrt{n})$ then then the total cost to determine whether A has displacement rank less than or equal to m is dominated by the cost to solve a system of linear equations with coefficient matrix A using any of the (Lanczos- or Wiedemann-based) algorithms currently included in the LinBox library. Furthermore, if efficient matrix-vector multiplication by A or A^T is supported, then the cost of the above computation is comparable to that needed to apply a "superfast" algorithm to the matrix A.

These techniques can also be applied to identify matrices that have a "Hankel-like" or "Toeplitz+Hankel"-like structure. As noted above, Pan [6] provides additional information about these classes of structured matrices.

Several other classes of structured matrices — notably including "Cauchy-like" and "Vandermonde-like" matrices — have also been studied, and "superfast" algorithms for these classes of matrices have also been identified. In all of these cases one can determine whether a given matrix $A \in \mathsf{F}^{n\times n}$ is a structured matrix of the given type by determining whether the matrix

$$L(A) = A - SAT$$

has low rank for a certain pair of matrices S and T (compare this with the expression at line (5), above). If one is checking to see whether the matrix A is Cauchy-like or Vandermonde-like then either S or T is a diagonal matrix with some set of entries $s_1, s_2, \ldots, s_n \in \mathsf{F}$ on its diagonal. If these diagonal entries are supplied ahead of time then one can determine whether a given matrix A is Cauchy-like or Vandermonde-like — with respect to these diagonal entries — using a process similar to the one that has been described above for the detection of Toeplitz-like matrices. However, it is not clear that this is the case if the entries on the diagonal of the matrix S (or T) are *not* supplied ahead of time.

Question 2.3. Is it possible to determine, efficiently, whether a given matrix $A \in \mathsf{F}^{n\times n}$ is either Cauchy-like or Vandermonde-like, if no other information is supplied?

It is not clear to the author that the answer to the above question is known if complete information about A (notably including the entries of this matrix) are available. Consequently this question may have more to do with the theory of structured matrices with displacement operators than with black box linear algebra.

3 A More Efficient Recovery of the Minimal Polynomial

The LinBox algorithm currently includes a variety of block algorithms as well as older "scalar" Krylov-based algorithms. Block algorithms are not always applicable. For example, it is not clear that block algorithms can be used to compute the minimal polynomial of a given matrix — using a block algorithm with block factor k, one generally encounters a polynomial that is a divisor of the product of the first k invariant factors rather than the minimal polynomial. Furthermore a distributed implementation seems to be necessary in order for a block algorithm to be faster (in the worst case) than a scalar algorithm in those situations where scalar algorithms can reliably be used. Work to improve scalar algorithms is therefore still of some interest.

With that in mind, recall that if $A \in \mathsf{F}^{n\times n}$ then the minimal polynomial of A, minpol(A), is the monic polynomial $f \in \mathsf{F}[x]$ with least degree such that $f(A) = 0$. Similarly if $A \in \mathsf{F}^{n\times n}$ and $v \in \mathsf{F}^{n\times 1}$ then the minimal polynomial of A and v, minpol(A, v) is the monic polynomial $f \in \mathsf{F}[x]$ with least degree such that $f(A)v = 0$. Finally, if $u, v \in \mathsf{F}^{n\times 1}$ then the minimal polynomial of A, u and v, minpol(A, u, v), is the monic polynomial $f \in \mathsf{F}[x]$ with least degree such that $u^T A^i f(A)v = 0$ for every integer $i \geq 0$, that is, the monic polynomial f with least degree that annihilates the linearly recurrent sequence

$$u^T v, u^T Av, u^T A^2 v, u^T A^3 v, \dots$$

If A, u, and v are as above them minpol(A, u, v) is always a divisor of minpol(A, v), and minpol(A, v) is always a divisor of minpol(A).

Theorem 3.1. *Let $\mathsf{F} = \mathsf{F}_q$ be a finite field with size q and let $A \in \mathsf{F}^{n\times n}$.*

(a) *If the vector v is chosen uniformly and randomly from $\mathsf{F}^{n\times 1}$ then the expected value of the degree of the polynomial*

$$\frac{\mathrm{minpol}(A)}{\mathrm{minpol}(A, v)}$$

is at most $\log_q n + 9 + \frac{8}{n}$.

(b) *If vectors u and v are chosen uniformly and independently from $\mathsf{F}^{n\times 1}$ then the expected value of the degree of the polynomial*

$$\frac{\mathrm{minpol}(A)}{\mathrm{lcm}(\mathrm{minpol}(A^T, u), \mathrm{minpol}(A, v))}$$

is at most $\frac{16}{9} < 2$.

Proof. For vectors $u, v \in \mathsf{F}^{n\times 1}$, suppose that

$$f_v = \frac{\mathrm{minpol}(A)}{\mathrm{minpol}(A, v)} \quad \text{and} \quad g_{u,v} = \frac{\mathrm{minpol}(A)}{\mathrm{lcm}(\mathrm{minpol}(A^T, u), \mathrm{minpol}(A, v))}.$$

Suppose that ϕ is an irreducible polynomial with degree $d > 0$ in $\mathsf{F}[x]$. Then an application of Wiedemann's probabilistic analysis [8, Section VI] can be used to establish that if v is chosen uniformly from $\mathsf{F}^{n\times 1}$ and i is a positive integer then

$$\mathrm{Prob}[f_v \text{ is divisible by } \phi^i] \leq q^{-id}. \tag{6}$$

Similarly, if u and v are chosen uniformly and independently from $\mathsf{F}^{n\times 1}$ then

$$\mathrm{Prob}[g_{u,v} \text{ is divisible by } \phi^i] \leq q^{-2id}. \tag{7}$$

Let $\lambda_{\phi,v}$ be the degree of the greatest common divisor of ϕ^n and f_v. It follows from the inequality at line (6) that if v is chosen uniformly from $\mathsf{F}^{n\times 1}$ then

$$\begin{aligned} \mathrm{E}[\lambda_{\phi,v}] &= \sum_{i\geq 1} q^{-id} id \\ &= dq^{-d} + \sum_{i\geq 2} q^{-id} id \\ &\leq dq^{-d} + 2d\tfrac{q^{-2d}}{(1-q^{-d})^2} \\ &\leq dq^{-d} + 8dq^{-2d}, \end{aligned} \tag{8}$$

since $q \geq 2$ and $d \geq 1$. Similarly, if $\mu_{\phi,u,v}$ is the degree of the greatest common divisor of ϕ^n and $g_{u,v}$, when u and v are chosen uniformly and independently from $\mathsf{F}^{n\times 1}$, then it follows the inequality at line (7) that

$$\mathrm{E}[\mu_{\phi,u,v}] \leq \sum_{i\geq 1} iq^{-2id} = \tfrac{q^{-2d}}{(1-q^{-2d})^2} \leq \tfrac{16}{9} q^{-2d}. \tag{9}$$

To continue, let $\psi_d \in \mathsf{F}[x]$ be the product of all of the monic irreducible polynomials with degree d in $\mathsf{F}[x]$. Recall that there are at most q^d/d such polynomials in $\mathsf{F}[x] = \mathsf{F}_q[x]$.

Let $\widehat{\lambda}_{d,v}$ be the degree of the greatest common divisor of ψ_d^n and f_v. Then it follows by the inequality at line (8), and using linearity of expectation, that if v is chosen uniformly from $\mathsf{F}^{n\times 1}$ then

$$\mathrm{E}[\widehat{\lambda}_{d,v}] \leq \tfrac{q^d}{d}\left(dq^{-d} + 8dq^{-2d}\right) = 1 + 8q^{-d}. \tag{10}$$

Similarly, if $\widehat{\nu}_{d,u,v}$ is the degree of the greatest common divisor of ψ_d^n and $g_{u,v}$ and u and v are chosen uniformly and independently from $\mathsf{F}^{n\times 1}$ then it follows by this argument and the inequality at line (9) that

$$\mathrm{E}[\widehat{\mu}_{d,u,v}] \leq \tfrac{q^d}{d}\left(\tfrac{16}{9}q^{-2d}\right) = \tfrac{16}{9d}q^{-d} \leq \tfrac{16}{9}q^{-d}. \tag{11}$$

Part (b) of the claim now follows using linearity of expectation and the inequality at line (11), above. In order to complete the proof of part (a), let $\kappa \in \mathsf{F}[x]$ be the product of all monic irreducible polynomials with degree greater than or equal to $\log_q n$ in $\mathsf{F}[x]$ and that divide the minimal polynomial of A. Then

$$\kappa = \prod_{i=1}^{m} \phi_i$$

for distinct monic irreducible polynomials $\phi_1, \phi_2, \ldots, \phi_m$. Let d_i be the degree of ϕ_i for $1 \leq i \leq m$. Since κ divides the minimal polynomial of A it is clear that $d_1 + d_2 + \cdots + d_m \leq n$. Let ν_v be the degree of the greatest common divisor of κ^n and f_v. Then it follows by linearity of expectation and the inequality at line (8) that

$$\begin{aligned}
\mathrm{E}[\nu_v] &= \sum_{i=1}^{m} \mathrm{E}[\lambda_{\phi_i,v}] \\
&\leq \sum_{i=1}^{m} d_i q^{-d_i} + 8d_i q^{-2d_i} \\
&\leq \sum_{i=1}^{m} \tfrac{d_i}{n} + 8\tfrac{d_i}{n^2} \\
&\leq 1 + \tfrac{8}{n}.
\end{aligned} \tag{12}$$

The inequality at the penultimate line, above, follows from the fact that $d_i \geq \log_q n$ for all i. The inequality at the last line follows from the fact that $d_1 + d_2 + \cdots + d_m \leq n$. Finally, notice that if v is chosen uniformly from $\mathsf{F}^{n\times 1}$ then it follows, by linearity of expectation, that the expected value of the degree of f_v is at most

$$\mathrm{E}[\widehat{\lambda}_{1,v}] + \mathrm{E}[\widehat{\lambda}_{2,v}] + \cdots + \mathrm{E}[\widehat{\lambda}_{\lfloor \log_q n \rfloor,v}] + \mathrm{E}[\nu_v].$$

Part (a) of the claim now follows using the inequalities given at lines (10) and (12), above. $\square$

A similar result is also of use: If $A \in \mathsf{F}^{n\times n} = \mathsf{F}_q^{n\times n}$, v is a vector in $\mathsf{F}^{n\times 1}$, and the vector u is chosen uniformly from $\mathsf{F}^{n\times 1}$, then the expected value of the degree of the polynomial $\frac{\mathrm{minpol}(A,v)}{\mathrm{minpol}(A,u,v)}$ is at most $\log_q n + 9 + \frac{8}{n}$ as well. The proof of this is virtually identical to the proof of part (a) of the above claim.

Now suppose that $A \in \mathsf{F}^{n\times n} = \mathsf{F}_q^{n\times n}$ is nonsingular and consider the application of a scalar Lanczos algorithm to solve the system of linear equations $Ax = b$. This computation proceeds by uniformly and randomly selecting a vector $u \in \mathsf{F}^{n\times 1}$ and attempting to construct dual orthogonal bases for the Krylov spaces

$$Ab, A^2b, A^3b, \ldots \qquad \text{and} \qquad u, (A^T)u, (A^T)^2u, \ldots$$

An estimate $\widehat{x}$ for the solution of the system $Ax = b$ is constructed along the way.

Let $f = \text{minpol}(A, Ab)$ and let $g = \text{minpol}(A, u, Ab)$. Let d_f and d_g be the degrees of f and g, respectively. Then $d_f \geq d_g$, since g is always a divisor of f. Since u is chosen uniformly from $\mathsf{F}^{n\times 1}$ it follows by the above analysis that the expected value of $d_f - d_g$ is at most $\log_q n + 9 + \frac{8}{n}$. Notice as well that, since g divides f,

$$Ab, A^2b, A^3b, \dots, A^{d_g}b, g(A)Ab, g(A)A^2b, \dots, g(A)A^{d_f-d_g}b \tag{13}$$

is a basis for the Krylov space generated by A and Ab, while

$$b, Ab, A^2b, \dots, A^{d_g-1}b, g(A)b, g(A)Ab, \dots, g(A)A^{d_f-d_g-1}b \tag{14}$$

is a basis for the Krylov space generated by A and b. Recall that the estimate $\widehat{x}$ is generated by using an orthogonalization process (using the dual bases being constructed). Consequently, when the standard Lanczos process terminates, each of the following properties holds.

- $A\widehat{x}-b = t_0g(A)Ab+t_1g(A)A^2b+\cdots+t_{d_f-d_g-1}g(A)A^{d_f-d_g}b$ for unknown values $t_0, t_1, \dots, t_{d_f-d_g-1} \in \mathsf{F}$.
- Consequently $\widehat{x} - t_0g(A)b - t_1g(A)Ab - \cdots - t_{d_f-d_g-1}g(A)A^{d_f-d_g-1}b$ is a solution for the system for the given system, for the same sequence of values $t_0, t_1, \dots, t_{d_f-d_g-1}$.
- The vectors $g(A)b$ and $g(A)Ab$ are available. Indeed, if $f \neq g$, so that the second of these vectors is nonzero, then this vector is the last vector in the Krylov space for A and Ab that is being considered when the algorithm terminates. The vector $g(A)b$ is also being stored, at this point in the computation, in order to try to update the estimate $\widehat{x}$.

Consider the following continuation of the algorithm:

1. Performing additional matrix-vector multiplications by A as needed, and applying an elimination process as needed, generate the vectors

$$g(A)Ab, g(A)A^2b, g(A)A^3b, \dots, g(A)A^kb,$$

stopping either because the above vectors are linearly independent and $k + d_g = n$, in which case $d_f = n$ and this is a basis for the Krylov space for A and $g(A)b$, or because the first $k-1$ of these vectors are linearly independent but $g(A)A^kb$ is a linear combination of the vectors preceding it. In this second case $d_f = d_g + k - 1$ and the first $k-1$ of the above vectors forms a basis for the Krylov space for A and $g(A)Ab$.

2. Set H to be the matrix whose columns are the entries of the Krylov space for A and $g(A)Ab$ obtained in the first step, above. Solve the system of linear equations

$$H\vec{t} = A\widehat{x} - b.$$

3. Suppose $\vec{t} = [t_0, t_1, \dots, t_\ell]^T$ (where $\ell+1$ is the dimension of the Krylov space for A and $g(A)b$). Return the vector

$$x = \widehat{x} - t_0g(A)b - t_1g(A)Ab - \cdots - t_\ell g(A)A^\ell b$$

as the solution for the given system of linear equations.

Using the above ideas, and previously established bounds on lookahead for computations over small finite fields [3], it should be possible to implement a randomized Lanczos algorithm that can be used to solve a system $Ax = b$ that fails with small probability (that is, at most n^{-c} for a user-supplied constant c) using storage space in $O(n \log n)$, using $O(n^2)$ additional arithmetic operations over the field F, and where the number of matrix-vectors multiplications by A or A^T is at most $2n$ in the *worst* (as well as average) case.

Finally, suppose that we wish to compute the minimal polynomial of a given matrix $A \in \mathsf{F}^{n\times n} = \mathsf{F}_q^{n\times n}$. Consider the use of a hybrid Lanzos-Wiedemann algorithm. That is, a scalar Lanczos algorithm is used, with randomly and independently selected vectors $u, v \in \mathsf{F}^{n\times 1}$, such that for each vector $\widehat{u}$ that is considered in the Krylov space for A^T and u (respectively, for each vector $\widehat{v}$ in the Krylov space for A and v), one also computes the coefficients of a polynomial $h \in \mathsf{F}[x]$ such that $\widehat{u} = h(A^T)u$ (respectively, such that $\widehat{v} = h(A)v$).

Let $f_u = \text{minpol}(A^T, u)$, $g_v = \text{minpol}(A, v)$, and $h_{u,v} = \text{minpol}(A, u, v)$. Let d_f, d_g and d_h be the degrees of f_u, g_v and $h_{u,v}$. Then $h_{u,v}$ divides each of f_u and g_v so that $d_h \le d_f$ and $d_h \le d_g$. Furthermore, since the vectors u and v were selected independently, it follows by the above analysis that the expected values of $d_f - d_h$ and $d_g - d_h$ are both at most $\log_q n + 9 + \frac{8}{n}$. A pair of elimination-based continuations of the process can then be performed to recover both f_u and g_v. The least common multiple of these polynomials can subsequently be used as an estimate for the minimal polynomial of A. Indeed, as noted by Wiedemann [8, page 61], this is equal to the minimal polynomial of A with probability at least 0.3. As noted above, the expected difference in degree between the minimal polynomial and this estimate is at most $\frac{16}{9}$.

Simple elimination-based phases can be used to refine this estimate as well. Suppose, in particular, that a polynomial $h \in \mathsf{F}[x]$ has currently been obtained as an estimate of the minimal polynomial. In order to continue the process one should choose a vector $z \in \mathsf{F}^{n \times 1}$ uniformly and independently of all previously chosen vectors. One should then compute $h(A)z$ using $\deg(h)$ multiplications of A by vectors and using $O(n\deg(h))$ additional operations in F. The polynomial $\widehat{h} = \text{minpol}(A, h(A)z)$ can then be generated using an elimination-based phase as described above, and the product of h and $\widehat{h}$ can then be used as an improved estimate for the minimal polynomial of A.

An application of Wiedemann's probabilistic analysis confirms that if only a single additional vector z is considered (so that the number of matrix-vector multiplications is at most $3n$) then the resulting polynomial is the minimal polynomial of A with probability at least $\frac{2}{3}$. In general, the number of matrix-vector multiplications needed to bound the failure below any given (reasonable) threshold is approximately one-half of the number required to achieve the same error bound if Wiedemann's algorithm was used instead.

Question 3.2. Can similar ideas be used to improve the performance of block algorithms for computations over small fields — in particular, when the given coefficient matrix has a small number of nilpotent blocks and a large number of invariant factors?

References

[1] R. R. Bitmead and B. D. O. Anderson. Asymptotically fast solution of Toeplitz and related systems of linear equations. *Linear Algebra and Its Applications*, 34:103–116, 1980.

[2] J.-G. Dumas, T. Gautier, M. Giesbrecht, P. Giorgi, B. Hovinen, E. Kaltofen, B. D. Saunders, W. J. Turner, and G. Villard. Linbox: A generic library for exact linear algebra. In *Proceedings, First International Congress of Mathematical Software*, pages 40–50, 2002.

[3] W. Eberly. Early termination over small fields. In *Proceedings, 2003 International Symposium on Symbolic and Algebraic Computation (ISSAC 2003)*, pages 80–87, 2003.

[4] B. A. LaMacchia and A. M. Odlyzko. Solving large sparse systems over finite fields. In *Advances in Cryptology: CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 109–133, 1991.

[5] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of Research, National Bureau of Standards*, 45:255–282, 1950.

[6] V. Y. Pan. *Structured Matrices and Polynomials: Unified Superfast Algorithms.* Birkhäuser, 2001.

[7] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27:701–717, 1980.

[8] D. H. Wiedemann. Solving sparse linear systems over finite fields. *IEEE Transactions on Information Theory*, 32:54–62, 1986.

[9] R. Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings, EUROSAM '79*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer-Verlag, 1979.

Computing Popov Form of General Ore Polynomial Matrices

Patrick Davies Howard Cheng
Department of Mathematics and Computer Science
University of Lethbridge, Canada

George Labahn
David R. Cheriton School of Computer Science
University of Waterloo, Canada

Abstract

The computation of the Popov form of Ore polynomial matrices is formulated as a problem of computing the left nullspace of such matrices. While this technique is already known for polynomial matrices, the extension to Ore polynomial matrices is not immediate because multiplication of the matrix entries is not commutative. A number of results for polynomial matrices are extended to Ore polynomial matrices in this paper. This in turn allows nullspace algorithms to be used in Popov form computations. Fraction-free and modular algorithms for nullspace computation can be used in exact arithmetic setting where coefficient growth is a concern. When specialized to ordinary polynomial matrices, our results simplify the proofs for the computation of Popov form while keeping the same worst case complexity.

1 Introduction

Ore polynomial matrices provide a general setting for describing systems of linear differential, difference and q-difference operators [12]. We look at the problem of transforming such matrices into a normal form known as the Popov form. If a matrix is in Popov form, one may rewrite high-order operators (e.g. derivatives) in terms of lower ones (Example 2.5). Algorithms for computing the Popov form for polynomial matrices are well known [9, 10], but there have been few works on the computation of Popov form for Ore polynomial matrices. The problem was studied in [8] using row reductions, which can introduce significant coefficient growth which must be controlled. This is important for Ore polynomials as coefficient growth is introduced in two ways—from multiplying by powers of the indeterminate and from elimination by cross-multiplication.

Fraction-free and modular algorithms [1, 5] exist to compute a minimal polynomial basis of the left nullspace of Ore polynomial matrices, such that the basis is given by an Ore polynomial matrix in Popov form. We show that the problem of computing the Popov form and the associated unimodular transformation matrix can be reduced to the problem of computing a left nullspace in Popov form. The case when the input matrix has full row rank has been examined in a previous work [6, 7]. When the input matrix does not have full row rank, the unimodular multiplier is not unique. Instead, we define a unique minimal multiplier and show the reduction can still be applied by giving a degree bound for the minimal multiplier.

The technique of reducing the computation of normal forms such as row-reduced form and Popov form is well known for polynomial matrices [2, 3, 4, 11]. Unfortunately, the proofs of many of the results rely on the fact that the entries of the matrices commute. The main contribution of our work is to extend the results to Ore polynomial matrices. For the special case of ordinary polynomial matrices, we obtain the same worst case complexity as those obtained previously [3] with simpler proofs.

2 Notations and Definitions

We first give some notations and definitions similar to those given in previous works [1].

For any matrix $\mathbf{A}$, we denote its elements by $\mathbf{A}_{i,j}$. For any sets of row and column indices I and J, we denote by $\mathbf{A}_{I,J}$ the submatrix of $\mathbf{A}$ consisting of the rows and columns indexed by I and J. For convenience, we use I_c to denote the complement of the set I, and $*$ for I and J to denote the sets of all rows and columns, respectively. For any vector of non-negative integers $\vec{\omega} = (\omega_1, \ldots, \omega_p)$, we denote by $|\vec{\omega}| = \sum_{i=1}^{p} \omega_i$. We define $\vec{e} = (1, \ldots, 1)$ of the appropriate dimension. We denote by $\mathbf{I}_m$ the $m \times m$ identity matrix.

In this paper, we will examine Ore polynomial rings with coefficients in a field $\mathbb{K}$. That is, the ring $\mathbb{K}[Z; \sigma, \delta]$ with σ an automorphism and δ a derivation, so that the multiplication rule holds for all $a \in \mathbb{K}$:

$$Z \cdot a = \sigma(a)Z + \delta(a).$$

Let $\mathbb{K}[Z; \sigma, \delta]^{m\times n}$ be the ring of $m \times n$ Ore polynomial matrices over $\mathbb{K}[Z; \sigma, \delta]$. Let $\mathbf{F}(Z) \in \mathbb{K}[Z; \sigma, \delta]^{m\times n}$ and $N = \deg \mathbf{F}(Z)$. An Ore polynomial matrix $\mathbf{F}(Z)$ is said to have *row degree* $\vec{\mu} = \operatorname{rdeg} \mathbf{F}(Z)$ if the ith row has degree μ_i. The *leading row coefficient* of $\mathbf{F}(Z)$, denoted $\mathrm{LC}_{row}(\mathbf{F}(Z))$, is the matrix whose entries are the coefficients of Z^N of the corresponding elements of $Z^{N\cdot\vec{e}-\vec{\mu}} \cdot \mathbf{F}(Z)$. An Ore polynomial matrix $\mathbf{F}(Z)$ is *row-reduced* if $\mathrm{LC}_{row}(\mathbf{F}(Z))$ has maximal row rank. We also recall that the *rank* of $\mathbf{F}(Z)$ is the maximum number of $\mathbb{K}[Z; \sigma, \delta]$-linearly independent rows of $\mathbf{F}(Z)$, and that $\mathbf{U}(Z) \in \mathbb{K}[Z; \sigma, \delta]^{m\times m}$ is *unimodular* if there exists $\mathbf{V}(Z) \in \mathbb{K}[Z; \sigma, \delta]^{m\times m}$ such that $\mathbf{V}(Z) \cdot \mathbf{U}(Z) = \mathbf{U}(Z) \cdot \mathbf{V}(Z) = \mathbf{I}_m$.

Definition 2.1 (Pivot Index) *Let $\mathbf{F}(Z) \in \mathbb{K}[Z; \sigma, \delta]^{m\times n}$ with row degree $\vec{\mu}$. We define the* pivot index Π_i *of the ith row as*

$$\Pi_i = \begin{cases} \min_{1\le j\le n} \left\{ j : \deg \mathbf{F}(Z)_{i,j} = \mu_i \right\} & \mu_i \ge 0, \\ 0 & \textit{otherwise.} \end{cases} \tag{1}$$

Definition 2.2 (Popov Normal Form) *Let $\mathbf{F}(Z) \in \mathbb{K}[Z; \sigma, \delta]^{m\times n}$ with pivot indices $\Pi_1, \ldots, \Pi_m$ and row degree $\vec{\mu}$. Then $\mathbf{F}(Z)$ is in* Popov form *if it may be partitioned as*

$$\mathbf{F}(Z) = \begin{bmatrix} \mathbf{0} \\ \mathbf{F}(Z)_{J_c,*} \end{bmatrix}, \tag{2}$$

where $J = (1, \ldots, n - r)$ and $r = rank\ \mathbf{F}(Z)$, and for all $i, j \in J_c$ we have

(a) $\Pi_i < \Pi_j$ *whenever $i < j$;*

(b) $\mathbf{F}(Z)_{i,\Pi_i}$ *is monic;*

(c) If $k = \Pi_j$ for some $j \ne i$, then $\deg \mathbf{F}(Z)_{i,k} < \mu_j$.

If a matrix is in Popov form, its pivot set *is defined as* $\{\Pi_i\ :\ \Pi_i > 0\}$.

Every matrix $\mathbf{F}(Z)$ can be transformed into a unique matrix in Popov form using the following elementary row operations:

(a) interchange two rows;

(b) multiply a row by a nonzero element in $\mathbb{K}$;

(c) add a polynomial multiple of one row to another.

Formally, we may view a sequence of elementary row operations as a *unimodular transformation matrix* $\mathbf{U}(Z) \in \mathbb{K}[Z; \sigma, \delta]^{m\times m}$ with the result of these operations given by $\mathbf{T}(Z) = \mathbf{U}(Z) \cdot \mathbf{F}(Z)$. We recall the following result from [1, Theorem 2.2].

Theorem 2.3 *For any $\mathbf{F}(Z) \in \mathbb{K}[Z; \sigma, \delta]^{m\times n}$ there exists a unimodular matrix $\mathbf{U}(Z) \in \mathbb{K}[Z; \sigma, \delta]^{m\times m}$, with $\mathbf{T}(Z) = \mathbf{U}(Z) \cdot \mathbf{F}(Z)$ having $r \le \min\{m, n\}$ nonzero rows, rdeg $\mathbf{T}(Z) \le$ rdeg $\mathbf{F}(Z)$, and where the submatrix consisting of the r nonzero rows of $\mathbf{T}(Z)$ is row-reduced. Moreover, the unimodular multiplier satisfies the degree bound*

$$\textit{rdeg}\ \mathbf{U}(Z) \le \vec{\nu} + (|\vec{\mu}| - |\vec{\nu}| - \alpha) \cdot \vec{e} \tag{3}$$

where $\vec{\mu} = \max(\vec{0}, \textit{rdeg}\ \mathbf{F}(Z))$, $\vec{\nu} = \max(\vec{0}, \textit{rdeg}\ \mathbf{T}(Z))$, and $\alpha = \min_j\{\mu_j\}$.

We also recall the predictable degree property for Ore polynomial matrices [1, Lemma A.1(a)]. This result is used a number of times in our proofs.

Lemma 2.4 (Predictable Degree Property) *Let $\mathbf{F}(Z) \in \mathbb{K}[Z;\sigma,\delta]^{m\times n}$ with $\vec{\mu} = rdeg\,\mathbf{F}(Z)$. Then $\mathbf{F}(Z)$ is row-reduced if and only if, for all $\mathbf{Q}(Z) \in \mathbb{K}[Z;\sigma,\delta]^{1\times m}$,*

$$\deg \mathbf{Q}(Z)\mathbf{F}(Z) = \max_j(\mu_j + \deg \mathbf{Q}(Z)_{1,j}. \tag{4}$$

Example 2.5 *Consider the differential algebraic system*

$$\begin{array}{rcrcrcl} y_1''(t) + (t+2)y_1(t) & + & y_2''(t) + y_2(t) & + & y_3'(t) + y_3(t) & = & 0 \\ y_1'(t) + 3y_1(t) & + & y_2'''(t) + 2y_2'(t) - y_2(t) & + & y_3'''(t) - 2t^2 y_3(t) & = & 0 \\ y_1'(t) + y_1(t) & + & y_2''(t) + 2ty_2'(t) - y_2(t) & + & y_3''''(t) & = & 0. \end{array} \tag{5}$$

Let D denote the differential operator on $\mathbb{Q}(t)$ such that $D \cdot f(t) = \frac{d}{dt}f(t)$. Then the matrix form of (5) is:

$$\begin{bmatrix} D^2 + (t+2) & D^2+1 & D+1 \\ D+3 & D^3+2D-1 & D^3-2t^2 \\ D+1 & D^2+2tD+1 & D^4 \end{bmatrix} \cdot \begin{bmatrix} y_1(t) \\ y_2(t) \\ y_3(t) \end{bmatrix} = \mathbf{0}. \tag{6}$$

The matrix of operators is in Popov form with row degree $(2,3,4)$ and pivot set $\{1,2,3\}$. Notice that we can now convert every highest derivative into ones of lower order. For example, we can eliminate the highest derivatives of $y_2(t)$ as

$$y_2'''(t) = -y_1'(t) - 3y_1(t) - 2y_2'(t) + y_2(t) - y_3'''(t) + 2t^2 y_3(t). \tag{7}$$

3 General Approach

Given an $m\times n$ matrix $\mathbf{F}(Z) \in \mathbb{K}[Z;\sigma,\delta]^{m\times n}$, we wish to compute a unimodular matrix $\mathbf{U}(Z) \in \mathbb{K}[Z;\sigma,\delta]^{m\times m}$ and $\mathbf{T}(Z) \in \mathbb{K}[Z;\sigma,\delta]^{m\times n}$ such that $\mathbf{U}(Z)\cdot\mathbf{F}(Z) = \mathbf{T}(Z)$, where $\mathbf{T}(Z)$ is in Popov form. The fraction-free and modular algorithms [1, 5] can be used to compute a minimal polynomial basis $\mathbf{M}(Z)$ of the left nullspace of a Ore polynomial matrix such that $\mathbf{M}(Z)$ is in Popov form. Using these algorithms, we compute the left nullspace of the matrix $\begin{bmatrix}\mathbf{F}(Z)\cdot Z^b & -\mathbf{I}_n.\end{bmatrix}^T$. Then the nullspace $\mathbf{M}(Z)$ can be partitioned as $[\mathbf{U}(Z) \quad \mathbf{T}(Z)\cdot Z^b]$ such that

$$\begin{bmatrix}\mathbf{U}(Z) & \mathbf{T}(Z)\cdot Z^b\end{bmatrix} \cdot \begin{bmatrix}\mathbf{F}(Z)\cdot Z^b \\ -\mathbf{I}_n\end{bmatrix} = \mathbf{0}. \tag{8}$$

The matrix $\mathbf{U}(Z)$ obtained in this manner is unimodular.

Lemma 3.1 *Suppose that $\begin{bmatrix}\mathbf{U}(Z) & \mathbf{T}(Z)\end{bmatrix}$ is a basis of the left nullspace of $\begin{bmatrix}\mathbf{F}(Z) \\ -\mathbf{I}_n\end{bmatrix}$. Then $\mathbf{U}(Z)$ is unimodular.*

Proof. The rows of $\begin{bmatrix}\mathbf{I}_m & \mathbf{F}(Z)\end{bmatrix}$ belong to the left nullspace of $\begin{bmatrix}\mathbf{F}(Z) \\ -\mathbf{I}_n\end{bmatrix}$. Since $\begin{bmatrix}\mathbf{U}(Z) & \mathbf{T}(Z)\end{bmatrix}$ is a basis of the left nullspace, there exists $\mathbf{V}(Z) \in \mathbb{K}[Z;\sigma,\delta]^{m\times m}$ such that $\mathbf{V}(Z)\cdot\mathbf{U}(Z) = \mathbf{I}_m$. Thus, $\mathbf{U}(Z)$ has a left inverse. Now, $\mathbf{U}(Z)\cdot\mathbf{V}(Z)\cdot\mathbf{U}(Z) = \mathbf{U}(Z)$. Therefore,

$$(\mathbf{U}(Z)\cdot\mathbf{V}(Z) - \mathbf{I}_m)\cdot\mathbf{U}(Z) = \mathbf{0}. \tag{9}$$

Since $m = \text{rank }\mathbf{I}_m = \text{rank }(\mathbf{V}(Z)\cdot\mathbf{U}(Z)) \le \text{rank }\mathbf{U}(Z) \le m$, $\mathbf{U}(Z)$ has full row rank. Thus, (9) implies that $\mathbf{U}(Z)\cdot\mathbf{V}(Z) - \mathbf{I}_m = \mathbf{0}$, so that $\mathbf{V}(Z)$ is also a right inverse of $\mathbf{U}(Z)$. Since $\mathbf{U}(Z)$ has a two-sided inverse, it is unimodular. $\square$

If $b > \deg\mathbf{U}(Z)$, this also implies that $\mathbf{T}(z)$ is in Popov form since the leading coefficients are "contributed" by $\mathbf{T}(z)$. Thus, our goal is to determine an upper bound on $\deg\mathbf{U}(Z)$. A similar approach has also been used to compute the row-reduced form and the Popov form of polynomial matrices [2, 3, 4, 11].

4 Degree Bound in the Full Row Rank Case

In the case when the input matrix $\mathbf{F}(Z)$ has full row rank, we follow the approach of [4] in order to obtain a bound for $\deg \mathbf{U}(Z)$. We first prove some results which relate the degrees of the input matrix $\mathbf{F}(Z)$, the unimodular multiplier $\mathbf{U}(Z)$, and any matrix $\mathbf{T}(Z)$ resulting from the row transformation specified by $\mathbf{U}(Z)$.

Lemma 4.1 *Suppose $\mathbf{F}(Z) \in \mathbb{K}[Z;\sigma,\delta]^{m\times n}$ has full row rank, and let $\mathbf{T}_1(Z) \in \mathbb{K}[Z;\sigma,\delta]^{m\times n}$ be a row-reduced form of $\mathbf{F}(Z)$. Suppose that $\mathbf{T}_2(Z) = \mathbf{U}_2(Z)\cdot\mathbf{F}(Z)$ for some unimodular matrix $\mathbf{U}_2(Z) \in \mathbb{K}[Z;\sigma,\delta]^{m\times m}$, with $\vec{\gamma} =$ rdeg $\mathbf{T}_2(Z)$. There exists a unimodular matrix $\mathbf{V}(Z)$ such that $\mathbf{T}_2(Z) = \mathbf{V}(Z) \cdot \mathbf{T}_1(Z)$ and $\deg \mathbf{V}(Z)_{i,j} \leq \gamma_i - \nu_j$ where $\vec{\nu} =$ rdeg $\mathbf{T}_1(Z)$.*

Proof. Since $\mathbf{T}_1(Z)$ is a row-reduced form of $\mathbf{F}(Z)$, there exists a unimodular matrix $\mathbf{U}_1(Z) \in \mathbb{K}[Z;\sigma,\delta]^{m\times m}$ such that $\mathbf{U}_1(Z) \cdot \mathbf{F}(Z) = \mathbf{T}_1(Z)$. Setting $\mathbf{V}(Z) = \mathbf{U}_2(Z) \cdot \mathbf{U}_1(Z)^{-1}$ gives $\mathbf{T}_2(Z) = \mathbf{V}(Z) \cdot \mathbf{T}_1(Z)$. Since $\mathbf{V}(Z)$ is a product of unimodular matrices, it is unimodular.

Since $\mathbf{T}_1(Z)$ is row-reduced, Lemma 2.4 gives

$$\deg \mathbf{V}(Z)_{i,j} + \deg \mathbf{T}_1(Z)_{j,\cdot} \leq \deg \mathbf{T}_2(Z)_{i,\cdot}, \tag{10}$$

which implies that $\deg \mathbf{V}(Z)_{i,j} \leq \gamma_i - \nu_j$. □

Theorem 4.2 *Suppose that $\mathbf{F}(Z) \in \mathbb{K}[Z;\sigma,\delta]^{m\times n}$ has full row rank. Let $\mathbf{V}(Z) \in \mathbb{K}[Z;\sigma,\delta]^{m\times m}$ be unimodular and let $\mathbf{T}(Z) = \mathbf{V}(Z) \cdot \mathbf{F}(Z)$ with $\vec{\gamma} =$ rdeg $\mathbf{T}(Z)$. There exists a unimodular matrix $\mathbf{U}(Z)$ such that $\mathbf{U}(Z) \cdot \mathbf{F}(Z) = \mathbf{T}(Z)$ and rdeg $\mathbf{U}(Z) \leq \vec{\gamma} + (|\vec{\mu}| - \alpha) \cdot \vec{e}$, where $\vec{\mu} =$ rdeg $\mathbf{F}(Z)$ and $\alpha = \min_j\{\mu_j\}$.*

Proof. By [1, Theorem 2.2], there exists a unimodular matrix $\mathbf{U}_1(Z)$ such that $\mathbf{T}_1(Z) = \mathbf{U}_1(Z) \cdot \mathbf{F}(Z)$ is row-reduced and rdeg $\mathbf{U}_1(Z) \leq \vec{\nu} + (|\vec{\mu}| - |\vec{\nu}| - \alpha) \cdot \vec{e}$, with $\vec{\nu} =$ rdeg $\mathbf{T}_1(Z)$. By Lemma 4.1, there exists a unimodular matrix $\mathbf{U}_2(Z)$ such that $\mathbf{T}(Z) = \mathbf{U}_2(Z) \cdot \mathbf{T}_1(Z) = \mathbf{U}_2(Z) \cdot \mathbf{U}_1(Z) \cdot \mathbf{F}(Z)$. Setting $\mathbf{U}(Z) = \mathbf{U}_2(Z) \cdot \mathbf{U}_1(Z)$ gives $\mathbf{U}(Z) \cdot \mathbf{F}(Z) = \mathbf{T}(Z)$. For the degree bound, note that

$$\deg \mathbf{U}(Z)_{i,j} \leq \max_{1\leq k\leq m} \deg \mathbf{U}_2(Z)_{i,k} + \deg \mathbf{U}_1(Z)_{k,j} \leq \max_{1\leq k\leq m} (\gamma_i - \nu_k) + (\nu_k + |\vec{\mu}| - |\vec{\nu}| - \alpha) \leq \gamma_i + |\vec{\mu}| - \alpha.$$

□

We have only stated the existence of unimodular matrices satisfying certain degree bounds in the previous results. We now show that such unimodular matrices are also unique.

Lemma 4.3 *Suppose that $\mathbf{F}(Z) \in \mathbb{K}[Z;\sigma,\delta]^{m\times n}$ has full row rank. Given $\mathbf{T}(Z) \in \mathbb{K}[Z;\sigma,\delta]^{m\times n}$, the solution $\mathbf{U}(Z) \in \mathbb{K}[Z;\sigma,\delta]^{m\times m}$ to the equation $\mathbf{U}(Z) \cdot \mathbf{F}(Z) = \mathbf{T}(Z)$ is unique (if it exists).*

Proof. Let $\mathbf{U}_1(Z)$ and $\mathbf{U}_2(Z)$ be two matrices such that

$$\mathbf{U}_1(Z) \cdot \mathbf{F}(Z) = \mathbf{T}(Z) = \mathbf{U}_2(Z) \cdot \mathbf{F}(Z). \tag{11}$$

Then $(\mathbf{U}_1(Z) - \mathbf{U}_2(Z)) \cdot \mathbf{F}(Z) = \mathbf{0}$. Since $\mathbf{F}(Z)$ has full row rank, it follows that $\mathbf{U}_1(Z) - \mathbf{U}_2(Z) = \mathbf{0}$ and hence $\mathbf{U}_1(Z) = \mathbf{U}_2(Z)$. □

Since $\mathbf{F}(Z)$ has full row rank, the uniqueness of the unimodular multiplier gives us a bound on the degree of the unimodular multiplier by Theorem 4.2 and Lemma 4.3.

Theorem 4.4 *Suppose that $\mathbf{F}(Z)$ has full row rank. If $\mathbf{T}(Z) = \mathbf{U}(Z) \cdot \mathbf{F}(Z)$ for some unimodular matrix $\mathbf{U}(Z)$ then $\mathbf{U}(Z)$ satisfies the degree bound (3).*

Finally, we give a degree bound on $\mathbf{U}(Z)$ and provide a method to compute the Popov form of $\mathbf{F}(Z)$ and the associated unimodular multiplier $\mathbf{U}(Z)$.

Theorem 4.5 *Suppose that $\mathbf{F}(Z) \in \mathbb{K}[Z;\sigma,\delta]^{m\times n}$ has full row rank and has row degree $\vec{\mu}$. Let $b > |\vec{\mu}| - \min_j\{\mu_j\}$, and suppose $\begin{bmatrix}\mathbf{U}(Z) & \mathbf{R}(Z)\end{bmatrix}$ is a basis in Popov form of the left nullspace of $\begin{bmatrix}\mathbf{F}(Z) \cdot Z^b & -\mathbf{I}_n\end{bmatrix}^T$. Let $\mathbf{T}(Z) = \mathbf{R}(Z) \cdot Z^{-b}$. Then* *(a)* $\mathbf{U}(Z)$ *is unimodular;*

(b) $\mathbf{T}(Z) = \mathbf{U}(Z) \cdot \mathbf{F}(Z) \in \mathbb{K}[Z;\sigma,\delta]^{m\times n}$*;*

(c) $\mathbf{T}(Z)$ *is in Popov form.*

Proof. Part (a) is immediate from Lemma 3.1. For (b), we see that $\mathbf{U}(Z)\cdot\mathbf{F}(Z)\cdot Z^b = \mathbf{R}(Z)$, so $\mathbf{T}(Z) = \mathbf{U}(Z)\cdot\mathbf{F}(Z)$. To prove (c), we see from Theorem 4.4 that rdeg $\mathbf{U}(Z) \le \vec{\nu} + (|\vec{\mu}| - \alpha)\cdot\vec{e}$ where $\vec{\mu} =$ rdeg $\mathbf{F}(Z)$, $\vec{\nu} =$ rdeg $\mathbf{T}(Z)$, and $\alpha = \min_j\{\mu_j\}$. Therefore, rdeg $\mathbf{U}(Z) \le$ rdeg $\mathbf{R}(Z) + (|\vec{\mu}| - \alpha - b)\cdot\vec{e} <$ rdeg $\mathbf{R}(Z)$. Thus, the leading coefficient of $\begin{bmatrix}\mathbf{U}(Z) & \mathbf{R}(Z)\end{bmatrix}$ is the same as the leading coefficient of $\begin{bmatrix}\mathbf{0} & \mathbf{R}(Z)\end{bmatrix}$. It follows that $\mathbf{R}(Z)$ and hence $\mathbf{T}(Z)$ is in Popov form. □

5 Minimal Multipliers

In the case when the input matrix $\mathbf{F}(Z)$ does not have full row rank, the situation is considerably more complicated. In fact, a unimodular multiplier of arbitrarily high degree exists. Suppose $\mathbf{T}(Z) = \begin{bmatrix}\mathbf{0} & \mathbf{T}(Z)_{J_c,*}\end{bmatrix}^T = \mathbf{U}(Z)\cdot\mathbf{F}(Z)$ is the Popov form of $\mathbf{F}(Z)$. One may add any polynomial multiple of the rows of $\mathbf{U}(Z)_{J,*}$ to the other rows of $\mathbf{U}(Z)$ and still obtain a unimodular multiplier $\mathbf{U}'(Z)$ satisfying $\mathbf{T}(Z) = \mathbf{U}'(Z)\cdot\mathbf{F}(Z)$.

In fact, all unimodular multipliers satisfying $\mathbf{T}(Z) = \mathbf{U}(Z)\cdot\mathbf{F}(Z)$ are related, and there is a unique multiplier that has minimal column degrees and is normalized in some way. We first give a result related to "division" of Ore polynomial matrices. This allows us to "reduce" one Ore polynomial matrix by another one that is in Popov form to obtain a unique remainder. This is an analogue of [3, Lemma 3.5].

Lemma 5.1 *Let* $\mathbf{B}(Z) \in \mathbb{K}[Z;\sigma,\delta]^{n\times n}$ *be a full row rank matrix in Popov form with row degree* $\vec{\beta}$*. Then for any* $\mathbf{A}(Z) \in \mathbb{K}[Z;\sigma,\delta]^{m\times n}$ *with row degree* $\vec{\gamma}$*, there exist unique matrices* $\mathbf{Q}(Z), \mathbf{R}(Z) \in \mathbb{K}[Z;\sigma,\delta]^{m\times n}$ *such that*

$$\mathbf{A}(Z) - \mathbf{Q}(Z)\cdot\mathbf{B}(Z) = \mathbf{R}(Z), \tag{12}$$

where for all i,j, $\deg\mathbf{R}(Z)_{i,j} < \beta_j$ *and* $\deg\mathbf{Q}(Z)_{i,j} \le \gamma_i - \beta_j$.

Proof. It suffices to prove this in the case $m = 1$ as we may consider each row of (12) independently.

We first show the existence of $\mathbf{Q}(Z)$ and $\mathbf{R}(Z)$. Let $K = \{k : \deg\mathbf{A}(Z)_{1,k} \ge \beta_k\}$, and $d = \deg\mathbf{A}(Z)_{1,K}$. Let $t \in K$ be the pivot index of $\mathbf{A}(Z)_{1,K}$. Thus, $\mathbf{A}(Z)_{1,t} = aZ^d + \cdots$ for some $a \in \mathbb{K}$. If $\mathbf{B}(Z)_{t,t} = bZ^{\beta_t} + \cdots$ for some $b \in \mathbb{K}$. Let $\hat{\mathbf{R}}_\mathbf{1}(Z) = \mathbf{A}(Z) - \hat{\mathbf{Q}}_\mathbf{1}(Z)\cdot\mathbf{B}(Z)$ where $\hat{\mathbf{Q}}_\mathbf{1}(Z) = \begin{bmatrix}0\cdots 0 & \frac{a}{\sigma^{d-\beta_t}(b)}Z^{d-\beta_t} & 0\cdots 0\end{bmatrix}$ with the nonzero element in the t^{th} column. It is easy to see that $\hat{\mathbf{R}}_\mathbf{1}(Z)_{1,t} < d$. Since $\mathbf{B}(Z)$ is in Popov form,

$$\deg\mathbf{B}(Z)_{t,s} \le \begin{cases}\beta_t & \text{if } s \ge t,\\ \beta_t - 1 & \text{otherwise.}\end{cases} \tag{13}$$

From the degree bounds on $\mathbf{A}(Z)_{1,K}$, we see that for $s \in K$ we have

$$\deg\hat{\mathbf{R}}_\mathbf{1}(Z)_{1,s} \le \begin{cases}d & \text{if } s > t,\\ d-1 & \text{otherwise.}\end{cases} \tag{14}$$

For $s \notin K$, we have $\deg\hat{\mathbf{R}}_\mathbf{1}(Z)_{1,s} \le \max(\deg\mathbf{A}(Z)_{1,s}, \deg[\hat{\mathbf{Q}}_\mathbf{1}(Z)\cdot\mathbf{B}(Z)]_{1,s})$. If $\deg\hat{\mathbf{R}}_\mathbf{1}(Z)_{1,s} \le \deg\mathbf{A}(Z)_{1,s}$, then $\deg\hat{\mathbf{R}}_\mathbf{1}(Z)_{1,s} < \beta_s$ by definition of K. Otherwise,

$$\deg\hat{\mathbf{R}}_\mathbf{1}(Z)_{1,s} = \deg[\hat{\mathbf{Q}}_\mathbf{1}(Z)\cdot\mathbf{B}(Z)]_{1,s} \le \begin{cases}(d-\beta_t)+\beta_t = d & \text{if } s > t,\\ (d-\beta_t)+\beta_t - 1 = d-1 & \text{otherwise.}\end{cases} \tag{15}$$

Let $\hat{K} = \{k : \deg \hat{\mathbf{R}}_\mathbf{1}(Z)_{1,k} \geq \beta_k\}$. We see that either $\deg \hat{\mathbf{R}}_\mathbf{1}(Z) < d$, or $\deg \hat{\mathbf{R}}_\mathbf{1}(Z) = d$ and the pivot index of $\hat{\mathbf{R}}_\mathbf{1}(Z)_{1,\hat{K}}$ must be greater than t. We also note that it is possible that $\hat{K} \neq K$.

Continuing in this way we may construct $\hat{\mathbf{R}}_\mathbf{2}(Z), \hat{\mathbf{R}}_\mathbf{3}(Z), \ldots$, so that after each step either the degree is decreased or the pivot index is increased. Therefore, in a finite number of steps we will have $\hat{\mathbf{R}}_\mathbf{k}(Z) = \mathbf{A}(Z) - \left[\hat{\mathbf{Q}}_\mathbf{1}(Z) + \cdots + \hat{\mathbf{Q}}_\mathbf{k}(Z)\right] \cdot \mathbf{B}(Z)$, where $\deg \hat{\mathbf{R}}_\mathbf{k}(Z)_{1,j} < \beta_j$ for all j. Finally, setting $\mathbf{Q}(Z) = \hat{\mathbf{Q}}_\mathbf{1}(Z) + \cdots + \hat{\mathbf{Q}}_\mathbf{k}(Z)$, $\mathbf{R}(Z) = \hat{\mathbf{R}}_\mathbf{k}(Z)$ gives us the desired divisor and remainder matrices of (12).

To show uniqueness, suppose that we have $\mathbf{A}(Z)_{1,*} = \mathbf{Q}_\mathbf{1}(Z) \cdot \mathbf{B}(Z) + \mathbf{R}_\mathbf{1}(Z) = \mathbf{Q}_\mathbf{2}(Z) \cdot \mathbf{B}(Z) + \mathbf{R}_\mathbf{2}(Z)$ for some $\mathbf{Q}_\mathbf{1}(Z)$, $\mathbf{Q}_\mathbf{2}(Z)$, $\mathbf{R}_\mathbf{1}(Z)$, and $\mathbf{R}_\mathbf{2}(Z) \in \mathbb{K}[Z;\sigma,\delta]^{1\times n}$. Letting $\hat{\mathbf{Q}}(Z) = \mathbf{Q}_\mathbf{1}(Z) - \mathbf{Q}_\mathbf{2}(Z)$ and $\hat{\mathbf{R}}(Z) = \mathbf{R}_\mathbf{2}(Z) - \mathbf{R}_\mathbf{1}(\mathbf{Z})$ gives $\hat{\mathbf{R}}(Z) = \hat{\mathbf{Q}}(Z) \cdot \mathbf{B}(Z)$ with $\deg \hat{\mathbf{R}}(Z)_{1,j} < \beta_j$. Let k be such that $\deg \hat{\mathbf{R}}(Z)_{1,k} = \deg \hat{\mathbf{R}}(\mathbf{Z})$. Since $\mathbf{B}(Z)$ is row reduced, Lemma 2.4 implies that $\deg \hat{\mathbf{Q}}(\mathbf{Z})_{1,k} \leq \deg \hat{\mathbf{R}}(Z)_{1,k} - \beta_k < 0$, so that $\hat{\mathbf{Q}}(\mathbf{Z})_{1,k} = 0$ whenever $\deg \hat{\mathbf{R}}(Z)_{1,k} = \deg \hat{\mathbf{R}}(\mathbf{Z})$. Now, let $K = \{k : \deg \hat{\mathbf{R}}(Z)_{1,k} < \deg \hat{\mathbf{R}}(\mathbf{Z})\}$. If K is nonempty, consider the equation $\hat{\mathbf{R}}(Z)_{1,K} = \hat{\mathbf{Q}}(Z)_{1,K} \cdot \mathbf{B}(Z)_{K,K}$. A similar argument shows that $\hat{\mathbf{Q}}(Z)_{1,k} = 0$ whenever $\deg \hat{\mathbf{R}}(\mathbf{Z})_{1,k} = \deg \hat{\mathbf{R}}(\mathbf{Z})_{1,K}$. Continuing in this way it can be seen that $\hat{\mathbf{Q}}(Z) = \hat{\mathbf{R}}(Z) = \mathbf{0}$, so that the matrices $\mathbf{Q}(Z)$ and $\mathbf{R}(Z)$ in (12) are unique.

Finally, we prove the degree bound for $\mathbf{Q}(Z)$. For any $1 \leq i \leq m$, let $L_i = \{j : \gamma_i \geq \beta_j\}$. Then for $j \notin L_i$ we have $\gamma_i < \beta_j$ and therefore $\mathbf{Q}(Z)_{i,j} = 0$ because $\mathbf{Q}(Z)$ is unique. If $j \in L_i$, we have

$$\deg(\mathbf{Q}(Z)_{i,L_i} \cdot \mathbf{B}(Z)_{L_i,L_i}) = \deg(\mathbf{A}(Z)_{i,L_i} - \mathbf{R}(Z)_{i,L_i}) \leq \gamma_i. \tag{16}$$

Lemma 2.4 gives $\deg(\mathbf{Q}(Z)_{i,L_i} \cdot \mathbf{B}(Z)_{L_i,L_i}) \geq \deg \mathbf{Q}(Z)_{i,j} + \beta_j$, for all $j \in L_i$. □

We can now show the main result in this section which shows the relationship among all unimodular multipliers. This result is an analogue of [3, Theorem 3.3].

Theorem 5.2 *Let $\mathbf{F}(Z) \in \mathbb{K}[Z;\sigma,\delta]^{m\times n}$ with row rank r. Let $\mathbf{U}(Z) \in \mathbb{K}[Z;\sigma,\delta]^{m\times m}$ be unimodular such that $\mathbf{U}(Z) \cdot \mathbf{F}(Z) = \mathbf{T}(Z)$, with $\mathbf{T}(Z) = \begin{bmatrix} 0 \\ \mathbf{T}(Z)_{J_c,*} \end{bmatrix}$ the unique Popov form of $\mathbf{F}(Z)$.*

(a) A unimodular matrix $\mathbf{U}(Z)$ is unique up to multiplication on the left by matrices of the form

$$\mathbf{W}(Z) = \begin{bmatrix} \mathbf{W}(Z)_{J,J} & 0 \\ \mathbf{W}(Z)_{J_c,J} & \mathbf{I}_r \end{bmatrix}, \tag{17}$$

where $\mathbf{W}(Z)_{J,J} \in \mathbb{K}[Z;\sigma,\delta]^{(m-r)\times(m-r)}$ is unimodular.

(b) There exists a unique multiplier $\mathbf{U}(Z)$ such that $\mathbf{U}(Z)_{J,*}$ is a minimal polynomial basis in Popov form for the left nullspace of $\mathbf{F}(Z)$ with pivot set K, and for all $k \in K, j \in J_c$:

$$\deg \mathbf{U}(Z)_{j,k} < \max_{\ell \in J} \deg \mathbf{U}(Z)_{\ell,k} \tag{18}$$

(c) Under all multipliers mentioned in (a), the sum of the row degrees of the unique multiplier $\mathbf{U}(Z)$ of (b) is minimal.

Proof. To prove (a), let $\mathbf{U}_\mathbf{1}(Z)$ and $\mathbf{U}_\mathbf{2}(Z)$ be two such unimodular multipliers for the Popov form of $\mathbf{F}(Z)$. Then $\mathbf{U}_\mathbf{1}(Z)_{J,*}, \mathbf{U}_\mathbf{2}(Z)_{J,*}$, are bases of the left nullspace of $\mathbf{F}(Z)$. Thus there exists a unimodular multiplier $\mathbf{W}(Z)_{J,J}$ such that $\mathbf{U}_\mathbf{1}(Z)_{J,*} = \mathbf{W}(Z)_{J,J}\mathbf{U}_\mathbf{2}(Z)_{J,*}$. By the uniqueness of $\mathbf{T}(Z)_{J_c,*}$, the rows of $\mathbf{U}_\mathbf{2}(Z)_{J_c,*} - \mathbf{U}_\mathbf{1}(Z)_{J_c,*}$ are in the nullspace of $\mathbf{F}(Z)$, so there exists a matrix $\mathbf{W}(Z)_{J_c,J}$ such that $\mathbf{U}_\mathbf{2}(Z)_{J_c,*} = \mathbf{U}_\mathbf{1}(Z)_{J_c,*} + \mathbf{W}(Z)_{J_c,J}\mathbf{U}_\mathbf{1}(Z)_{J,*}$.

For (b), assume that $\mathbf{U}(Z)_{J,*}$ is the unique Popov minimal polynomial basis for the left nullspace with pivot set K. Given any multiplier $\mathbf{U}_\mathbf{0}(Z)$ we may divide $\mathbf{U}_\mathbf{0}(Z)_{J_c,K}$ on the right by $\mathbf{U}(Z)_{J,K}$ to get $\mathbf{U}_\mathbf{0}(Z)_{J_c,K} = \mathbf{W}(Z)_{J_c,J}\mathbf{U}(Z)_{J,K} + \mathbf{U}(Z)_{J_c,K}$. By Lemma 5.1, (18) is satisfied. Since $\mathbf{U}(Z)_{J_c,K}$ is the unique matrix such that (18) is satisfied, the generic form of a multiplier given in (a) implies that $\mathbf{U}(Z)_{J_c,*} = \mathbf{U}_\mathbf{0}(Z)_{J_c,*} - \mathbf{W}(Z)_{J_c,J}\mathbf{U}(Z)_{J,*}$. Thus, the minimal multiplier $\mathbf{U}(Z)$ is well defined and unique.

To prove (c), let $\mathbf{U_0}(Z)$ be a second unimodular multiplier. From the general form of the multipliers, the sum of the row degrees of J and J_c can be minimized independently. Since the degrees in J are minimized by choosing a minimal polynomial basis, we are only concerned about the rows in J_c. We want to show that $|\text{rdeg } \mathbf{U_0}(Z)_{J_c,*}| \geq |\text{rdeg } \mathbf{U}(Z)_{J_c,*}|$. Let $\vec{\beta} = \text{rdeg } \mathbf{U}(Z)_{J,*}$, $\vec{\mu} = \text{rdeg } \mathbf{U_0}(Z)_{J_c,K}$, and $\vec{\gamma} = \text{rdeg } \mathbf{U_0}(Z)_{J_c,K_c}$. The degree sum for $\mathbf{U_0}(Z)_{J_c,*}$ is $\sum_j \max(\mu_j, \gamma_j)$. By Lemma 5.1, we have quotient $\mathbf{W}(Z)_{J_c,J}$ such that $\mathbf{U}(Z)_{J_c,*} = \mathbf{U_0}(Z)_{J_c,*} - \mathbf{W}(Z)_{J_c,J}\mathbf{U}(Z)_{J,*}$ with $\deg \mathbf{W}(Z)_{i,j} \leq \mu_i - \beta_j$. Therefore we have, for $1 \leq i \leq m$ and $j \in J_c$, $\deg \mathbf{U}(Z)_{i,j} \leq \max(\max(\mu_i, \gamma_i), \mu_i) = \max(\mu_i, \gamma_i)$. Thus the degree sum of the J_c rows is not increased by the normalizing division, and gives (c). $\square$

The unique multiplier given in Theorem 5.2 (b) is called the *minimal multiplier.*

Theorem 5.3 *Let $\mathbf{U}(Z) \in \mathbb{K}[Z;\sigma,\delta]^{m\times m}$ be the minimal multiplier for $\mathbf{F}(Z) \in \mathbb{K}[Z;\sigma,\delta]^{m\times n}$ as in Theorem 5.2, and $\vec{\mu} = \text{rdeg } \mathbf{F}(Z)$. Then*

$$\deg \mathbf{U}(Z) \leq |\vec{\mu}| - \min_j\{\mu_j\}. \tag{19}$$

Proof. Let $\mathbf{T}(Z)$, J, and K be defined as in Theorem 5.2. We first note that if $\vec{\beta}$ is the row degree of the minimal polynomial basis, we have

$$\deg \mathbf{U}(Z)_{j,k} \leq \begin{cases} \beta_j & \text{if } j \in J, \\ \beta_j - 1 & \text{if } j \in J_c \text{ and } k \in K. \end{cases} \tag{20}$$

Since $\beta_i \leq |\vec{\mu}| - \min_j\{\mu_j\}$, it remains to obtain a bound for $\deg \mathbf{U}(Z)_{J_c,K_c}$.

Let $\mathbf{V}(Z) = \mathbf{U}(Z)^{-1}$ with row degree $\vec{\gamma}$. Then we have $\mathbf{F}(Z) = \mathbf{V}(Z)\cdot\mathbf{T}(Z)$, or $\mathbf{F}(Z) = \mathbf{V}(Z)_{*,J_c}\cdot\mathbf{T}(Z)_{J_c,*}$ because $\mathbf{T}(Z)_{J,*} = \mathbf{0}$. We wish to obtain a degree bound for $\mathbf{V}(Z)$ and relate it to $\deg \mathbf{U}(Z)$.

Since $\mathbf{T}(Z)_{J_c,*}$ is in Popov form and hence row-reduced, Lemma 2.4 gives a degree bound on $\mathbf{V}(Z)_{*,J_c}$: $\deg \mathbf{V}(Z)_{i,j} \leq \mu_i - \gamma_j \leq \mu_i$ for all $1 \leq i \leq m$, $j \in J_c$.

Let $r = \text{rank } \mathbf{F}(Z)$. Since $\mathbf{V}(Z) \cdot \mathbf{U}(Z) = \mathbf{I}$, we have

$$\mathbf{I}_{m-r} - \mathbf{V}(Z)_{K,J_c} \cdot \mathbf{U}(Z)_{J_c,K} = \mathbf{V}(Z)_{K,J} \cdot \mathbf{U}(Z)_{J,K} \tag{21}$$

$$-\mathbf{V}(Z)_{K_c,J_c} \cdot \mathbf{U}(Z)_{J_c,K} = \mathbf{V}(Z)_{K_c,J} \cdot \mathbf{U}(Z)_{J,K}. \tag{22}$$

In each of the above equations, the degree bound of row i on the left-hand side is at most $\mu_i + |\vec{\mu}| - \min_j\{\mu_j\}$. On the right-hand side, $\mathbf{U}(Z)_{J,K}$ is in Popov form and hence row-reduced. Lemma 2.4 again gives

$$\mu_i + |\vec{\mu}| - \min_j\{\mu_j\} \geq \deg \mathbf{V}(Z)_{i,j} + |\vec{\mu}| - \min_j\{\mu_j\}, \tag{23}$$

or

$$\mathbf{V}(Z)_{i,j} \leq \mu_i \tag{24}$$

for all $1 \leq i \leq m$ and $j \in J$. Combining with the above, we see that $\text{rdeg } \mathbf{V}(Z) \leq \vec{\mu}$.

To obtain a degree bound for $\mathbf{U}(Z)$, we observe that the row-reduced form of $\mathbf{V}(Z)$ is the identity matrix and $\mathbf{U}(Z)$ is the unique unimodular transformation matrix for $\mathbf{V}(Z)$. Applying [1, Theorem 2.2]

$$\text{rdeg } \mathbf{U}(Z) \leq (|\vec{\mu}| - \min_j\{\mu_j\}) \cdot \vec{e}, \tag{25}$$

and the theorem follows. $\square$

Remark 5.4 *The degree bound obtained this way is not as accurate as the one in the commutative case in [3]. However, our proofs are simpler and our bounds are not worse than those obtained in [3, Corollary 5.5] in the worst case when the rank of the input matrix is not known in advance.*

Thus, the same value of b is sufficient even when the input matrix does not have full row rank. In particular, we do not need to know the rank of the input matrix in advance.

Theorem 5.5 *Theorem 4.5 is true for any $\mathbf{F}(Z) \in \mathbb{K}[Z;\sigma,\delta]^{m\times n}$.*

6 Conclusion

We have given a bound on the minimal multiplier, which in turn allows us to reduce the problem of computing the Popov form and the associated unimodular transformation as a left nullspace computation. Thus, nullspace algorithms which control coefficient growth can be applied.

In practice, the bound on the minimal multiplier may be too pessimistic. Because the complexity of the nullspace algorithms depend on the degree of the input matrix [1, 5], having a bound that is too large will decrease the performance of these algorithms. An alternate approach is suggested in [4] in which (8) is solved with a small starting value of b. The value of b is increased if the matrix $\mathbf{T}(Z)$ obtained from the nullspace is not in Popov form. In the cases where the degree bound on the minimal multiplier is very pessimistic this will provide a faster algorithm.

References

[1] B. Beckermann, H. Cheng, and G. Labahn. Fraction-free row reduction of matrices of Ore polynomials. *Journal of Symbolic Computation*, 41(5):513–543, 2006.

[2] B. Beckermann, G. Labahn, and G. Villard. Shifted normal forms of polynomial matrices. In *Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation*, pages 189–196. ACM, 1999.

[3] B. Beckermann, G. Labahn, and G. Villard. Normal forms of general polynomial matrices. *Journal of Symbolic Computation*, 41(6):708–737, 2006.

[4] Th. G. Beelen, G. J. van den Hurk, and C. Praagman. A new method for computing a column reduced polynomial matrix. *Systems & Control Letters*, 10:217–224, 1988.

[5] H. Cheng and G. Labahn. Output-sensitive modular algorithms for row reduction of matrices of Ore polynomials. *Computer Algebra 2006: Latest Advances in Symbolic Algorithms*, pages 43–66, 2007.

[6] P. Davies and H. Cheng. Computing popov form of Ore polynomial matrices. Technical report, Department of Mathematics and Computer Science, University of Lethbridge, Sep 2006.

[7] P. Davies, H. Cheng, and G. Labahn. Computing Popov form of Ore polynomial matrices. *Communications in Computer Algebra, ISSAC 2007 Poster Abstracts*, 41(2):49–50, 2007.

[8] M. Giesbrecht, G. Labahn, and Y. Zhang. Computing valuation popov forms. In *Workshop on Computer Algebra Systems and their Applications (CASA'05)*, 2005.

[9] T. Kailath. *Linear Systems*. Prentice-Hall, 1980.

[10] T. Mulders and A. Storjohann. On lattice reduction for polynomial matrices. *Journal of Symbolic Computation*, 35(4):377–401, 2003.

[11] W. H. L. Neven and C. Praagman. Column reduction of polynomial matrices. *Linear Algebra and Its Applications*, 188,189:569–589, 1993.

[12] O. Ore. Theory of non-commutative polynomials. *Annals of Mathematics*, 34:480–508, 1933.

Teaching first-year engineering students with “modern day” Maple

Frederick W. Chapman, Bruce W. Char, and Jeremy R. Johnson
Department of Computer Science
Drexel University, Philadelphia, PA 19104 USA
fwchapman@alumni.uwaterloo.ca, charbw@drexel.edu, jjohnson@cs.drexel.edu

October 22, 2008

1 Introduction

Computer algebra systems in general and Maple in particular have been used in mathematics education for over 20 years. Since the initial use of Maple in the classroom, such as described in [5], there has been substantial enhancements, such as student packages, tutors, maplets, and MapleTA to support education. Despite these enhancements, there remain issues in teaching these systems to a wide audience and using them in mathematics education. In lieu of alternative technologies such as graphing calculators, applets, and interactive textbooks that support mathematics education without the overhead of a full blown computer algebra system, it makes sense to ask whether from the educator’s viewpoint whether there is still a case for teaching high school or lower-division undergraduates “regular old Maple”.

In this paper, we argue that there is still a case to be made for doing so. We review the goals of teaching Maple or similar computer algebra systems (CAS), and make observations from our recent experience teaching technical computation to freshman engineering students about problems that persist in teaching these systems to a wide audience.

2 Teaching to freshmen in 1987 and 2008 – a comparison

The mid-’80s was the “voyage of discovery” by many mathematical educators, as they became aware of the capabilities of computer algebra systems (CAS) and considered ways of using them in their instruction of undergraduates. CAS promised to remove the emphasis of hand computation allowing more time devoted to conceptual understanding and the ability to include less routine and more realistic computational examples and problems (see [2] for an early discussion of the use and promise of CAS in education, and [10] for a discussion of the role of CAS in calculus reform). Early advocates, such as Hosack, Lane, and Small [7] speculated that “Perhaps an experimental, exploratory, approach towards mathematics can be fostered, where the students study examples looking for patterns and framing hypotheses.”

A large amount of materials were developed for classroom use; some of these are still currently in print or in use. Some made heavy use of CAS, offered major curricular changes and provided many tools and examples to allow students to experimentally explore mathematical concepts [9, 11, 3, 4, 6]. Many mainstream North American calculus textbooks now include computer or calculator-based exercises.

In those days, it was easy to justify the use of Maple or similar CAS for mathematics instruction because they were the only way to get symbolic computation or easy graphing capabilities. As alternatives such as handheld calculators or applets have grown in power and convenience, it has become important to rethink the justification for why one should use a CAS on a computer for undergraduate instruction.

2.1 The difficulties of sustained use of a CAS in a mathematics course

Use of Maple-style computer algebra has in many cases faded from use in undergraduate math education. We list some of the reasons we think this has happened:

1. Undergraduate math courses don't have time to teach substantial technology; they're already filled with important math content.

2. Additional (technical) expertise is needed to teach effectively with technology. Many mathematics instructors do not have such expertise and see that there are many paths towards being an effective instructor which don't involve much use of technology.

3. Graphing calculators with symbolic capabilities such as the TI 89 or HP 50g make limited (but perhaps the most valued) portions of CAS available in a highly portable and relatively inexpensive form. These devices are in common use in high school and students are comfortable using them.

4. Applets allow limited CAS functionality with almost no training with "point and click" operation. If the point is to explore a particular math feature, it's much easier to use them in a class.

5. Many students trying to use a CAS in a calculus course are not secure in the mathematics that is the subject of the computing. A course that uses both Maple and new calculus ideas has two different sources of confusion – mathematical concepts and notation, and computing concepts and Maple notation.

In our experience, Maple-using math courses aimed at more advanced undergraduates seem to have far fewer difficulties.

2.2 Teaching Maple in the present day

The justification for the present day is more complicated but also, we believe, likely to stand the test of time. Students should learn Maple or similar systems not only because they can help do homework problems in undergraduate mathematics, but for the same reasons that the "grown ups" use them – they speed the development of insight into a technical problem. The long-term user of a CAS stays with it because the system has *extensive mathematical functionality* in a *scripted, interpretive environment.* The value of Maple can be the speed with which a script can be developed to do a particular computation, and the ease with which it can be modified or turned into a library to support a continuing investigation. The effort to teach Maple becomes rejustified because it is a *tool with on-going value for a technical professional because it has the flexibility to cover a wide variety of situations and is a productive environment to work in.*

We believe that shifting the justification of CAS use to the long-term value of knowing and using one changes pedagogical goals. Rather than trying to bypass as much of the technical complexity as possible so as to get to mathematical content more quickly, one should take the time to allow the development of experienced productive use. This means:

1. Being able to handle major scripting features such as iteration, conditionals, and procedure construction, as well as acquiring sensibility about script development – testing, readability, and resource consumption.

2. Being able to learn new system features from documentation.

3. Having a developer's knowledge of the major uses of scripted technical computation systems in professional use – quick modeling or formula derivation, as a way of documenting results, and as a portal to other applications systems.

4. Being able to transfer the tools and experimental approach to mathematics and problem solving to other classes, and to "real world" problems.

Here at Drexel we have regularly taught courses using Maple to explore a wide range of topics ranging from calculus, differential equations, discrete mathematics, and numerical analysis to code generation, cryptography, and the analysis of algorithms. While early efforts of incorporating Maple into freshman calculus courses [1] had only moderate success and were largely discontinued, upper level computer science students are regularly exposed to Maple as a computational mathematics tool and find it useful in exploring and learning mathematical concepts. Recently [8], we have attempted to bring back the use of Maple to the freshman curriculum and to make it widely used tool for all engineering students. However, this time a separate year-long computation lab (one credit per term), outside of the calculus sequence, was created with the goal of teaching engineering students the general skills to successfully use Maple across the curriculum and beyond.

2.3 Continuing difficulties with using regular Maple with novice users

In our Engineering Computation Lab course, we deal with the mathematics that students have already had time to become familiar with – high school algebra only, in the first term; differential calculus in the second term after the initial calculus course has concluded, etc. Furthermore, the instructors are computer scientists who do not have to make such an effort to become familiar with Maple. Nevertheless, even without the difficulties of section 2.1, we saw several sources of learning difficulties.

1. Maple syntax isn't the same as the syntax of math books.
2. *Maple is a language.* Most introductory-level students have not had programming. Giving Maple commands leads to a new experience: having a computer refuse to accept their orders because they don't have the proper spelling, order, or case.
3. *Students need more automated help.* Since we emphasize construction of scripts and programs in our course, we want support comparable to what GUI-based IDEs provide: auto completion, automatic indentation, highlighted keywords, menu-driven function completion, a point-and-click debugger.
4. *What is a wrong answer?* Maple has improved from the 1980's, where typically error messages indicated that a problem had been detected, but were not very helpful about where or what. The amount of time we spend pointing out missing semi-colon problems in lab has dramatically dropped from twenty years ago, for example. What remains a problem, however, is that Maple's sophisticated and powerful programming language allows plausible mistakes by beginners to lead to expressions that cause no errors or warnings but differ substantially from what the student intended. Understanding the error usually requires a level of computer expertise far beyond that of beginners. For example, in our class we try to get the students to define short functions in Maple using arrow notation, e.g.:

   ```
   f := (x,y,z) -> (x+y+z)/3;
   ```

   If they are successful, they find having such functions very useful. However, we have found that it isn't easy to get the students how to notice whether they've correctly entered the function they intended, nor to explain why something similar is accepted but produces drastically different results. For example, if a student enters `f2 := x,y,z -> (x+y+z)/3;` , the response looks very similar to the response for `f` above. However, f2(5,6,7) doesn't produce what the student expects while f(5,6,7) does.

   Much of the output from Maple is novel or unfamiliar to them; telling which ones are "wrong" and which ones are "right" takes more experienced-based discrimination than beginners typically have.

   We believe that Maple would be more hospitable to beginners with a greater collection of "Did you really mean to do that?" tests in an expertise level-appropriate way. There should be support for people to write scripts in worksheets with testing in mind.
5. *Maple user interface has too many degrees of freedom for beginners to master easily.*

   The Maple worksheet has six different kinds of text to enter, all resulting in different kinds of actions. Commands will be processed all together, or need multiple keystrokes to execute depending on whether they are in the same execution group. These kinds of features have made learning Maple command execution a non-trivial task.

6. *Maple on-line help is not appropriate for introductory-level students.* We don't see how any modest amount of curricular materials can bring novices up to the level of sophistication needed to deal with the documentation on standard Maple features such as plotting, if statements, or solving. Not only does the documentation often talk about mathematics that is incomprehensible to typical freshmen (the discussion of RootOfs in the solve documentation comes to mind), it is written at a level more appropriate to technical professionals (see, for example, the documentation in Maple 11 for "if").

3 Conclusions

We believe that powerful calculators and applets have greatly reduced the need to teach CAS in order to learn basic undergraduate mathematics. Nevertheless, Maple has appeal to those would learn undergraduate technical computation because in use of one system can provide experiences with interactive script development, standard procedural programming, and presentation of technical results while still having the option to deal with both symbolic and numeric mathematics and a reasonable built-in collection of abstract data structures and math libraries.

While the content and functionality is appealing, the vastness of Maple's syntax, its sophisticated programming concepts and math expertise makes it a difficult system to teach to beginners. This is complicated by its "by professionals for professionals" stance, not withstanding the Student math packages. We have made observations during our paper about interface, documentation, and "domain of discourse" adjustments that will make life easier for novices and those who would teach them.

4 Acknowledgments

We wish to acknowledge the support of the Drexel University College of Engineering, NSF Grant IERI-0325872 and the helpful comments of the anonymous referees.

References

[1] Loren Argabright and Robert Busby. *Calculus Workbook using "Maple"*. Kendall Hunt Publishing Company, 2 edition, 1993.

[2] Bruno Buchberger and David Stoutemeyer, editors. *Report on the Work of Group 3.1.4 on Symbolic Mathematical Systems and Their Effects on the Curriculum*, volume 19. ACM SIGSAM Bulletin, 1984.

[3] Texas A& M University Department of Mathematics Instructional Laboratories - The Calclabs. `http://calclab.math.tamu.edu`.

[4] Calculus & Mathematica At the University of Illinois Urbana-Champaign. `http://www-cm.math.uiuc.edu`.

[5] B. W. Char, K. O. Geddes, G. H. Gonnet, B. J. Marshman, and P. J. Ponzo. Computer algebra in the undergraduate mathematics classroom. In *SYMSAC '86: Proceedings of the fifth ACM symposium on Symbolic and algebraic computation*, pages 135–140, New York, NY, USA, 1986. ACM.

[6] George Tech School of Mathematics Core Curriculum Course Materials. `http://www.math.gatech.edu/~bourbaki`.

[7] John Hosack, Kenneth Lane, and Donald Small. Report on the use of symbolic mathematics system in undergraduate instruction. *SIGSAM Bulletin*, 19:19–22, 1984.

[8] Jeremy Johnson. Development of a calculus based computation lab - an algorithmic approach to calculus. In Ilias Kotsireas, editor, *Maple Conference 2006*, 2006.

[9] Math archives. `http://archives.math.utk.edu`.

[10] Lisa Denise Murphy. Computer algebra systems in calculus reform. `http://www.mste.uiuc.edu/users/Murphy/Papers/CalcReformPaper.html`.

[11] Project CALC: Calculus as a laboratory course. `http://www.math.duke.edu/education/proj\_calc`.

Numerical Analysis with Maple[1]

Mirko Navara[2] and Aleš Němeček[3]

Abstract

We summarize more than 10 years of our experience with a course of Numerical Analysis with the use of Maple. Besides software packages, we discuss also the principles of education.

History of the course

Numerical Analysis with computer support has been taught at our university since late 80's. During the first years, the main subject was programming of numerical methods in Pascal. The students gained experience with common errors in numerical programming, however, this reduced the time spent with the use of these methods. Students were usually satisfied when the program worked and they considered it useless to make further experiments with it.

The skills of our incoming students do not allow to put sufficient emphasis on both programming and the use of the programs. We have decided to concentrate on the latter. Thus we prepared programs which perform the studied methods at least in their standard form. Then we used MathCAD on Apple Macintosh computers for several years.

In 1994 the Numerical Methods course changed significantly with the introduction of Maple for demonstrations and calculations. Students can use Maple worksheets which implement standard algorithms. They are expected to extend them for the use in non-standard situations requiring some additional hints to solve the given tasks.

We decided to use open Maple worksheets whose structure is visible and can be modified arbitrarily. As an option, we considered object-oriented programs (particularly impressive in Maple 11 document style or Maplets) which allow to handle all possibilities by several components and buttons. (Both approaches have been tested in a novel course of Multivariate Calculus in winter semester 2007/08.) We have decided not to follow this line because students should see what is behind their commands. Thus the source code has to be visible and subject to potential change (on the students' own risk). Maple worksheets are subject to permanent updates and improvements.

The course still has a theoretical core of lectures that cover definitions and theorems with proofs. This helps students see properties and connections of methods and design of specific algorithms for computers. This core is demonstrated by graphical presentation of methods including animations.

Principles of the current course

Our present course is based on programs which are modified by students. Having too short time for programming (the total length of the course is 28 hours of lectures and 28 hours of work in computer laboratory), the students are not expected to build up completely new programs, but adapt basic algorithms to the needs of specific tasks, i.e., to extend standard tools to non-standard situations.

The use of floating point operations is natural in Numerical Analysis. In Maple, symbolic computation had to be suppressed in order to demonstrate the properties of numerical algorithms and round-off errors. On the other hand, symbolic algebra is useful in derivation of estimates of errors and modifications of problems (e.g., substitution in integration). Last but not least, graphical facilities of Maple are useful for demonstration of results and view of the properties of methods which are mathematically correct, but possibly inappropriate

[1] The first author acknowledges the support by the Czech Government under the reseach program MSM 6840770038. The second author is supported by project *Interactive Information Portal Construction for Research Applications*, No. 1N04075.

[2] Czech Technical University, Faculty of Electrical Engineering, Department of Cybernetics, Center for Machine Perception, Technická 2, 166 27 Prague 6, Czech Republic, Phone: +420 224357388, Fax: +420 224357385, e-mail: navara@cmp.felk.cvut.cz

[3] Czech Technical University, Faculty of Electrical Engineering, Department of Mathematics, Technická 2, 166 27 Prague 6, Czech Republic, Phone: +420 224353482, Fax: +420 233339238, e-mail: nemecek@math.feld.cvut.cz

in a particular application. We put emphasis on practical experience with numerical methods. The two basic principles of our course are:

1. Do not trust all results obtained by a computer. Verify them by alternative solutions and tests.
2. Learn what to do when the results are correct for the method chosen, but different from your expectation.

These principles might look too crude, but we need to emphasize them for users who believe in technology and computer results too much. (We often observe this attitude of our students.) Weaker forms of this statment (by Peter Henrici and Nick Trefethen, originally due to Wilkinson, see [1]) are: "A good numerical method gives you the exact solution to a nearby problem," or "Some problems are sensitive to changes." Nevertheless, we remind our students that it is them who will sign the final results and take responsibility (neither the computers, nor the authors of software). Beside a large majority of correct results, some errors are so crucial that they cannot be explained as mere "sensitivity to changes." It is important that the graduates recognize these situations.

We had to make a crucial decision:

Should the students follow a plan prepared in detail, or improvise on their individual problems?

We decided to implement the latter – more general and less restricted – approach. Besides problems suggested by the teachers, the students may apply the methods on data obtained in other courses (e.g., Electrical Measurements) and encounter situations not planned by the teacher. This approach requires less preparation, but more improvisation of the teachers during the course. Still our worksheets may be useful for standard solutions and their presentation. They contain input and output interfaces, choice of methods, algorithms (with optional levels of information about intermediate results), error estimates, graphical outputs, and mostly also comparison with the standard tools of Maple (which sometimes succeed and sometimes fail to solve the tasks correctly). Because we teach future electrical engineers, we collected a database of problems motivated by electrical circuits and measurements or by general physics.

Contribution of Maple

The greatest changes in education were introduced at seminars. These take place in a computer laboratory using Maple. The first classes (four hours) are devoted to a quick introduction to the system (work environment and commands) which is new to most students.

We have prepared modules (Maple worksheets) for all numerical methods that are covered by the course. These contain not only the necessary algorithms, but also selection of input formats and solving methods, error estimate construction section, graphical presentation of results, sometimes also a comparison with the precise (symbolical) solution. Some exercises allow to compare the numerical results with those of standard Maple procedures. Maple solutions are only sometimes satisfactory, the students have to compare different methods and make their own conclusions about the validity of results and error estimates. Whenever possible, we present the graphs of absolute and/or relative errors and their estimates.

In some cases, not only the choice of method, but also its proper application to the given task is important. E.g., we created a collection of difficult exercises on numerical integration which require modifications for obtaining sufficiently precise results. A change may lead to a task which is mathematically equivalent, but its numerical error is of a different order. The students are expected to find such tricks and validate the achieved precision. Standard methods are usually insufficient to solve these tasks by brute force.

Even the seminar work is assigned in electronic form. During the semester the students receive five files in their home directories (the ownership properties are set in such a way that students cannot change them). The text is already in the proper format which can be directly loaded into a Maple worksheet. They see the formulation of the problem and at the same time variables are assigned proper values needed for further calculations. Students can solve their assignments in the computer laboratory with the help of an instructor or at home[4]. The seminar work is collected in the lab, where the student presents the calculations and the examiner has a chance to ask questions to see whether the student really understands the subject. Besides, Maple allows to check particular outputs of the algorithm and other criteria needed to decide whether the implemented method really solves the task. The course is supported by a textbook [3] (in Czech) which was also prepared with the help of Maple (examples, figures, etc.). The choice of topics covers approximation, numerical differentiation and integration, roots finding, and differential equations. We follow mainly the approach of [5] with reference to [2, 6]. Emphasis is put on approximation. This topic is frequently used by graduates and it allows a wide use of computer graphics in experiments. Maple allows to demonstrate the results quickly and thus obtain experience with numerous methods. The students see that even correct solutions of a mathematical task may be of low practical value if the choice of methods was inappropriate. Sample worksheets for approximation and differential equations are shown in Appendix B.

It is no exaggeration to say that the students leave the course with practical skills at a much higher level than before the introduction of this type of course. The subject matter is thus much clearer and exercises from older textbooks seem a distant memory.

Links to other activities

As a support, we established also a course of Computer Algebra Systems where we teach different CAS's (Derive, Maple, Mathematica, Matlab) with emphasis on Maple. This course is intended for students who want to learn more about software tools and apply them as regular instruments for their engineering work. It also offers them a comparison of different computer algebra systems and their facilities.

Some of our students use Maple extensively in their diploma and PhD theses as a tool for scientific computing. Besides, we supervised several semestral projects and one diploma thesis [7] dealing only with the use of Maple as a computational environment.

In a separate lecture within the course of Computer Algebra Systems, we also summarize advantages and drawbacks of the CAS's used. A collection of benchmark problems [4] has been developed for this purpose with a help of [7, 8] and others.

Keywords: Numerical Analysis, Maple, classroom materials, student training.

Intended audience: Teachers of mathematics at undergraduate level, Maple users interested in numerical methods.

[4] In 1996 and several subsequent years, our activity was supported by a grant which, among others, allowed to pay a multilicence of Maple. It allows for installation of Maple on home computers of students participating in education or projects related to Maple.

References

[1] Corless, R.: AM372/272: Numerical Analysis for Engineers, AM361b Numerical Analysis for Scientists. Course curricula, 2008, `http://www.apmaths.uwo.ca/ rcorless/BIO/dossier/node13.html`

[2] Knuth, D. E.: *Fundamental Algorithms.* Vol. 1 of *The Art of Computer Programming,* Addison-Wesley, Reading, MA, 1968.

[3] Navara, M., Němeček, A.: *Numerical Analysis* (in Czech). Czech Technical University, Prague, 2005.

[4] Navara, M., Němeček, A.: Long-term experience with Maple: Advantages and challenges of Maple 10. Book of Proceedings, Maple Conference 2006, Wilfrid Laurier University, Maplesoft; Waterloo, Ontario, Canada, 2006, 353–354.

[5] Press, W. H., Flannery, B. P., Teukolsky, S. A., Vetterling, W. T.: *Numerical Recipes (The Art of Scientific Computing).* Cambridge University Press, Cambridge, 1986.

[6] Stoer, J., Bulirsch, R.: *Introduction to Numerical Analysis.* Springer Verlag, New York, 2000.

[7] Vrba, L.: *Comparison of Computer Algebra Systems* (in Czech). Diploma thesis, Czech Technical University, Prague, 1999.

[8] Wester, M.: *A Review of CAS Mathematical Capabilities.* Preprint, University of New Mexico, Albuquerque, 1994.

Appendix B

Numerical solution of ordinary differential equtions

Cauchy's initial value problem

Program description

Global variables:

Input:

f, x0, y0 ... we solve problem *y'(x)=f(x,y(x))* with initial value condition *y(x0)=y0*

infolevel[res] ... print level of information to the user

Output:

x, y ... lists (0..k) of coordinates of points of the last solution

h ... step for the last solution

Common method of numerical solution

xk ... final point

k ... number of intervals

metoda1 ... main method of solution

metoda2 ... single-step method

metoda3, metoda4 ... predictor in case of PECE method

```
>
```

Program inicialization

Assignment

Read data from file

Right-hand side

```
> f := (x,y) -> 1+y^2;
```

$$f := (x, y) \rightarrow 1 + y^2$$

Initial value

```
> x0:=0;
> y0:=0;
> xk:=1.5;
```

$$x0 := 0$$

$$y0 := 0$$

$$xk := 1.5$$

```
>
```

Graphical representation

Calculation and examples

User information level

Set the desired level and evaluate only one row.

```
> infolevel[res]:=1;
```
only solution *y*

```
> infolevel[res]:=4;
```
" + results of individual steps *x, y*

```
> infolevel[res]:=6;
```
" " + results of the predictor *x, predx*

```
> infolevel[res]:=8;
```
" " " + right-hand sides + (directions) *k1, k2, ...*

```
> infolevel[res]:=12;
```
" " " " + general direction kk

```
> infolevel[res]:=16;
```
" " " " " + instrumental coordinates *y1, y2, ...*

$$infolevel_{res} := 16$$

Demonstration examples

```
> k:=5;
```

$$k := 5$$

Single-step method (Euler)

```
> res(xk, k, RK, Euler);
```

```
res:, `x =`, 0, `y =`, 0
Euler:, `k1 =`, 1
res:, `x =`, .3000000000, `y =`, .3000000000
Euler:, `k1 =`, 1.090000000
res:, `x =`, .6000000000, `y =`, .6270000000
Euler:, `k1 =`, 1.393129000
res:, `x =`, .9000000000, `y =`, 1.044938700
Euler:, `k1 =`, 2.091896887
res:, `x =`, 1.200000000, `y =`, 1.672507766
Euler:, `k1 =`, 3.797282227
res:, `x =`, 1.500000000, `y =`, 2.811692434
```

$$2.811692434$$

Explicit method (Adams-Bashforth method 2nd order, start - Euler's Method - 1st modification)

```
> res(xk, k, AB2, EulerMod1);
```

```
res:, `x =`, 0, `y =`, 0
EulerMod1:, `y2 =`, .1500000000
EulerMod1:, `k1 =`, 1, `k2 =`, 1.022500000
res:, `x =`, .3000000000, `y =`, .3067500000
res:, `x =`, .6000000000, `y =`, .6490930029
res:, `x =`, .9000000000, `y =`, 1.124573445
res:, `x =`, 1.200000000, `y =`, 1.930474631
res:, `x =`, 1.500000000, `y =`, 3.717804351
```

$$3.717804351$$

Predictor-corrector method (corrector - Adams-Moulton method 4th order, start - Runge-Kutta method 3rd order; predictor - Heune's method)

```
> res(xk, k, AM4, RungeKutta3, RK, EulerMod2);
```

```
res:, `x =`, 0, `y =`, 0
RungeKutta3:, `y2 =`, .1500000000, `y3 =`, .3135000000
```

```
RungeKutta3:, `k1 =`, 1, `k2 =`, 1.022500000, `k3 =`, 1.098282250
RungeKutta3:, `kk =`, 1.031380375
res:, `x =`, .3000000000, `y =`, .3094141125
RungeKutta3:, `y2 =`, .4737746765, `y3 =`, .7153704510
RungeKutta3:, `k1 =`, 1.095737093, `k2 =`, 1.224462444, `k3 =`, 1.511754\
882
RungeKutta3:, `kk =`, 1.250890292
res:, `x =`, .6000000000, `y =`, .6846812001
EulerMod2:, `y2 =`, 1.125317704
EulerMod2:, `k1 =`, 1.468788346, `k2 =`, 2.266339935
EulerMod2:, `kk =`, 1.867564141
AM4:, `x =`, .9000000000, `ypred =`, 1.244950442
res:, `x =`, .9000000000, `y =`, 1.264398794
EulerMod2:, `y2 =`, 2.044010087
EulerMod2:, `k1 =`, 2.598704310, `k2 =`, 5.177977236
EulerMod2:, `kk =`, 3.888340773
AM4:, `x =`, 1.200000000, `ypred =`, 2.430901026
res:, `x =`, 1.200000000, `y =`, 2.580782488
EulerMod2:, `y2 =`, 4.878913963
EulerMod2:, `k1 =`, 7.660438250, `k2 =`, 24.80380146
EulerMod2:, `kk =`, 16.23211986
AM4:, `x =`, 1.500000000, `ypred =`, 7.450418446
res:, `x =`, 1.500000000, `y =`, 10.61331010

                                 10.61331010
```

Symbolic solution

```
> MSymb(xk);
```

MSymb(xk, k);   # with the possibility of additional informations

Numerical solution

MNum(xk);

```
> MNum(xk, k);     # with ability additionaly informations
```

MNum(xk, k, dverk78);   # set other methods, see Help

```
>
```

Graphical demonstration of the last solution

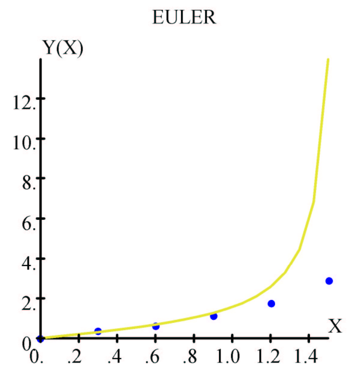
```
> plots[setoptions](thickness=1, font=[TIMES, ROMAN, 5]);
```

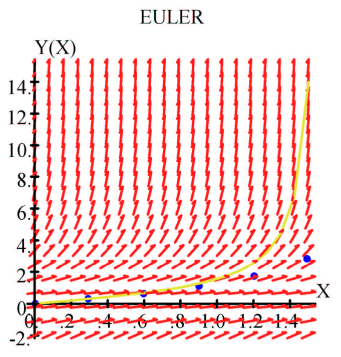
```
>
```

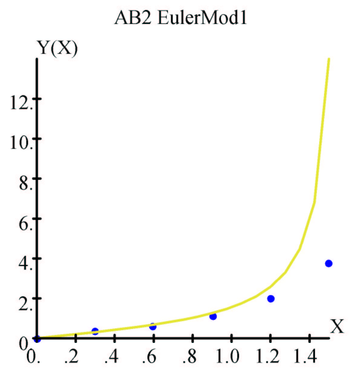
Tile horizontally

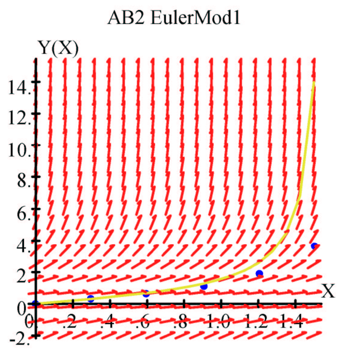
Tile vertically

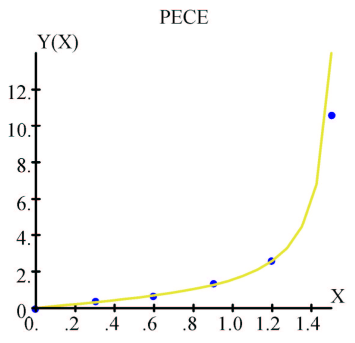
```
> reseni:=plotxy(x, y, k, style=POINT, symbol=POINT, color=BLUE,
  title="PECE", titlefont=[TIMES, ROMAN, 5]):
> p1 := plots[display]([presres, reseni]):
> p2 := plots[display]([pole, reseni]):
> plots[display](array(1..2, [p1, p2]));
```

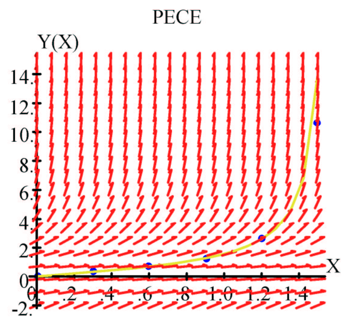












>

Go to demonstration examples.

Systematic Tensor Simplification: a Diagrammatic Approach

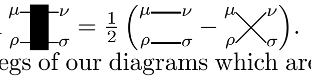
A. D. Kennedy and T. Reiter

October 22, 2008

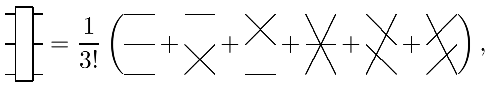
Simplification of tensor expressions is important in many applications of computer algebra, and many systems have been written to undertake this task. It is crucial to make use of the all the symmetries of an expression both to reduce the computational complexity of the simplification algorithm and the size of the simplified expression. Most if not all existing systems do this using various heuristic approaches, including Keith Geddes' contributions to the subject [6, 7, 5, 3, 4]: we propose instead a systematic non-heuristic approach using completeness and recoupling relations for the irreducible representations of the appropriate symmetry group. This reduces any tensor expression into a sum of basis tensors, corresponding to tree graphs in our diagrammatic notation [1], with coefficients that are rational functions of 0–j (dimensions), 3–j, and 6–j coefficients. These n–j coefficients are readily computed and reused for all symmetry groups of interest, and for the case of $\mathrm{S}\ell(N)$ we give a new efficient algorithm for computing them.

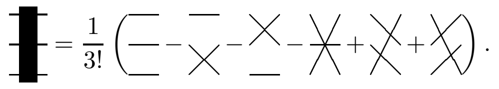
Tensor calculations, in their traditional form, are plagued by a proliferation of indices. Graphical representations of tensor expressions have been introduced not only as a visualisation but also as a calculational tool. We follow the notation of Cvitanović's book [1], where tensors are represented as the vertices of a graph and their indices are represented by its edges. A complicated tensor expression can thus be completely encoded into a diagram, similar to Feynman diagrams.

The aim of tensor reduction in this diagrammatic context is to represent an arbitrary tensor expression, i.e., an arbitrary diagram, as a sum of trees times group-theoretic invariants. This is always possible due to the Wigner–Eckhart theorem. In order to do this systematically we construct a set of basis tensors consisting only of irreducible representations by applying completeness relations (Clebsch–Gordan series) to the given tensor expression. This decomposes the expression into a sum over primitive tensors (tree diagrams) that carry the index structure of the tensor expression times scalar coefficients that are represented by bubble diagrams without any external legs.

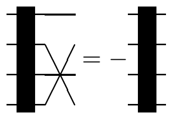
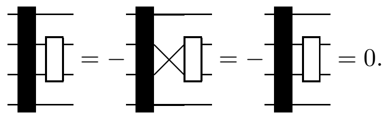
In our diagrammatic notation we represent a Kronecker tensor δ_μ^ν by a line, so the antisymmetrizer $\frac{1}{2}\Big(\delta_\mu^\nu\delta_\rho^\sigma - \delta_\mu^\sigma\delta_\rho^\nu\Big)$ is represented by the diagram 

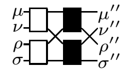
. In general we do not label any of the indices: free indices correspond to external legs of our diagrams which are the same in each term, and dummy indices correspond to internal legs. We denote a symmetrizer by an open box and an antisymmetrizer by a solid one, so for example



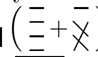
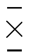
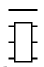
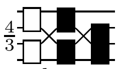
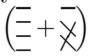
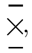


These satisfy identities like

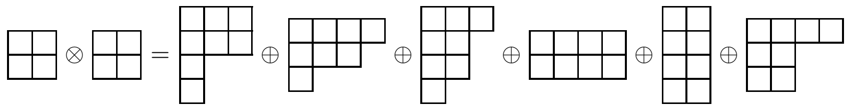
 and hence 

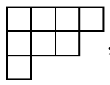
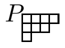
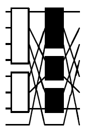
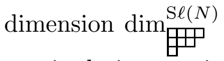
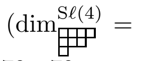
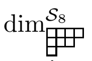
To illustrate how we construct a basis of irreducible tensors consider the Riemann tensor $R_{\mu\nu\rho\sigma}$, which corresponds to the $\mathrm{S}\ell(N)$ irreducible representation labelled by the Young diagram $\boxplus$. The diagrammatic projection operator onto the subspace carrying this representation is $P_{\boxplus} = \frac{4}{3}$  , or in tensor notation

$$\frac{4}{3}\cdot\frac{1}{2}\left(\delta_{\mu}^{\mu'}\delta_{\nu}^{\nu'}+\delta_{\mu}^{\nu'}\delta_{\nu}^{\mu'}\right)\cdot\frac{1}{2}\left(\delta_{\rho}^{\rho'}\delta_{\sigma}^{\sigma'}+\delta_{\rho}^{\sigma'}\delta_{\sigma}^{\rho'}\right)\cdot\frac{1}{2}\left(\delta_{\mu'}^{\mu''}\delta_{\rho'}^{\rho''}-\delta_{\mu'}^{\rho''}\delta_{\rho'}^{\mu''}\right)\cdot\frac{1}{2}\left(\delta_{\nu'}^{\nu''}\delta_{\sigma'}^{\sigma''}-\delta_{\nu'}^{\sigma''}\delta_{\sigma'}^{\nu''}\right).$$

This projector not only corresponds to an irreducible representation of $\mathrm{S}\ell(N)$ but also to one of the symmetric group $\mathcal{S}_4$, so the tensor expression $R_{\mu\nu\rho\sigma}+R_{\mu\rho\sigma\nu}$ corresponds to the sum of permutations $P_{\boxplus}$ . The representation of $\mathcal{S}_4$ is 2 dimensional, with a basis corresponding to the standard tableaux $\begin{array}{|c|c|}\hline 1&2\\\hline 3&4\\\hline\end{array}$ and $\begin{array}{|c|c|}\hline 1&3\\\hline 2&4\\\hline\end{array}$, i.e., $P_{\boxplus}$ and $P_{\boxplus}$  , so all permutations acting on $P_{\boxplus}$ are expressible as sums of these two basis elements, and this reduction may be carried out using the Garnir relation [2, 10] $P_{\boxplus}$  $= \frac{4}{3}$  $= 0$ (if the central column of antisymmetrizers are expanded into a sum of permutations then in each term two legs from the antisymmetrizer on the right must be connected to the same symmetrizer on the left). We thus find that $P_{\boxplus}$  $= -P_{\boxplus}$ , so $R_{\mu\nu\rho\sigma}+R_{\mu\rho\sigma\nu} = -R_{\mu\rho\nu\sigma}$.

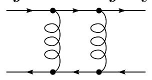
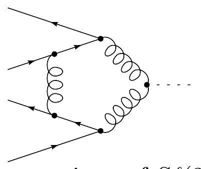
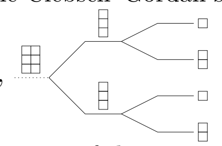
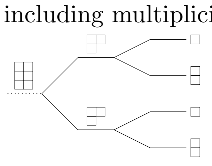
If we want to simplify the product of two Riemann tensors $R_{\mu\nu\rho\sigma}R_{\alpha\beta\gamma\tau}$ then we use the Littlewood–Richardson theorem to construct the Clebsch–Gordan series which allow us to express the product in terms of irreducible parts

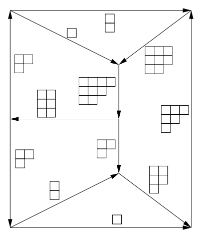


where the representation , for example, corresponds to the projector  $= 6$  and has dimension  $= \frac{(N^2-4)(N^2-1)N^2(N+1)(N+3)}{576}$  $= 175$) and  $= 70$. Our indicial tensor manipulation requires explicit manipulation of the 70×70 representation matrices, but this is to be compared with having to manipulate $8! = 40,320$ terms if we were to expand all the Young operators).

As there is an invariant metric tensor $g_{\mu\nu}$ the Riemann tensor is reducible with respect to the symmetry group $\mathrm{SO}(N)$, and thus we can reduce the $\mathrm{S}\ell(N)$ representation further into the traceless Weyl and Einstein tensors and a scalar.

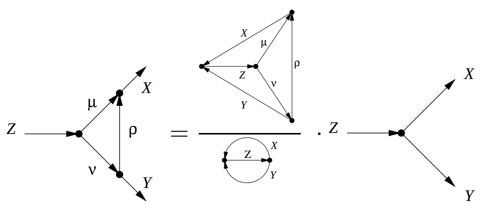
The choice of basis (trees) is not unique, and a choice conforming to the known symmetries of the problem is clearly wise. Transforming from one such basis to another is easily done using recoupling relations that involve 3–j and 6–j symbols.

Our second simple example of a tensor reduction illustrates how 3–j and 6–j symbols arise. Consider the Sℓ(3) colour structure of the following Feynman diagram in QCD  . This may be deformed into  , which may be considered as the reduction of the tensor product of irreducible representations of Sℓ(3) to a scalar. In this diagram two quark-antiquark pairs (solid lines) are coupled to produce a scalar (dotted line) via the exchange of gluons (springs). These irreducible representations of Sℓ(3) may be labelled by Young diagrams which indicate how they may be constructed from tensor products of quark (fundamental) representations: a quark (fundamental) is $\yng(1)$, an antiquark is $\yng(1,1)$, and a gluon (adjoint) is $\yng(2,1)$. The Littlewood–Richardson rule allows to enumerate all possible ways of coupling the two quark-pairs to a scalar, i.e., the Clebsch–Gordan series including multiplicity. In our example there are only two possible (balanced) trees,  and  . The bubble diagram (12–j coefficient) resulting from the projection of the Feynman diagram onto the second tree is

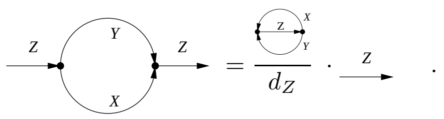
 .

Note that any column of height 3 could be omitted for Sℓ(3) in principle, but we our methods allow us to compute the value of this 12–j coefficient as an explicit rational function of N for S$\ell(N)$.

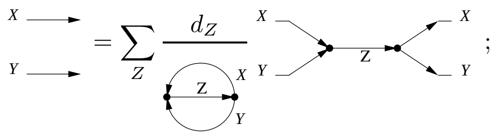
In order to reduce the bubble diagrams into 0–j, 3–j and 6–j coefficients we select the shortest cycle by using the LLL algorithm [8, 9] and eliminate it: cycles of length two and three are can be eliminated directly via



which contains a 6–j symbol in the numerator, or by Schur's lemma



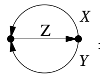
Longer loops can be broken up by pinching two opposite edges of the loop using the completeness relation



where the summation runs over all irreducible representations Z which X and Y can couple to. In the preceding example the reduction terminates after applying the star-triangle relation twice.

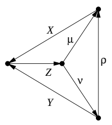
It turns out that the computationally most expensive part of the reduction is the calculation of the 3–j and 6–j symbols, especially when large $\mathcal{S}_k$ representations are involved. However, the advantage of our approach is the fact that these coefficients can be computed once and for all as a rational function of N.

For the calculation of the 3–j and 6–j coefficients in $\mathrm{S}\ell(N)$ we exploit the observation that there is a unique irreducible representation Z whose a Young diagram has the most boxes. The $\mathrm{S}\ell(N)$ n–j coefficients thus factorise into the $\mathrm{S}\ell(N)$-dimension of this representation Z and a trace of products of projection operators in the symmetric group representation corresponding to Z,



$$= \mathrm{tr}_{\mathcal{S}_Z}(P_X P_Y P_Z)\,\mathrm{tr}_{\mathrm{S}\ell(N)}(P_Z) = \mathrm{tr}_{\mathcal{S}_Z}(P_X P_Y P_Z)\,\mathrm{dim}_Z^{\mathrm{S}\ell(N)}$$

and



$$= \mathrm{tr}_{\mathcal{S}_Z}(P_\mu P_\nu P_\rho P_X P_Y P_Z)\,\mathrm{tr}_{\mathrm{S}\ell(N)}(P_Z) = \mathrm{tr}_{\mathcal{S}_Z}(P_\mu P_\nu P_\rho P_X P_Y P_Z)\,\mathrm{dim}_Z^{\mathrm{S}\ell(N)}\,.$$

We construct the $\mathcal{S}_Z$ representation matrices of the Young projectors as a product of row-symmetrizers and column-antisymmetrizers, these being easy to construct recursively from the matrices representing transpositions, and we use Garnir relations to construct the representations of the such transpositions.

We are currently developing a Python [11] package that implements the algorithms described herein; a stable version is expected to be released this summer.

References

[1] Pedrag Cvitanović. *Group Theory: Birdtracks, Lie's, and Exceptional Groups.* Princeton University Press, Princton, 2008. To appear in July 2008.

[2] Henri Georges Garnir. Théorie de la représentation linéaire des groupes symétriques. *Mém. Soc. Roy. Sci. Liége*, 10(4), 1950.

[3] M. Kavian, R.G. McLenaghan, and K.O. Geddes. Mapletensor: A new system for performing indicial and component tensor calculations by computer. In *Proceedings of the 7th Marcel Grossman Conference*, Singapore, 1995. World Scientific.

[4] M. Kavian, R.G. McLenaghan, and K.O. Geddes. Mapletensor: A new system for performing indicial and component tensor calculations by computer. In *Proceedings of the 14th International Conference on General Relativity and Gravitation*, Florence, Italy, 1995.

[5] M. Kavian, R.G. McLenaghan, and K.O. Geddes. Mapletensor: Progress report on a new system for performing indicial and component tensor calculations using symbolic computation. In Lakshman Y.N., editor, *Proceedings of ISSAC'96*, pages 204–211, New York, 1996. ACM Press.

[6] M. Kavian, R.G. McLenaghan, and K.O. Geddes. Application of genetic algorithms to the algebraic simplification of tensor polynomials. In W.W. Kuechlin, editor, *Proceedings of ISSAC'97*, pages 93–100, New York, 1997. ACM Press.

[7] M. Kavian, R.G. McLenaghan, and K.O. Geddes. Mapletensor: A new system for performing indicial and component tensor calculations by computer. *Fields Institute Comm.*, 15:269–272, 1997.

[8] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.

[9] Maurice Mignotte. *Mathematics for Computer Algebra.* Springer-Verlag, New York, 1992.

[10] Bruce Eli Sagan. *The symmetric group: representations, combinatorial algorithms, and symmetric functions.* Springer-Verlag, New York, 2001.

[11] Guido van Rossum and Fred L. Drake. Python tutorial release 2. 1.1.

Max-Plus Linear Algebra in Maple and Generalized Solutions for First-Order Ordinary BVPs via Max-Plus Interpolation

Georg Regensburger*

Abstract

If we consider the real numbers extended by minus infinity with the operations maximum and addition, we obtain the max-algebra or the max-plus semiring. The analog of linear algebra for these operations extended to matrices and vectors has been widely studied.

We outline some facts on semirings and max-plus linear algebra, in particular, the solution of max-plus linear systems. As an application, we discuss how to compute symbolically generalized solutions for nonlinear first-order ordinary boundary value problems (BVPs) by solving a corresponding max-plus interpolation problem. Finally, we present the Maple package `MaxLinearAlgebra` and illustrate the implementation and our application with some examples.

1 Semirings and Idempotent Mathematics

The *max-algebra* or *max-plus semiring* (also known as the schedule algebra) $\mathbb{R}_{\max}$ is the set $\mathbb{R} \cup \{-\infty\}$ with the operations

$$a \oplus b = \max\{a, b\} \quad \text{and} \quad a \odot b = a + b.$$

So for example, $2 \oplus 3 = 3$ and $2 \odot 3 = 5$. Moreover, we have $a \oplus -\infty = a$ and $a \odot 0 = a$ so that $-\infty$ and 0 are respectively the neutral element for the addition and for the multiplication. Hence $\mathbb{R}_{\max}$ is indeed a *semiring*, a ring "without minus", or, more precisely, a triple $(S, \oplus, \odot)$ such that $(S, \oplus)$ is a commutative additive monoid with neutral element $\mathbf{0}$, $(S, \odot)$ is a multiplicative monoid with neutral element $\mathbf{1}$, we have distributivity from both sides, and $\mathbf{0} \odot a = a \odot \mathbf{0} = \mathbf{0}$.

Other examples of semirings are the natural numbers $\mathbb{N}$, the dual $\mathbb{R}_{\min}$ of $\mathbb{R}_{\max}$ (the set $\mathbb{R} \cup \{\infty\}$ and min instead of max), the ideals of a commutative ring with sum and intersection of ideals as operations or the square matrices over a semiring; see [Gol99] for the theory of semirings in general and applications.

The semirings $\mathbb{R}_{\max}$ and $\mathbb{R}_{\min}$ are *semifields* with $a^{(-1)} = -a$. Moreover, they are *idempotent* semifields, that is, $a \oplus a = a$. Note that nontrivial rings cannot be idempotent since then we would have $1 + 1 = 1$ and so by subtracting one also $1 = 0$. Idempotent semirings are actually "as far away as possible" from being a ring because in such semirings $a \oplus b = \mathbf{0} \Rightarrow a = b = \mathbf{0}$. Hence zero is the only element with an additive inverse.

There is a *standard partial order* on idempotent semirings defined by $a \preceq b$ if $a \oplus b = b$. For $\mathbb{R}_{\max}$ this is the usual order on $\mathbb{R}$. Due to this order, the theory of idempotent semirings and modules is closely related to lattice theory. Moreover, it is a crucial ingredient for the development of *idempotent analysis* [KM97], which studies functions with values in an idempotent semiring. The idempotent analog of algebraic geometry over $\mathbb{R}_{\min}$ and $\mathbb{R}_{\max}$ respectively is known as *tropical algebraic geometry* [RGST05]. For a recent survey on *idempotent mathematics* and an extensive bibliography we refer to [Lit05].

*Johann Radon Institute for Computational and Applied Mathematics (RICAM), Austrian Academy of Sciences. E-mail: georg.regensburger@oeaw.ac.at.

This work was supported by the Austrian Science Fund (FWF) under the SFB grant F1322.

I would like to thank Martin Burger for his suggestions to study semirings in connection with nonlinear differential equations and Symbolic Computation and for useful discussions. I also extend my thanks to Markus Rosenkranz and our project leaders Bruno Buchberger and Heinz W. Engl for helpful comments.

2 Max-Plus Linear Algebra

The analog of linear algebra for matrices over idempotent semirings and in particular for the max-algebra has been widely studied starting from the classical paper [Kle56]. The first comprehensive monograph on this topic is [CG79]. See for example the survey [GP97] for more references, historical remarks, and some typical applications of max-plus linear algebra ranging from language theory to optimization and control theory.

From now we consider only the max-algebra $\mathbb{R}_{\max}$, although the results remain valid for $\mathbb{R}_{\min}$ after the appropriate changes (for example, replacing $\leq$ with $\geq$ or $-\infty$ with ∞). Moreover, most of the results can be generalized to linearly ordered commutative groups with addition defined by the maximum, see for example [But94].

For matrices with entries in $\mathbb{R}_{\max}$ and compatible sizes we define

$$(A \oplus B)_{ij} = A_{ij} \oplus B_{ij} \quad \text{and} \quad (A \odot B)_{ij} = \bigoplus_k A_{ik} \odot B_{kj} = \max_k (A_{ik} + B_{kj}).$$

Like in Linear Algebra matrices represent max-plus linear operators over max-plus semimodules and the matrix operartions correspond to the addition and composition of such operators.

The *identity matrix* is

$$I = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{1} \end{pmatrix} = \begin{pmatrix} 0 & -\infty & \dots & -\infty \\ -\infty & 0 & \dots & -\infty \\ \vdots & \vdots & \ddots & \vdots \\ -\infty & -\infty & \dots & 0 \end{pmatrix}.$$

More generally, we denote *diagonal matrices* with $\mathbf{0} = -\infty$ outside the diagonal by $\operatorname{diag}(a_1, \dots, a_n)$. A *permutation matrix* is a matrix obtained by permuting the rows and/or the columns of the identity matrix, and a *generalized permutation matrix* is the product of a diagonal matrix and a permutation matrix. It can be shown [CG79, GP97] that the only invertible matrices in the max-algebra are generalized permutation matrices. So in particular a matrix $A \in \mathbb{R}^{n\times n}$ is not invertible in $\mathbb{R}_{\max}$.

Many basic problems in max-plus linear algebra such as systems of linear equations, eigenvalue problems, linear independence and dimension are closely related to combinatorial problems and hence also the corresponding solution algorithms, see [But03]. For the application described in the next section we are interested in particular in solving linear systems over $\mathbb{R}_{\max}$, see Section 4.

3 Generalized Solutions for BVPs and Max-Plus Interpolation

We consider *boundary value problems* (BVPs) for implicit first-order nonlinear ordinary differential equations of the form

$$f(x, y'(x)) = 0, \tag{1}$$

which are known as (stationary) Hamilton-Jacobi equations. As a simple example, take

$$(y'(x))^2 = 1 \quad \text{with} \quad y(-1) = y(1) = 0. \tag{2}$$

Such BVPs usually do not have classical C^1 solutions, one has to define a suitable solution concept to ensure existence and uniqueness of solutions; see [MS92, KM97] for *generalized solutions* in the context idempotent analysis and the relation to *viscosity solutions* as in [CIL92] and for ordinary differential equations in [Li01].

We want to compute symbolically generalized solutions for BVPs assuming that we have a symbolic representation of some or all solutions for the differential equation. The approach is based on Maslov's *idempotent superposition principle*, which in our setting amounts to the following observation.

Suppose we are given two classical C^1 solutions $y_1(x), y_2(x)$ of (1). Then the *max-plus linear combination*

$$y(x) = \max(a_1 + y_1(x), a_2 + y_2(x))$$

for two constants $a_1, a_2 \in \mathbb{R}$ is a again a (generalized) solution, possibly nondifferentiable at some points.

So if we want to solve a BVP given by Equation (1) and two boundary conditions $y(x_1) = b_1$ and $y(x_2) = b_2$ with x_1, x_2 and b_1, b_2 in $\mathbb{R}$, we have to solve the system

$$\max(a_1 + y_1(x_1), a_2 + y_2(x_1)) = b_1$$
$$\max(a_1 + y_1(x_2), a_2 + y_2(x_2)) = b_2.$$

of max-plus linear equations.

More generally, we arrive at the following *max-plus interpolation problem*: *Given m points $x_1, \dots, x_m$ with the corresponding values $b_1, \dots, b_m$ in $\mathbb{R}$ and n functions $y_1(x), \dots, y_n(x)$. Find a (or all) max-plus linear combinations $y(x)$ of $y_1(x), \dots, y_n(x)$ such that $y(x_i) = b_i$.*

To solve this interpolation problem, we have to find a (or all) solutions of the max-plus linear system $A \odot x = b$ with the *interpolation matrix* $A_{ij} = (y_j(x_i))$ and $b = (b_1, \dots, b_m)^T$.

4 Max-Plus Linear Systems

In this section, we outline how we can compute the solution set

$$S(A, b) = \{x \in \mathbb{R}^n \mid A \odot x = b\}$$

of a max-plus linear system for given $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. The method is known since the 1970s. Our presentation and notation is based on [But03], see also there for further details and references.

Note first that by multiplying the linear system $A \odot x = b$ with the invertible diagonal matrix $D = \operatorname{diag}(b_1^{-1}, \dots, b_m^{-1}) = \operatorname{diag}(-b_1, \dots, -b_m)$, we obtain an equivalent *normalized system* $D \odot A \odot x = D \odot b = 0$ (but not a homogenous system in the usual sense since $0 = \mathbf{1}$ in $\mathbb{R}_{\max}$).

So we can assume that we have have to solve a normalized system $A \odot x = 0$, which is in conventional notation the nonlinear system

$$\max_j (a_{ij} + x_j) = 0,$$

for $i = 1, \dots, m$. We see immediately that if x is a solution, then

$$x_j \leq \min_i -a_{ij} = -\max_i a_{ij}$$

for $j = 1, \dots, n$. Writing $\bar{x}_j = -\max_i a_{ij}$ for the negative of the jth column maximum, this gives in vector notation $x \leq \bar{x}$.

On the other hand, for x being a solution, we must also have in each row at least one column maximum that is attained by x_j. More precisely, let

$$M_j = \{k \mid a_{kj} = \max_i a_{ij}\}.$$

Then $x \in S(A) = S(A, 0)$ iff

$$x \leq \bar{x} \quad \text{and} \quad \bigcup_{j \in N_x} M_j = \{1, \dots m\},$$

where $N_x = \{j \mid x_j = \bar{x}_j\}$. Hence $A \odot x = 0$ has a solution iff the *principal solution* $\bar{x}$ solves the system iff $\bigcup_j M_j = \{1, \dots m\}$.

Since the principal solution can be computed in $\mathcal{O}(mn)$ operations, we can decide the *solvability* of a max-plus linear system with this complexity. With the above characterization of solutions one also sees that for deciding if the principal solution is the *unique* solution, we have to check that $\bar{x}$ is a solution and $\bigcup_{j \in N} M_j \neq \{1, \dots m\}$ for any proper subset $N \subset \{1, \dots n\}$. This amounts to a *minimal set covering problem*, which is well known to be NP-complete. For the max-plus interpolation problem this means that deciding if there exists a solution and computing it is fast but deciding uniqueness for larger problems is difficult.

Like in Linear Algebra the *number of solutions* $|S(A,b)|$ for a linear system is either 0, 1 or ∞. By contrast, even if a system $A \odot x = b$ has a unique solution for some right hand side b, one can always find a b such that there are respectively no and infinitely many solutions. More precisely,

$$T(A) = \{|S(A,b)| \mid b \in \mathbb{R}^m\} = \{0, 1, \infty\}.$$

Furthermore, the only other possible case is $T(A) = \{0, \infty\}$. For the max-plus interpolation problem this implies in particular that the solvability depends on b and there are always values b such that it is solvable.

Finally, we want to emphasize that unlike in Linear Algebra, a *general max-plus linear system* $A \odot x \oplus b = C \odot x \oplus d$ is not always equivalent to one of the form $A \odot x = b$. For several other important cases, like the spectral problem $A \odot x = \lambda \odot x$, the fixed point problem $x = A \odot x \oplus b$, or two-sided linear systems $A \odot x = B \odot x$, there also exist efficient solution methods, see [But03, GP97].

5 The MaxLinearAlgebra Package

To the best of our knowledge, the only package for max-plus computations in a computer algebra system is the Maple package MAX by Stéphane Gaubert. It implements basic scalar-matrix operations, rational operations in the so called minmax-algebra, and several other more specialized algorithms. The package works in Maple V up to R3 but not in newer versions, for details see `http://amadeus.inria.fr/gaubert/PAPERS/MAX.html`.

For numerical computations in the max-algebra, there is the Maxplus toolbox for Scilab, which is developed by the Maxplus INRIA working group. The current version is available at `http://www.scilab.org`. A toolbox for max-algebra in Excel and some MATLAB functions (e.g. for two-sided max-plus linear systems) by Peter Butkovič and his studends are available at `http://web.mat.bham.ac.uk/P.Butkovic/software`. Some additional software is available at `http://www-rocq.inria.fr/MaxplusOrg`.

Our Maple package `MaxLinearAlgebra` is based on the `LinearAlgebra` package introduced in Maple 6. We also use the `ListTools` and `combinat` package. The names correspond (wherever applicable) to the commands in Maple with a `Max` and `Min` prefix, respectively. We have implemented basic matrix operations and utility functions, solvability tests and solutions for max/min-plus linear systems, and max/min linear combinations and interpolation. The package could serve as framework for implementing other max-plus algorithms in Maple, some also based on the already implemented ones, as for example the computation of bases in $\mathbb{R}_{\max}$, see [CGB04].

For the application to BVPs we rely on Maple's `dsolve` command to compute symbolic solutions of differential equations. Using the identities

$$\max(a,b) = \frac{a+b+|a-b|}{2} \quad \text{and} \quad \min(a,b) = \frac{a+b-|a-b|}{2},$$

we can express max/min linear combinations and hence generalized solutions for BVPs with nested absolute values. This has advantages in particular for symbolic differentiation.

The package and a worksheet with examples for all functions, large linear systems, and BVPs are available at `http://gregensburger.com`. See also the next section for two examples.

6 Examples

We first consider the example (2). The differential equation has the two solutions $y_1(x) = x$ and $y_2(x) = -x$. After loading the package

```
> with(MaxLinearAlgebra):
```

we compute the interpolation matrix

```
> A:=InterpolationMatrix([x->x,x->-x],<-1,1>);
```

$$A := \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}$$

and solve the corresponding max-plus linear system

```
>   linsolmax:=MaxLinearSolve(A);
```

$$linsolmax := \left[\begin{bmatrix} -1 \\ -1 \end{bmatrix}, [[1,2]]\right]$$

The first element is the principal solution and the second element describes the solution space, here we have a unique solution $(x_1, x_2) = (\bar{x}_1, \bar{x}_2)$. The generalized max solution is then

```
>   MaxLinearCombination(linsolmax[1],[x,-x]);
```

$$\max(-1+x, -1-x)$$

or with absolute values

```
>   MaxLinearCombinationAbs(linsolmax[1],[x,-x]);
```

$$-1+|x|$$

As a second example, we consider the BVP

$$y'^3 - xy'^2 - y' + x \quad \text{with} \quad y(-1) = y(0) = y(1) = 0. \tag{3}$$

The differential equation has three solutions $y_1(x) = x$, $y_2(x) = -x$ and $y_3(x) = 1/2x^2$. The corresponding interpolation matrix is

```
>   A:=InterpolationMatrix([x->x,x->-x,x->1/2*x^2],<-1,0,1>);
```

$$A := \begin{bmatrix} -1 & 1 & 1/2 \\ 0 & 0 & 0 \\ 1 & -1 & 1/2 \end{bmatrix}$$

There is no max-plus solution

```
>   IsMaxMinSolvable(A,ColumnMax(A));
```

false

but one min-plus solution that gives the generalized solution

```
>   MinLinearCombinationAbs(linsolmin[1],[x,-x,1/2*x^2]);
```

$$1/2 - 1/2\,|x| + 1/4\,x^2 - 1/2\,\left|-1+|x|+1/2\,x^2\right|$$

for (3), and it looks like:

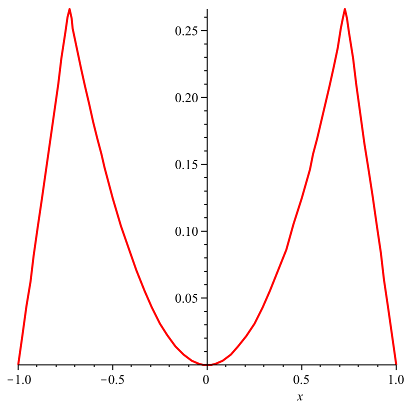


Figure 1: The generalized min-plus solution for (3).

References

[But94] Peter Butkovič, *Strong regularity of matrices—a survey of results*, Discrete Appl. Math. **48** (1994), no. 1, 45–68. MR1254755

[But03] ______, *Max-algebra: the linear algebra of combinatorics?*, Linear Algebra Appl. **367** (2003), 313–335. MR1976928

[CG79] Raymond Cuninghame-Green, *Minimax algebra*, Lecture Notes in Economics and Mathematical Systems, vol. 166, Springer-Verlag, Berlin, 1979. MR580321

[CGB04] Raymond Cuninghame-Green and Peter Butkovič, *Bases in max-algebra*, Linear Algebra Appl. **389** (2004), 107–120. MR2080398

[CIL92] Michael G. Crandall, Hitoshi Ishii, and Pierre-Louis Lions, *User's guide to viscosity solutions of second order partial differential equations*, Bull. Amer. Math. Soc. (N.S.) **27** (1992), no. 1, 1–67. MR1118699

[Gol99] Jonathan S. Golan, *Semirings and their applications*, Kluwer Academic Publishers, Dordrecht, 1999. MR1746739

[GP97] Stéphane Gaubert and Max Plus, *Methods and applications of* $(\max,+)$ *linear algebra*, STACS 97 (Lübeck), Lecture Notes in Comput. Sci., vol. 1200, Springer, Berlin, 1997, pp. 261–282. MR1473780

[Kle56] Stephen C. Kleene, *Representation of events in nerve nets and finite automata*, Automata studies, Annals of mathematics studies, no. 34, Princeton University Press, Princeton, N. J., 1956, pp. 3–41. MR0077478

[KM97] Vassili N. Kolokoltsov and Victor P. Maslov, *Idempotent analysis and its applications*, Mathematics and its Applications, vol. 401, Kluwer Academic Publishers Group, Dordrecht, 1997. MR1447629

[Li01] Desheng Li, *Peano's theorem for implicit differential equations*, J. Math. Anal. Appl. **258** (2001), no. 2, 591–616. MR1835561

[Lit05] Grigori L. Litvinov, *The Maslov dequantization, idempotent and tropical mathematics: a very brief introduction*, Idempotent mathematics and mathematical physics, Contemp. Math., vol. 377, Amer. Math. Soc., Providence, RI, 2005, pp. 1–17. MR2148995

[MS92] Victor P. Maslov and S. N. Samborskiĭ, *Stationary Hamilton-Jacobi and Bellman equations (existence and uniqueness of solutions)*, Idempotent analysis, Adv. Soviet Math., vol. 13, Amer. Math. Soc., Providence, RI, 1992, pp. 119–133. MR1203788

[RGST05] Jürgen Richter-Gebert, Bernd Sturmfels, and Thorsten Theobald, *First steps in tropical geometry*, Idempotent mathematics and mathematical physics, Contemp. Math., vol. 377, Amer. Math. Soc., Providence, RI, 2005, pp. 289–317. MR2149011

Computer Algebra and Experimental Mathematics

Petr Lisoněk*
Department of Mathematics
Simon Fraser University
Burnaby, BC
Canada V5A 1S6

e-mail: plisonek@sfu.ca

4 April 2008

Abstract

We discuss a new paradigm for experimental mathematics research supported by computer algebra. Instead of using the computer algebra system for studying one specific mathematical phenomenon interactively, we aim at acquiring a broader understanding of it by searching (in a "data mining" style) a large data set, such as the On-line encyclopedia of integer sequences. We give a case study documenting the viability and usefulness of this approach.

1 Introduction

Since its very beginnings, computer algebra has impacted research in mathematics profoundly and in various ways. It has freed us from tedious manual computations and it has enabled symbolic computations at an immense scale. Later on the computer algebra systems (CAS) have allowed a new approach to mathematics as an experimental science. We recommend the excellent volumes [1, 2] or the fine selection of examples in [8] for a survey of the state-of-the-art in experimental mathematics. Using Wilf's words [8], in experimental mathematics the computer plays the rôle of astronomer's telescope, providing insight into the problem under consideration. Subsequently this insight is used in formulation of rigorous mathematical proofs.

A distinct feature of CAS is their interactive environment, which in our opinion has had major influence on the way experimental mathematics has been done. In this note we propose to depart from the interactive style of experimental mathematics, but to still take advantage of today's powerful CAS. This will be done by means of one case study in our recent research; some more general conclusions will be proposed at the end of this note.

2 Data Mining the OEIS

Data mining is the process of extracting new, often unexpected information and relationships from large data sets. Often this is performed numerically on approximate (statistical) data sets. A natural counterpart to be considered is data mining by symbolic methods applied to exact data sets. In the case study outlined below we apply data mining to the famous *On-line encyclopedia of integer sequences* [5], which we will henceforth abbreviate by OEIS. This date base is maintained by N.J.A. Sloane who created it partially in collaboration with Simon Plouffe [6]. At this time OEIS contains about 137,000 sequences and new sequences are being added daily.

*Research partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

The design of the massive OEIS web site does not make it very obvious that the entire data base is freely available for download, and that this can be done quite easily. In fact one of the leading experimental mathematicians used to give the OEIS as an example of "data that you would not wish/need to store locally on your computer," due to the superb search capabilities of its Web interface.

The download page for the full OEIS database is at

http://www.research.att.com/~njas/sequences/Seis.html#FULL.

As of today the complete data base can be downloaded as a set of 137 files each of which (except the last one) contains 1,000 sequences. (One can easily create a simple shell script that will download all files, for example by repeatedly calling the widely available `wget` program.) The downloaded files are formatted in the internal format used in the OEIS itself, as described at

http://www.research.att.com/~njas/sequences/eishelp1.html.

This format is highly structured and it can be parsed easily by a short program written in any higher level programming language. The database contains one entry per sequence. The most important components of each entry are: the identification line, the first few terms of the sequence, a brief description or definition of the sequence, a list of references in the literature and/or on the WWW, and a formula/recurrence/generating function/computer program for the sequence (if known).

One can also download a single compressed file containing just the sequences and their serial numbers. However, this abridged information would be unsuitable for our purposes, as we will explain shortly.

2.1 Case study: mathematical background

In a recent paper [3] we gave a structure theorem (along with many concrete examples) of combinatorial enumerating sequences belonging to the so-called quasi-polynomial class. We now very briefly summarize the mathematical background for quasi-polynomials. After that in the rest of Section 2 we outline the process of discoveries that ultimately led to our fairly general results on this topic. This process is not described in [3]. We wish to give an account of it since it might motivate similar research in other areas of combinatorics or algebra.

We say that the sequence (a_n) is *quasi-polynomial* if its ordinary generating function can be written as

$$\sum_{n\geq 0} a_n z^n = \frac{P(z)}{Q(z)}$$

with $P, Q \in \mathbb{Q}[z]$ and Q being a product of (not necessarily distinct) cyclotomic polynomials. (That is, all roots of Q are complex roots of unity.) The term "quasi-polynomial" is well established in the literature, see for example Sections 4.4 and 4.6 of [7] (and the historical remarks at the end of Chapter 4 therein).

One rich source of quasi-polynomial sequences is given by counting integer points in rational polytopes. A *rational convex polyhedron* is the set of those points $u \in \mathbb{R}^d$ that satisfy $Au \geq b$ for some $A \in \mathbb{Z}^{k\times d}$ and $b \in \mathbb{Z}^k$. If a rational convex polyhedron is bounded, then we call it a *rational convex polytope*. For a rational convex polytope P by $i(P)$ we denote the number of integer points in P, i.e., $i(P) := |P \cap \mathbb{Z}^d|$. If P is the rational convex polytope determined by $Au \geq b$, then for $n \in \mathbb{N}$ the *n-th dilate of P*, denoted by nP, is defined as the polytope determined by $Au \geq nb$. In 1962 Ehrhart proved:

Theorem 1. *For each rational convex polytope P the sequence $(i(nP))$ is quasi-polynomial in n.*

2.2 Search for quasi-polynomial sequences

Prior to this work we already had some isolated results on quasi-polynomials, which were proved in an ad hoc manner. We desired a broader coverage of examples, leading to a better understanding of the topic. From each OEIS entry we produced a truncated generating function, which was subjected to Maple's `numapprox[pade]` routine, with all mathematically admissible combinations of numerator and denominator degrees. It was not necessary to make this process more sophisticated as it took only about 40 hours of CPU time to scan the entire OEIS in this way.

Of course, the idea of guessing a generating function for a sequence from its initial segment has been around for a long time. Maple's excellent `gfun` package [4] provides (among many other things) the guessing functionality for a much broader class of generating functions than those considered here.

Whenever a putative quasi-polynomial generating function was discovered, we stored it in a file along with all other information given for it in the OEIS records as described above. This additional information was of immense importance: It enabled us to separate potential discoveries (when the OEIS contained no known generating function for the sequence and/or no hints for its possible quasi-polynomial nature) from rediscovering examples belonging to some well known quasi-polynomial families.

2.3 Results

A distinct set of newly conjectured examples came from counting certain classes of combinatorial objects such as non-linear codes or block designs in the presence of some natural isomorphism relation on these structures. These objects depend on two or more parameters (in the case of codes, the parameters are the block length and the number of codewords) and as such their counts occur in the OEIS in several instances. This caused multiple detections for each family, thus reinforcing the conjectures. Understanding the common features that link these automatically discovered conjectures allowed us to give a proof for each individual type of structure. [3] More importantly we were also able to generalize Ehrhart's theorem quoted above to the case when instead of counting individual lattice points we count their orbits, assuming a suitable group action on the integer lattice $\mathbb{Z}^d$; this is Theorem 2.5 in [3].

3 Conclusion

In has been perhaps overlooked in the context of experimental mathematics that the current computer algebra systems are sufficiently powerful to be applied in a batch mode to large data sets instead of studying individual phenomena interactively, and this is the observation that we aim to convey in this note. Naturally the benefit from acquiring a broader perspective is that in the end the proofs can be formulated more generally and they may become useful to a broader audience. We found some evidence of this when the paper [3] became the "hottest article" of the Journal of Combinatorial Theory Ser. A for the time period April–June 2007 [9].

References

[1] J.M. Borwein, D.H. Bailey, *Mathematics by experiment: plausible reasoning in the 21st century.* AK Peters, 2004.

[2] J.M. Borwein, D.H. Bailey, R. Girgensohn, *Experimentation in mathematics: computational paths to discovery.* AK Peters, 2004.

[3] P. Lisoněk, Combinatorial families enumerated by quasi-polynomials. *J. Combin. Theory Ser. A* **114** (2007), 619–630.

[4] B. Salvy, P. Zimmermann, Gfun: a Maple package for the manipulation of generating and holonomic functions in one variable. *ACM Trans. Math. Softw.* **20** (1994), 163-177.

[5] N.J.A. Sloane, *The on-line encyclopedia of integer sequences.* http://www.research.att.com/~njas/sequences/ (Retrieved on 28 February 2008.)

[6] N.J.A. Sloane, S. Plouffe, *The encyclopedia of integer sequences.* Academic Press, 1995.

[7] R.P. Stanley, *Enumerative Combinatorics. Volume I.* Wadsworth & Brooks, 1986.

[8] H. Wilf, *Mathematics: An experimental science.* Draft of a chapter in the forthcoming volume "The Princeton Companion to Mathematics," edited by T. Gowers. Available from http://www.math.upenn.edu/~wilf/reprints.html. (Retrieved on 28 February 2008.)

[9] http://top25.sciencedirect.com/index.php?journal_id=00973165&cat_id=12 (Retrieved on 28 February 2008.)

Automatic Regression Test Generation for the SACLIB Computer Algebra Library (Extended Abstract)

David Richardson and Werner Krandick

Department of Computer Science
Drexel University
Philadelphia, Pennsylvania, USA

1 Introduction

Regression testing is a software engineering technique that retests previously tested segments of a software system to ensure that they still function properly after a change has been made [1, 10]. Functional regression testing involves executing unit tests and verifying that the output agrees with the output of an earlier version of the software. Regression tests are usually automated and performed in regular intervals during software development. During software maintenance automated regression tests are performed after each modification of the software.

Developers of computer algebra software use a variety of execution-based testing methods. In many cases published test suites such as mathematical tables are used. Some test suites involve mathematically conceived test cases designed to exercise certain features of an algorithm. Other testing techniques involve round-trip computations, comparisons with results computed by other computer algebra systems, or comparisons with results computed by reference implementations within the same computer algebra system. Those testing techniques typically test high-level functionalities and thus tend to be of limited value for the localization of program defects.

We present a technique for the automated generation of unit tests for the SACLIB library of computer algebra programs [4, 6]. While running a high-level computation we automatically collect the input–output pairs of each function that is called. We then automatically generate, for each function, a test environment that takes the collected inputs, runs the function, and checks whether the obtained outputs agree with the collected outputs.

Our technique does not verify whether system functions conform to specifications, nor does it provide more code coverage than the high-level computation we run. However, the unit tests we generate help localize errors and provide a framework that can be easily augmented with additional test cases.

We use aspects to weave tracing code into SACLIB functions. Aspect-Oriented Programming (AOP) [8, 7] is a programming methodology designed to facilitate the encapsulation of program requirements that cannot be implemented in a single component using traditional software development methods. We use AspectC++ [15], an extension to C++ that provides direct language support for aspects.

Applications of AspectC++ have primarily been focused on using AOP to provide refactorings and implementations with improved modularity and configurability without compromising runtime efficiency in memory footprint or execution speed. There has also been some use of aspects for generating trace information useful in debugging and profiling [9]. The testing research on aspects has been focused on adapting existing testing algorithms to handle aspects [11], improving test selection in the presence of aspects [17, 16, 18, 19], and providing unit test facilities for aspects [12]. We are not aware of any literature on the use of aspects for automated test bed generation.

The SACLIB computer algebra library serves as the basis of the quantifier elimination systems QEPCAD [5] and QEPCAD B [2, 3]. In earlier work we ported SACLIB from C to C++ so as to be able to use iterator concepts to refactor the SACLIB memory management subsystem. In the resulting library, SACLIB 3.0, the absence of memory leaks and double deletes is proved during compilation [14, 13]. The present work allows us to perform a systematic regression test of SACLIB 3.0 with respect to the original SACLIB—the last step before a release of SACLIB 3.0.

There are 1070 SACLIB routines; of these, 894 take only SACLIB objects as arguments. Our current aspects serialize SACLIB objects, and hence we can generate test beds for those 894 routines. We are close to completing aspects that will serialize the remaining kinds of arguments and thus allow us to generate test beds for the remaining SACLIB functions.

```
 1: /*=============================================
 2:                 L <- LIST2(a,b)
 :
 :  <Specifications omitted>
 :
13: #include "saclib.h"
14: #include "trace_utils.h"
15:
16: Word LIST2(Word a, Word b)
17: {
18:    trace::trace_signature("Word LIST2(Word,Word)");
19:    trace::trace_input("argument 0", a);
20:    trace::trace_input("argument 1", b);
21:
22:  Word L,M;
 :
 :  <Function body omitted>
 :
44: Return: /* Prepare for return. */
45:        trace::trace_output("argument 0", a);
46:        trace::trace_output("argument 1", b);
47:        trace::trace_return(L);
48:        return(L);
49: }
```

(a)

```
 1: Word LIST2(Word,Word)
 2:    return: (12,15)
 3:    argument 0 input:  12
 4:    argument 0 output: 12
 5:    argument 1 input:  15
 6:    argument 1 output: 15
 7: %%
 8: Word LIST2(Word,Word)
 9:    return: ((12,15),11)
10:    argument 0 input:  (12,15)
11:    argument 0 output: (12,15)
12:    argument 1 input:  11
13:    argument 1 output: 11
14: %%
```

(b)

Figure 1: (a) Manually inserted tracing code. The inputs (lines 18-20) and the outputs (lines 45-47) are traced using functions obtained from the header file trace_utils.h (line 14). (b) Test cases produced by invoking the function in (a) as LIST2(LIST2(12,15),11). Each invocation of LIST2 produces a record that starts with the signature of the traced function (Lines 1,8) and ends with %% (Lines 7,14). Lines 2-6, 9-13 trace inputs and outputs.

2 Function level tracing

Automatically recording function level test cases during the execution of a program requires that function inputs and outputs are collected as each function is executed. Figure 1(a) shows the manually instrumented SACLIB function LIST2 which composes two SACLIB objects into a list. Figure 1(b) shows the test cases that are produced by invoking LIST2(LIST2(12,15),11).

In order for the trace functions to record enough information to use as a test case, they must serialize enough of execution context of the instrumented function to allow it to be invoked solely from the information recorded in the trace. This is only possible if the trace_* functions have knowledge of the execution context of the instrumented function.

The key to building a test harness for SACLIB is to determine both what is necessary for the serialization of a test case and how the needed state can be efficiently computed on a time scale that makes testing worthwhile. The design and implementation of the tracing functions is driven by striving for the proper balance between these two competing objectives. For SACLIB, correct tracing requires the ability to identify the function currently executing, handle recursive calls of the traced function, trace input/outputs of arbitrary type, allow SACLIB functions to be used inside of the trace_* functions without being traced, trace relevant global state used by the traced function, and trace functions with multiple exit points. Performing this tracing automatically requires the ability to insert tracing code into all SACLIB functions without the need for hand instrumentation. Care must also be taken in the storage and reuse of test cases. A test case is only worth storing for use in later testing if it provides fault detection power beyond the test cases that have been previously stored. After the test cases are stored, a test harness must be provided to execute the test cases.

3 Function call identification

The trace_signature function (Figure 2) is responsible for identifying the function being traced and allowing the tracing of recursive function invocations. Each stack frame is described by a stack_frame_record (lines 4-14). When trace_signature is invoked, it adds a stack_frame_record to the map stack_frame and stores the signature for the function being traced (lines 22-23). The signature of the traced function is supplied by the caller of trace_signature (line 20). This argument must match the function being traced. Because the other tracing functions require knowledge of the stack frame they are tracing data for, test_signature must be called inside the traced function

```

namespace trace{

   struct stack_frame_record{
      stack_frame_record():has_return(false){}

      std::string              signature;
      bool                     has_return;
      std::string              return_value;
      std::vector<std::string> input;
      std::vector<std::string> output;

      void clear();//reset all fields
   };
   std::ostream& operator<<(std::ostream& out, const stack_frame_record& r);

   extern int                               stack_frame_id;
   extern std::map<int, stack_frame_record> stack_frame;

   void trace_signature(const std::string& signature){

      ++stack_frame_id;
      stack_frame[stack_frame_id].signature = signature;

   }//trace_signature

   template <typename T>
   void trace_return(T t){

         stack_frame[stack_frame_id].has_return=true;
         stack_frame[stack_frame_id].return_value = to_string(t);
         trace_stream << stack_frame[stack_frame_id];
         stack_frame[stack_frame_id].clear();
         --stack_frame_id;

   }//trace_return

}//namespace
```

Figure 2: The implementation of trace_signature and trace_return must record the call stack in order to allow tracing of recursive functions.

```
#include <string>

namespace trace{

   extern std::ostream& trace_stream;

   template <typename T>
   void trace_input(std::string& argument arg, const T& value){
      trace_stream << arg << " input: " << value << "\n";
   }//trace_input

   void trace_input(std::string& argument arg, const Word& value){
      trace_stream << arg << " input: ";
      OWRITE(trace_stream, value);
   }//trace_input

}//namespace
```

Figure 3: An overload of the trace_input function must be provided for each type to be traced. The first overload is for streamable types. The second overload is for SACLIB objects.

before any other tracing functions are called.

Immediately before the traced function returns, trace_return (lines 27-36) must be called with the return value of the traced function. The return value is stored in stack_frame_record (line 30-31) and the trace information for the stack frame is serialized to trace_stream (line 32). The stack_frame_record is then made available for reuse (lines 33-34). Because trace_return removes the stack last record, it can only be called after all trace_input and trace_output calls have completed.

All tracing information for a stack frame is aggregated into a stack_frame_record and is not serialized until all trace information for a single frame is available. This is needed to trace recursive function applications and functions that call other traced functions. If this aggregation were not performed, the serialized tracing output from different functions would be interspersed in trace_stream. Although the implementation of the trace_* functions all properly aggregate information to trace::stack_record, all subsequent code examples in this paper will be shown with direct serialization to trace_stream in order to simplify the presentation.

4 Recording input/output

Tracing of the inputs is accomplished by calls to the trace_input function defined in Figure 3. These calls must occur once for each argument to the trace function. The trace_input calls must occur after the call to test_signature, once for each argument of the traced function, before any calls to trace_output or trace_return, and with an indication of which input is being traced. Tracing of the outputs is handled similarly by trace_output. All trace_output calls must occur after all trace_input calls and before the trace_return call. Note the string literals used as the arguments to the trace_* functions correspond to the strings in the output file.

Generally, input/output tracing of a C++ variable t of type T requires the ability to serialize objects of type T. Unfortunately, this requires each type to be handled differently. Tracing SACLIB requires the ability to serialize C++ fundamental types, C++ pointers to fundamental types, std::strings, SACLIB atoms, SACLIB lists, recursively fixed iterators, structure protecting iterators, and simple pointers [14]. We do not describe the serialization techniques here.

5 Recording global state

The output of SACLIB routines depends on a relatively limited set of external state. Most routines are only influenced by the state of the heap and the space array. The space array is where all garbage collected SACLIB lists and integers are stored. The trace_input, trace_output, and trace_return functions described in Section 4 automatically take care of the heap and space array. This is because they serialize the value of the variables stored on the heap or in the space array. The stored values may then be deserialized into storage provided by the test harness. From the perspective of SACLIB routines, the serialized and deserialized values are identical because SACLIB functions on values.

A few SACLIB routines are influenced by the state of the SACLIB random number generator (the global variables RINC, RTERM, and RMULT) and the floating point error status (the global variable FPHAND). These

```
#ifndef TRACE_AH
#define TRACE_AH

#include "trace_utils.h"

aspect Trace{

   pointcut exclude_set() = execution(
      "% trace::%(...)"
   );

   pointcut trace_set() = !exclude_set() && execution("% %(...)");

   advice trace_set(): around() {
      trace::trace_signature(tjp->signature());
      trace::trace_input(tjp);

      tjp->proceed();

      trace::trace_output(tjp);
      trace::trace_return(tjp->result());
   }

};

#endif
```

Figure 4: An aspect to weave tracing code around execution joinpoints.

global variables are all SACLIB objects and can be handled by adding code to the trace aspect that uses trace_input and trace_output to serialize these four variables from all SACLIB routines.

The remaining global state does not need to be serialized for testing. There are several read-only lookup tables that are populated by SACLIB when a SACLIB program is started. They contain precomputed reference data such as a list of the first primes. These tables will be populated with identical values each time a SACLIB program is initialized with BEGINSACLIB. All other SACLIB routines cannot be used before BEGINSACLIB has been called. Because of this, these SACLIB global variables are effectively serialized in the code used to implement BEGINSACLIB. This allows SACLIB functions called from a test harness to safely access this global state without any danger of missing global state required by the function.

There are four SACLIB routines that do not have all of the global state serialized by the trace_* functions. The SACLIB routines CREAD, CWRITE, SWRITE, and BKSP all perform i/o and produce side effects on i/o streams. Regression testing of these routines would require the serialization of the full state of the streams used for i/o. This would also require the ability to serialize any data that had been written to storage by the streams. The SACLIB library is primarily designed for computation and does not preform very much i/o. Additionally, many uses of i/o are only for debugging. Performing serialization of the stream state is not worthwhile. Not only is i/o not a significant part of SACLIB, but constructing unit tests for the i/o routines requires less effort than devising a method for serializing the state of the i/o streams. The i/o routines in SACLIB are not traced.

6 Aspect based tracing

SACLIB contains over 1,000 functions. Adding hand tracing to all of these functions is clearly a tedious and undesirable task. The requirement to call the tracing functions in the correct order with the correct arguments is an error prone process. The requirement to call the trace_* functions to match the structure of the trace function results in duplicating information about the trace functions arguments in the code. This poses a maintenance hazard: any updates to the traced function require trace_* calls to be updated. Given that adding the trace_* calls is tedious, error-prone, and the source of the traced function provides the information of the only correct way to call the trace_* functions, automated insertion of the trace_* functions is a natural solution.

We remove the limitations of hand instrumentation by using AspectC++ [15]. AspectC++ provides a convenient mechanism to automatically add tracing code to all SACLIB functions. AspectC++ extends C++ with three significant extensions: aspects, pointcuts, and joinpoints. The purpose of these extensions is to allow code that implements *cross-cutting* to be stored in an aspect as *advice* and then *woven* into existing source code using an *aspect weaver*. Joinpoints are points in the code where the aspect weaver may place the code contained in an aspect's advice. A pointcut is a set of joinpoints.

While aspects provide a convenient implementation mechanism and vocabulary for the discussion of our tracing method, similar results could be obtained with any source-to-source translation technology.

Figure 4 contains an aspect to weave tracing into all SACLIB functions. Lines 6-24 define the aspect. Lines 8-10 define a pointcut for all of the functions used to implement the tracing. The pointcut exclude_set() defines a pointcut name exclude_set, and the execution("% trace::%(...)") provides the pointcut that contains the execution joinpoint for each function matching the expression "% trace::%(...)". The first % is a wild card that matches any return type, the second % is a wild card that matches any function name, and the ... is a wild card that matches any arguments list. The execution joinpoint for a function is the function invocation. The net result is that "% trace::%(...)" matches all functions in the namespace trace and exclude_set contains all execution joinpoints for these functions.

On line 12 the pointcut trace_set is created. It is created from all joinpoints in execution("% %(...)") that are not in the pointcut exclude_set. Because "% %(...)" matches any function, trace_set will contain all execution joinpoints except those that implement tracing logic. This is exactly the set of joinpoints that tracing should be added to. Lines 14-22 provide the advice needed to trace a function. We use around advice because it allows advice to be woven both before and after each joinpoint. Lines 15-16 weave signature and input tracing to the beginning of the joinpoint, line 18 executes the code contained in the joinpoint, and lines 20-21 weave output and return value tracing after the joinpoint. Notice that the signatures trace_input and trace_output have been modified to take tjp, which stand for "the joinpoint", as an argument. This is possible because in the advice, tjp is aware of its arguments.

In addition to removing all the limitations of hand instrumentation, using AspectC++ also automatically handles traced functions with multiple return statements. On line 21 the return value is obtained by from tjp->result. This provides the return value of the joinpoint after it has executed. Correctly weaving the advice into functions with multiple return statements is handled by the weaver.

7 First order tracing

Because the SACLIB routines OREAD and OWRITE are used to serialize SACLIB atoms and lists, they will be called by routines such as trace::trace_input. If the invocation of OWRITE made from trace::trace_input were also traced, each call to OWRITE from trace::trace_input would result in a call to trace::trace_input, which would result in another call to OWRITE, and ultimately result in a stack overflow. This is dealt with by disabling tracing inside of the trace_* functions. Once tracing has been disabled, SACLIB routines can safely be used for serialization.

8 Test case filtering and execution

During the execution of a SACLIB executable that has been woven with the Trace aspect, it is possible that a single SACLIB routine will be called multiple times (possibly with the same arguments). This is particularly true for routines such as the list processing functions that are used in the implementation of most SACLIB routines. Because each execution of a traced routine will serialize a test case, it is possible that disk space will be used inefficiently. This can occur in two ways. The first is when a test cases does not add to the fault-detecting power of the already collected test cases. In this case, the test could be discarded. The second source of inefficiency arises if the run-time required to execute the collected test cases exceeds the time a user is willing to devote to running the test cases. Currently, we address only the second problem. When the trace aspect is woven into SACLIB, it can be directed to stop test collection for each routine after a certain number of test cases have been collected. In the future, we will address the first problem by using code coverage tools. We currently use the tests with the Retest-All strategy.

9 Test harness generation

Once the test cases have been obtained from the trace aspect, they must be played through a test harness. Figure 5 contains a test harness for LIST2. This test harness was produced automatically from a python code generator that constructs a test harness capable of executing test cases for any SACLIB routine. The output in Figure 5 was restricted to only the code needed to test LIST2 and was slightly modified for readability.

```
using namespace std;

int sacMain(int argc, char **argv){

   ifstream test_cases("test_input");

   while(!test_cases.eof()){
      string signature; read_signature(test_cases,signature);
      if("Word LIST2(Word,Word)" != signature){
         cerr << "signature='" << signature << "'\n"; exit(1);
      }//if

      Word return_value, expected_return;
      read_expected_return(test_cases, expected_return);

      verify_signature("Word LIST2(Word,Word)", signature);

      Word a0; read_input(test_cases, a0, "0");
      Word a1; read_input(test_cases, a1, "1");

      return_value = LIST2(a0, a1);
      check_equal(return_value, expected_return);

      string s;
      getline(test_cases,s);
      if("%%"!=s){cerr << "terminator='" << s << "'\n"; exit(1);}

   }//while

}//main
```

Figure 5: A test harness to execute test cases for the SACLIB routine LIST2.

References

[1] Robert V. Binder. *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Addison-Wesley, 2000.

[2] Christopher W. Brown. QEPCAD B: A program for computing with semi-algebraic sets using CADs. *SIGSAM Bulletin*, 37(4):97–108, 2003.

[3] Christopher W. Brown. QEPCAD B: A system for computing with semi-algebraic sets via cylindrical algebraic decomposition. *SIGSAM Bulletin*, 38(1):23–24, 2004.

[4] George E. Collins et al. SACLIB User's Guide. Technical Report 93-19, Research Institute for Symbolic Computation, RISC-Linz, Johannes Kepler University, A-4040 Linz, Austria, 1993.

[5] George E. Collins and Hoon Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12(3):299–328, 1991. Reprinted in: B. F. Caviness, J. R. Johnson, editors, Quantifier Elimination and Cylindrical Algebraic Decomposition, Springer-Verlag, 1998, pages 174–200.

[6] Hoon Hong, Andreas Neubacher, and Wolfgang Schreiner. The design of the SACLIB/PACLIB kernels. *Journal of Symbolic Computation*, 19(1–3):111–132, 1995.

[7] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of AspectJ. In J. L. Knudsen, editor, *Proceedings of the 15th European Conference on Object-Oriented Programming*, volume 2072 of *Lecture Notes in Computer Science*, pages 327–353. Springer-Verlag, 2001.

[8] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In M. Akşit and S. Matsuoka, editors, *Proceedings of the 11th European Conference on Object-Oriented Programming*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer-Verlag, 1997.

[9] Daniel Mahrenholz, Olaf Spinczyk, and Wolfgang Schröder-Preikschat. Program instrumentation for debugging and monitoring with AspectC++. In L. Bacellar, P. Puschner, and S. Hong, editors, *Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 249–256. IEEE Computer Society Press, 2002.

[10] William E. Perry. *Effective Methods for Software Testing.* John Wiley & Sons, third edition, 2006.

[11] Reginaldo Ré, Otávio Augusto Lazzarini Lemos, and Paulo Cesar Masiero. Minimizing stub creation during integration test of aspect-oriented programs. In D. Xu and R. T. Alexander, editors, *Proceedings of the Third Workshop on Testing Aspect-Oriented Programs*, pages 1–6. ACM Press, 2007.

[12] André Restivo and Ademar Aguiar. Towards detecting and solving aspect conflicts and interferences using unit tests. In *Proceedings of the Fifth Workshop on Software Engineering Properties of Languages and Aspect Technologies.* Article no. 7, 5 pp. ACM Press, 2007.

[13] David G. Richardson. Compiler-Enforced Memory Semantics in the SACLIB Computer Algebra Library. Master's thesis, Drexel University, 2005. Published as Department of Computer Science Technical Report DU-CS-05-14.

[14] David G. Richardson and Werner Krandick. Compiler-enforced memory semantics in the SACLIB computer algebra library. In V. G. Ganzha, E. W. Mayr, and E. V. Vorozhtsov, editors, *International Workshop on Computer Algebra in Scientific Computing*, volume 3718 of *Lecture Notes in Computer Science*, pages 330–343. Springer-Verlag, 2005.

[15] Olaf Spinczyk, Andreas Gal, and Wolfgang Schröder-Preikschat. AspectC++: An aspect-oriented extension to the C++ programming language. In *Proceedings of the 40th International Conference on Technology of Object-Oriented Languages and Systems*, pages 53–60. Australian Computer Society, Inc., 2002.

[16] Dianxiang Xu and Weifeng Xu. State-based incremental testing of aspect-oriented programs. In H. Masuhara and A. Rashid, editors, *Proceedings of the 5th International Conference on Aspect-Oriented Software Development*, pages 180–189. ACM Press, 2006.

[17] Guoqing Xu. A regression tests selection technique for aspect-oriented programs. In *Proceedings of the Second Workshop on Testing Aspect-Oriented Programs*, pages 15–20. ACM Press, 2006.

[18] Guoqing Xu and Atanas Rountev. Regression test selection for AspectJ software. In *Proceedings of the 29th International Conference on Software Engineering*, pages 65–74. IEEE Computer Society Press, 2007.

[19] Jianjun Zhao, Tao Xie, and Nan Li. Towards regression test selection for AspectJ programs. In *Proceedings of the Second Workshop on Testing Aspect-Oriented Programs*, pages 21–26. ACM Press, 2006.

Geometric properties of locally minimal energy configurations of points on spheres and special orthogonal groups

Elin Smith and Chris Peterson

Abstract

In this paper, we construct locally minimal energy configurations of t points on the unit sphere $S^{n-1} \subseteq \mathbb{R}^n$. We utilize basic linear algebra and the computer program *Groups, Algorithms, Programming* (GAP) to generate the subgroups of $SO(n), O(n)$ which permute these points. We also consider the colored complete graph K_t induced by the configuration and the subgroup of the symmetric group, S_t, which preserves the edge colored K_t. Next we consider locally minimal energy configurations of points on $SO(n)$ (as a manifold). After a shift of the configuration, we consider the group generated by the corresponding elements of $SO(n)$ (as a group). In some cases we are able to utilize the LLL algorithm (via Maple) to recover exact representations from the numerical data produced by the algorithms. Finally, we consider basic examples of locally minimal energy configurations of points on other manifolds.

Introduction

The study of configurations of points on spheres may at first sound like a mundane topic but a closer examination reveals a richness of accessible, hard and interesting problems. Configurations which are extremal with respect to some measurement (such as potential energy) often have an associated geometric and/or combinatorial structure of interest. The consideration of point configurations which minimize an energy potential (at least locally) extends beyond the case of spheres [5, 11, 15, 22]. Such extremal sets of points have found applications and connections with several areas of mathematics, including coding/communication theory [17, 24], number theory [12, 16] and group theory [6, 10]. There is a close connection between the packing of points on a sphere (minimal energy configurations) [20] and the packing of spheres in a manifold [7, 8, 14, 21]. There are further connections to be found when one considers packings of points on more general manifolds. For instance, packings of points on $SO(n)$ correspond to packings of special orthogonal frames while packings in the grassmann varieties correspond to packings of subspaces. It is also natural to consider packings in Flag varieties, Stiefel manifolds, general homogeneous spaces, nested spheres, etc. In this paper we utilize a mixture of numeric, symbolic-numeric, and symbolic methods in order to produce and study locally minimal energy configurations of points with special emphasis on S^n and $SO(n)$. In particular we study their geometry, their group of automorphisms, exact representations, their minimal distance graphs and the colorings of complete graphs induced by distances between points.

We use the paper by Cohn and Kumar as a starting point in our study of locally optimal distributions of points on spheres [9]. In order to study configurations of points on spheres, we think of each point as a charged particle and impose a potential energy function between points. Starting from a random initial configuration, we allow the points to spread out in such a way that potential energy is minimized locally for each point with respect to its nearest neighbors. We then analyze the geometry of the point distribution to which varying initial configurations of points stabilize. In particular, we determine (and construct) the subgroups of $SO(n)$ and $O(n)$ which fix the point configurations as sets. In order to construct these groups, we find explicit group elements then utilize GAP [13] to determine and analyze the subgroups of $SO(n)$, respectively $O(n)$, generated by these elements. We check our algorithms on S^1 and find (fortunately) that minimal energy configurations of n points are the vertices of a regular n-gon, that the cyclic group of order n is the subgroup of $SO(2)$ which fixes the n points as a set and the dihedral group of order $2n$ is the subgroup of $O(2)$ which fixes the set. However, on S^n with $n > 1$, we find unexpected behavior at almost every turn. When we extend our analyses to point configurations on $SO(n)$ (as a manifold), the surprises continue.

1 Background

The algorithm we utilize finds point configurations which locally minimize potential energy. However, the most interesting point configurations are the ones that globally minimize potential energy. A sufficient (though not necessary) condition for a local energy minimizer to be a global minimizer for potential energy was obtained by Cohn and Kumar [9]. We follow their framework in the paragraphs below.

Definition 1.1. (Basic definitions for functions)

(1) Given a decreasing, continuous function, f, on $[0, \infty)$ and a finite set C of points on the unit sphere, S^{n-1}, the **potential energy of** C is

$$\sum_{x,y \in C,\ x \neq y} f(|x-y|^2).$$

(2) A C^∞ function f mapping an interval I to the real numbers is **completely monotonic** if $(-1)^k f^{(k)}(x) \geq 0$ for all $x \in I$ and for all $k \geq 0$.

Definition 1.2. (Basic definitions for configurations)

(1) An (n, N, t) **spherical code** is a subset $N \subseteq S^{n-1}$ such that no two distinct points in N have inner product greater than t.

(2) A **spherical** M**-design** is a finite subset of S^{n-1} such that any polynomial f on $\mathbb{R}^n$ with $deg(f) \leq M$ has the same average on the design as on S^{n-1}.

(3) A **sharp configuration** is a finite subset of S^{n-1} such that between distinct points there exists a total of m distinct inner products, and such that the subset is a spherical $(2m-1)$-design.

In [9], Cohn and Kumar prove the following remarkable (and useful) result about sharp configurations.

Theorem 1.3. *For any completely monotonic potential function, sharp arrangements are global potential energy minimizers.*

Note that knowing the dot product between two points on a sphere centered at the origin is equivalent to knowing the distance between the points. If the coordinates of the points on a sphere are stored as the columns of a matrix, A, then the matrix $A^T A$ has as entries all possible dot products between points which can be easily transformed into a matrix containing all distances between points. As a consequence, it is easy to determine the potential energy of the set, to see if a point configuration is a (n, N, t) spherical code and to determine the number of distinct inner products between points. Checking whether a configuration is a spherical M-design is also not difficult; one merely has to check whether every monomial of degree at most M has the same average on the configuration as on S^{n-1}. These averages are known [2]. Linearity then extends the result to all polynomial functions of degree at most M. In summary, one can effectively determine whether a given configuration is sharp.

2 Results

Let P be a point configuration on a D dimensional manifold that (at least locally) maximizes the minimal distance between points. If such a point configuration is on a manifold such as S^n, $SO(n)$ (and a large class of other manifolds) and if the point configuration contains sufficiently many points, then by the local existence and uniqueness theorem for geodesics the minimal distance graph of the point configuration has degree at least $D + 1$ at every vertex.

2.1 Algorithm

We utilize the following algorithm (which is exceedingly easy to implement) in order to find configurations of points on spheres which maximize the minimum distance between points in the configuration:

Algorithm:

```
Pick $G, R, N, K, n, \epsilon$.
Let $S$ be $N$ random points on the unit sphere $S^{n-1} \subset \mathbb{R}^n$.
For ECount:=1 to R
For Count:=1 to K
Randomly pick one of the $N$ points, call it $x$.
Find the nearest neighbor, $y$ to $x$ ($y$ has largest dot product with $x$ among points in $S$).
Determine the line, $L$, between $x$ and $y$.
Move $x$ away from $y$ by a distance $\epsilon$ along $L$, call this new point $x'$.
Normalize $x'$ so that it lies on the unit sphere.
Set $x = x'$.
Next Count.
Let $\epsilon = \frac{\epsilon}{G}$.
Next ECount.
END
```

2.2 Groups acting on point configurations

We first utilized a Matlab implementation of the algorithm for N points on S^2 but later found it advantageous to use a Maple implementation. For each $4 \leq N \leq 24$ we repeated the experiment 100 times. For each of these configurations, we utilized basic linear algebra and GAP [13] to determine the order of the subgroups of $SO(3)$ and $O(3)$ which act on the configurations. We then extended the algorithm to S^n for $n \leq 6$. We check if the configurations are sharp by determining the number of different dot products which occur between distinct points and by averaging each monomial function over the points in the configuration. Let α be one of the dot products obtained through these numerical experiments. When α is algebraic, exact values for α can be determined via an application of the LLL algorithm (utilizing the *LinearDependency* command in Maple) to the vector $V = [1, \alpha, \alpha^2, \dots]$. Several highlights of our results are the following:

- With $N = 4$ points in S^2 we obtain the vertices of a regular simplex 100% of the time. The subgroup of $SO(3)$ which acts on these points is A_4 while the subgroup of $O(3)$ is S_4. Similarly, for $n+1$ points in S^{n-1} the points always converge to lie at the vertices of a regular simplex. These are all sharp configurations.
- With $N = 2n$ points in S^{n-1} we obtain higher dimensional analogues of the square and octahedron (the cross polytopes). Dot products between distinct points all lie in the set $\{0, -1\}$. For 6 points in S^2, a subgroup of $SO(3)$ of order 24 acts on the configuration. These configurations are sharp.
- With 12 points in S^2 we obtained the vertices of an icosahedron 98% of the time. In these cases the subgroup of $SO(3)$ had order 60. The dot products between distinct points all lie in the set $\{-1, \pm\frac{1}{\sqrt{5}}\}$. The vertices of the icosahedron form a sharp configuration. One time out of our 100 experiments we obtained a trivial subgroup. Another time out of our 100 experiments, the subgroup had order 6.
- With 16 points in S^4 and with 27 points in S^5 we recover known sharp configurations. With 16 points, the dot products lie in the set $\{-\frac{3}{5}, \frac{1}{5}\}$. With 27 points, the dot products lie in the set $\{-\frac{1}{2}, \frac{1}{4}\}$.
- With 8 points in S^2 we do not get the vertices of a cube. Instead we always converged to the vertices of a square antiprism (i.e. a cube with the top face rotated by 45 degrees). Adjacent vertices of a cube are 1.1547 units apart while nearest neighbors in a square antiprism are 1.2156 units apart. Thus,

while at first surprising, the vertices of the square antiprism has lower energy than the vertices of the cube.

- With 20 points we do not obtain a dodecahedron. 9 times out of 100 we obtained a configuration whose subgroup of $SO(3)$ had order 6. 8 times out of 100 the subgroup had order 2. 83 times out of 100 the subgroup was trivial.
- With $N = 16$ points in S^2 we obtained 6 different locally minimal energy configurations. With $N = 24$ points in S^2 we obtained 5 different locally minimal energy configurations. We do not know the number of possible locally minimal configurations.

2.3 Point configurations in SO(3)

We modify the algorithm to determine minimal energy configurations in $SO(3)$. A point P_i in $SO(3)$ is a 3×3 orthogonal matrix with determinant 1. In our algorithm, a nearest neighbor to P_i is defined as the point which attains the maximum value in the set

$$\{P_{i,k} \cdot P_{j,k} \mid i \neq j \text{ and } 1 \leq k \leq 3\}$$

where $P_{i,k}$ denotes the k^{th} column of matrix P_i. If $P_{j,k}$ achieves this maximum then we apply a transformation that rotates P_i away from P_j in the plane spanned by the k^{th} columns of P_i, P_j. As an example, consider a starting configuration of 4 random orthogonal matrices. Let A, B, C, D denote the points in a locally minimal energy state as determined by the algorithm. If we now take advantage of the group structure, we can shift the four points to $I, BA^{-1}, CA^{-1}, DA^{-1}$. We next find the group G generated by these four elements. In 80% of our experiments, G was found to be isomorphic to the symmetric group S_3. In the other 20% of our experiments, G was found to be isomorphic to the dihedral group of order 4. We determined the orbit of the vector $[1, 0, 0]$ under the representation of S_3 and obtained the sharp configuration corresponding to the vertices of a regular octahedron.

3 Further Questions

Consider the (n, N, t) spherical code associated to a locally minimal energy configuration. It is natural to construct the associated *minimal distance graph* where the vertices of the graph are the points in the configuration and where two vertices are connected if and only if their distance apart is equal to t. It is easy to show that the degree of each vertex of the graph is at least n. Another natural object to associate to the configuration is a coloring of the edges of the complete graph K_N. The vertices of the K_N correspond to the points in the configuration, while the coloring of the edges correspond to the different dot products. It is natural to ask the following questions:

(1) How many different locally minimal energy configurations exist for each pair n, N?

(2) How many different t are possible for a (n, N, t) spherical code associated to a locally minimal energy configuration?

(3) For a fixed n, N what is the minimum value of t?

(4.1) For which values of n, N is the configuration associated to a global energy minimizer unique up to an action of $O(n)$?

(4.2) Modulo the orbit of the point configuration via $O(n)$, what is the dimension of the parameter space of each local/global energy minimizer?

(5) Characterize/classify the possible minimal distance graphs and K_N colorings that exist for local energy minimizers and for global energy minimizers.

(6) Determine all possible automorphism groups for minimal distance graphs and K_N colorings.

(7) Find algorithms that increase the likelihood of finding global energy minimizers.

(8) Classify sharp configurations.

(9) Classify global energy minimizers.

(10) Determine answers to these questions for point configurations on other manifolds.

(11) For a given manifold M, find the largest chromatic number among graphs that can be embedded in M as an $S = \{a_1, \ldots, a_r\}$ distance graph. I.e. such that the vertices of the graph are connected by an edge if and only if their distance apart lies in the set S.

4 Final Comments

In the full paper version of this abstract, we will include our extensions of energy minimization techniques to other manifolds and (very) partial answers to some of the questions posed on the previous page. We recently (late March, 2008) became aware of the preprint [3] which, for energy minimization questions for spheres, has several spectacular results. Their energy minimization approach is extremely efficient yielding 150 digits of accuracy. Such accuracy allows for a broader application of the LLL algorithm to "exactify" results. This is important as a step in transforming numerically produced results into solid existence proofs. We hope to better understand the approach and results in this paper and apply them to our study of points on $SO(n)$ and other manifolds (with the goal of including these results in the paper version of this abstract).

We feel that there are many interesting research projects waiting to be carried out that are related to minimal energy distributions on a wide variety of other manifolds that we have not yet considered such as Nested Spheres, Tori, Grassmann Varieties, Flags, Stiefel manifolds, Lie Groups (other than $SO(n)$), etc. It is hoped that new combinatorial objects of interest can be discovered in this manner. Already, several new codes have been found by this method and several classical combinatorial configurations have been rediscovered (as minimal distance graphs) while considering points on spheres [3]. It is exciting to think about what may lie around the corner, waiting to be discovered, when considering point distributions on other manifolds. In any case, we find that the study of these objects allows a pleasant mix of tools (analytic, algebraic, combinatoric) and computational approaches (both numeric and symbolic) and that the problems are accessible, appealing and interesting.

References

[1] T. Aste and D. Weaire, *The Pursuit of Perfect Packing*, Institute of Physics Publishing, London, 2000.

[2] J. Baker, *Integration over Spheres and the Divergence Theorem for Balls*, The AMerican Mathematical Monthly, Vol. 104 (1997), no. 1, 36-47.

[3] B. Ballinger, G. Blekherman, H. Cohn, N. Giansiracusa, E. Kelly, A. Schuermann, *Experimental study of energy-minimizing point configurations on spheres.*, Arxiv math/0611451

[4] K. Bezdek, *Sphere packings revisited*, European J. Combin. 27 (2006), no. 6, 864–883.

[5] P. Biran, *A stability property of symplectic packing*, Invent. Math. 136 (1999), no. 1, 123–155.

[6] A.R. Calderbank, R.H. Hardin, E.M. Rains, P.W. Shor, N.J.A. Sloane, *A group-theoretic framework for the construction of packings in Grassmannian spaces*, J. Algebraic Combin. 9 (1999), no. 2, 129–140.

[7] H. Cohn, *New upper bounds on sphere packings II*, Geom. Topol. 6 (2002), 329–353.

[8] H. Cohn, N. Elkies, *New upper bounds on sphere packings I*, Ann. of Math. (2) 157 (2003), no. 2, 689–714.

[9] H. Cohn and A. Kumar, *Universally Optimal Distribution of Points on Spheres*, J. Amer. Math. Soc. 20 (2007), 99-148.

[10] J.H. Conway and N.J.A. Sloane, *Sphere Packings, Lattices and Groups*, Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences], 290. Springer-Verlag, New York, 1999. lxxiv+703 pp. ISBN: 0-387-98585-9

[11] J.H. Conway, R.H. Hardin, N.J.A. Sloane, *Packing Lines, Planes, etc: Packings in Grassmannian Spaces*, Journal of Experimental Mathematics 5 (1996), 139-159.

[12] N. Elkies, *Lattices, linear codes, and invariants. I*, Notices Amer. Math. Soc. 47 (2000), no. 10, 1238–1245.

[13] GAP, The GAP Group, *GAP – Groups, Algorithms, and Programming, Version 4.4.10*; 2007, (`http://www.gap-system.org`).

[14] T.C. Hales, *Sphere packings. I*, Discrete Comput. Geom. 17 (1997), no. 1, 1–51.

[15] D.P. Hardin, E.B. Saff, *Minimal Riesz energy point configurations for rectifiable d-dimensional manifolds*, Adv. Math. 193 (2005), no. 1, 174–204.

[16] H.A. Helfgott, A. Venkatesh, *Integral points on elliptic curves and 3-torsion in class groups* J. Amer. Math. Soc. 19 (2006), no. 3, 527–550

[17] O. Henkel, *Sphere-packing bounds in the Grassmann and Stiefel manifolds*, IEEE Trans. Inform. Theory 51 (2005), no. 10, 3445–3456.

[18] A.K. Lenstra, H.W. Lenstra, L. Lovász, *Factoring polynomials with rational coefficients*, Math Ann 261 (1982), no. 4, 515-534.

[19] J. Martinet, Jacques, *Perfect lattices in Euclidean spaces*, Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences], 327. Springer-Verlag, Berlin, 2003. xxii+523 pp. ISBN: 3-540-44236-7

[20] Oleg Musin, *The Kissing Number in Four Dimensions*, To Appear: Annals of Mathematics.

[21] F. Pfender, G.M. Ziegler, *Kissing numbers, sphere packings, and some unexpected proofs*, Notices Amer. Math. Soc. 51 (2004), no. 8, 873–883.

[22] P.W. Shor, N.J.A. Sloane, *A family of optimal packings in Grassmannian manifolds*, J. Algebraic Combin. 7 (1998), no. 2, 157–163.

[23] M. Skoge, A. Donev, F.H. Stillinger, S. Torquato, *Packing hyperspheres in high-dimensional Euclidean spaces*, Phys. Rev. E (3) 74 (2006), no. 4, 11 pp.

[24] L. Zheng, David N.C. Tse, *Communication on the Grassmann manifold: a geometric approach to the noncoherent multiple-antenna channel*, IEEE Trans. Inform. Theory 48 (2002), no. 2, 359–383.

[25] C. Zong, *Sphere packings*, Universitext. Springer-Verlag, New York, 1999. xiv+241 pp. ISBN: 0-387-98794-0

Solving the separation problem for two ellipsoids involving only the evaluation of six polynomials

Laureano Gonzalez-Vega*, Esmeralda Mainar†
Departamento de Matematicas, Estadistica y Computacion
Universidad de Cantabria, Spain
E_mail: laureano.gonzalez@unican.es, esmeralda.mainar@unican.es

Abstract

By using several tools coming from Real Algebraic Geometry and Computer Algebra (Sturm–Habicht sequences), a new condition for the separation of two ellipsoids in three-dimensional Euclidean space is introduced. This condition is characterized by a set of equalities and inequalities depending only on the matrices defining the two considered ellipsoids and does not require in advance the computation (or knowledge) of the intersection points between them.

Moreover, this characterization is specially well adapted for computationally treating the case where the considered ellpsoids depend on one or several parameters. But specific techniques for dealing with the big involved expressions when rational motions are involved are required.

Introduction

The problem of detecting the collisions or overlap of two ellipsoids is of interest to robotics, CAD/CAM, computer animation, etc., where ellipsoids are often used for modelling (or enclosing) the shape of the objects under consideration. The problem to be considered here is obtaining closed formulae characterizing the separation of two ellipsoids in the three dimensional real affine space by using several tools coming from Real Algebraic Geometry and Computer Algebra. Moreover this characterization will provide easily the manipulation of the formulae for exact collision detection of two ellipsoids under rational motions.

Note that the problem considered in this paper is not the computation of the intersection points between the two considered ellipsoids. This intersection problem can be solved by any numerical nonlinear solver or by "ad–hoc" methods. Nevertheless, the results later described can be used as a preprocessing step since any intersection problem is highly simplified if the structure of the intersection set is known in advance.

Let

$$\mathcal{A} = \{(x,y,z) \in \mathbb{R}^3 \colon a_{11}x^2 + a_{22}y^2 + a_{33}z^2 + 2a_{12}xy + 2a_{13}xz + 2a_{23}yz + 2a_{14}x + 2a_{24}y + 2a_{34}y + a_{44} = 0\}$$

be the equation of an ellipsoid. As usual it can be rewritten as $X^TAX = 0$, where $X^T = (x, y, z, 1)$ and $A = (a_{ij})_{4\times 4}$ is the symmetric matrix of coefficients normalized so that $X_0^TAX_0 < 0$ for the interior points of $\mathcal{A}$. Considering two ellipsoids $\mathcal{A}$ and $\mathcal{B}$ given by $X^TAX = 0$ and $X^TBX = 0$ and, following the notation in [7] and [6], the degree four polynomial $f(\lambda) = \det(\lambda A + B)$ is called the *characteristic polynomial* of the pencil $\lambda A+B$. In [7] and [6] the authors give some partial results about how two ellipsoids intersect (without computing the intersection points), obtaining a complete characterization, in terms of the sign of the real roots of the characteristic polynomial, of the separation case:

1. The characteristic equation $f(\lambda) = 0$ always has at least two negative roots.

*Partially supported by the Spanish Ministerio de Educacion y Ciencia grant MTM2005-08690-C02-02.
†Partially supported by the Spanish Ministerio de Educacion y Ciencia grant BFM2003–03510.

2. The two ellipsoids are separated by a plane if and only if $f(\lambda) = 0$ has two distinct positive roots.

3. The two ellipsoids touch externally if and only if $f(\lambda) = 0$ has a positive double root.

It is important to notice that in these characterization conditions only the signs of the real roots are important and that their exact value is not needed. As soon as two distinct positive roots are detected, one concludes that the two ellipsoids are separated.

By using Sturm–Habicht sequences (as done in [3] for the ellipses case), the conditions the coefficients of $f(\lambda)$ must verify in order to have exactly two positive real roots are determined. These conditions provide the searched closed formulae depending only on the entrees of matrices A and B and characterizing when two ellipsoids are separated. The main difference with the approach in [2, 8] is the fact that, when the two ellipsoids depend on a parameter t, the curve $f(t;\lambda) = 0$ does not need to be analyzed: only the study of the real roots of six polynomials in t are required.

The approach presented in this paper is specially well suited for analyzing the relative position of two ellipsoids depending on a parameter t. For example, let $\mathcal{A}$ and $\mathcal{B}$ be two ellipsoids that depend on a parameter t in the following way:

$$\mathcal{A}(t)\colon X^T A(t) X = 0, \quad \mathcal{B}(t)\colon X^T B(t) X = 0.$$

In this case the characteristic polynomial $f(t;\lambda) = \det(\lambda A(t) + B(t))$ is a degree four polynomial in λ whose coefficients depend on the parameter t. The sign of the real roots of the characteristic polynomial is the only information needed: the behaviour of the real roots of $f(t;\lambda)$ by using algebraic techniques and without requiring the knowledge of any approximation of those roots will provide easy to manipulate formulae in t specially suited in order to characterize when $\mathcal{A}(t)$ and $\mathcal{B}(t)$ are separated in terms of t.

1 Characterization of the sign behaviour of the real roots of the characteristic polynomial in terms of its coefficients

In order to characterize when two ellipsoids are separated, the first step is the study of the sign of the real roots of its characteristic polynomial. The main tools (coming from Computer Algebra and Real Algebraic Geometry) to solve the sign behaviour problem before described will be the Sturm–Habicht sequence and the sign determination scheme.

1.1 Sturm–Habicht sequence

This section is devoted to introduce the definition of the Sturm–Habicht coefficients and their main properties related with the real root counting and sign determination problems. Sturm–Habicht sequence (and coefficients) was introduced in [4]; proofs of the results summarized into this section can be found in [4] or [5].

Definition 1.1.
Let P, Q be polynomials in $\mathbb{R}[x]$ and $p, q \in \mathbb{N}$ with $\deg(P) \leq p$ and $\deg(Q) \leq q$:

$$P = \sum_{k=0}^{p} a_k x^k, \qquad Q = \sum_{k=0}^{q} b_k x^k.$$

If $i \in \{0, \ldots, \inf(p,q)\}$ then the polynomial subresultant associated to P, p, Q and q of index i is defined as follows:

$$\mathbf{Sres}_i(P, q, Q, q) = \sum_{j=0}^{i} M_j^i(P, Q) x^j$$

where every $M_j^i(P,Q)$ is the determinant of the matrix built with the columns 1, 2, …, $p+q-2i-1$ and $p+q-i-j$ in the matrix:

$$m_i(P,p,Q,q) = \overbrace{\begin{pmatrix} a_p & \dots & a_0 & & \\ & \ddots & & \ddots & \\ & & a_p & \dots & a_0 \\ b_q & \dots & b_0 & & \\ & \ddots & & \ddots & \\ & & b_q & \dots & b_0 \end{pmatrix}}^{p+q-i}$$

The determinant $M_i^i(P,Q)$ will be called i-th principal subresultant coefficient and will be denoted by $\mathbf{sres}_i(P,p,Q,q)$.

Next definition introduces Sturm–Habicht sequence associated to P and Q as the subresultant sequence for P and $P'Q$ modulo some well precised sign changes.

Definition 1.2.
Let P and Q be polynomials in $\mathbb{R}[x]$ with $p = \deg(P)$ and $q = \deg(Q)$. Writing $v = p+q-1$ and $\delta_k = (-1)^{\frac{k(k+1)}{2}}$ for every integer k, the Sturm–Habicht sequence associated to P and Q is defined as the list of polynomials $\{\mathbf{StHa}_j(P,Q)\}_{j=0,\dots,v+1}$ where $\mathbf{StHa}_{v+1}(P,Q) = P$, $\mathbf{StHa}_v(P,Q) = P'Q$ and for every $j \in \{0,\dots,v-1\}$:

$$\mathbf{StHa}_j(P,Q) = \delta_{v-j}\mathbf{Sres}_j(P,v+1,P'Q,v).$$

For every j in $\{0,\dots,v+1\}$ the principal j–th Sturm–Habicht coefficient is defined as:

$$\mathbf{stha}_j(P,Q) = \operatorname{coef}_j(\mathbf{StHa}_j(P,Q))$$

In case $Q = 1$, the notations $\mathbf{StHa}_j(P) = \mathbf{StHa}_j(P,1)$ and $\mathbf{stha}_j(P) = \mathbf{stha}_j(P,1)$ are to be used.

Sign counting on the principal Sturm–Habicht coefficients provides a very useful information about the real roots of the considered polynomial. Next definitions show which are the sign counting functions to be used in the sequel (see [4] or [5]).

Definition 1.3.
Let $\mathbb{I} = \{a_0, a_1, \dots, a_n\}$ be a list of non zero elements in $\mathbb{R}$.

- $\mathbf{V}(\mathbb{I})$ is defined as the number of sign variations in the list $\{a_0, a_1, \dots, a_n\}$,
- $\mathbf{P}(\mathbb{I})$ is defined as the number of sign permanences in the list $\{a_0, a_1, \dots, a_n\}$.

Definition 1.4.
Let $a_0, a_1, \dots, a_n$ be elements in $\mathbb{R}$ with $a_0 \neq 0$ and with the following distribution of zeros:

$$\mathbb{I} = \{a_0, a_1, \dots, a_n\} =$$

$$= \{a_0, \dots, a_{i_1}, \overbrace{0,\dots,0}^{k_1}, a_{i_1+k_1+1}, \dots, a_{i_2}, \overbrace{0,\dots,0}^{k_2}, a_{i_2+k_2+1}, , a_{i_3}, 0, \dots\dots, 0, a_{i_{t-1}+k_{t-1}+1}, \dots, a_{i_t}, \overbrace{0,\dots,0}^{k_t}\}$$

where all the a_i's that have been written are not 0. Defining $i_0 + k_0 + 1 = 0$ and:

$$\mathbf{C}(\mathbb{I}) = \sum_{s=1}^{t}\big(\mathbf{P}(\{a_{i_{s-1}+k_{s-1}+1}, \dots, a_{i_s}\}) - \mathbf{V}(\{a_{i_{s-1}+k_{s-1}+1}, \dots, a_{i_s}\})\big) + \sum_{s=1}^{t-1}\varepsilon_{i_s}$$

where:

$$\varepsilon_{i_s} = \begin{cases} 0 & \text{if } k_s \text{ is odd} \\ (-1)^{\frac{k_s}{2}}\operatorname{sign}\left(\frac{a_{i_s+k_s+1}}{a_{i_s}}\right) & \text{if } k_s \text{ is even} \end{cases}$$

Next the relation between the real zeros of a polynomial $P \in \mathbb{R}[x]$ and the polynomials in the Sturm–Habicht sequence of P is presented. Its proof can be found in [4] or [5].

Definition 1.5.
Let $P, Q \in \mathbb{R}[x]$ with $p = \deg(P)$ and $\epsilon \in \{-, 0, +\}$. Then:

$$c_\epsilon(P; Q) = \operatorname{card}(\{\alpha \in \mathbb{R} : P(\alpha) = 0,\ \operatorname{sign}(Q(\alpha)) = \epsilon\}).$$

With this definition, $c_+(P; 1)$ represents the number of real roots of P and $c_-(P; 1) = 0$.

Theorem 1.1.
If P is a polynomial in $\mathbb{R}[x]$ with $p = \deg(P)$ then:

$$\mathbf{C}(\{\mathbf{stha}_p(P, Q), \ldots, \mathbf{stha}_0(P, Q)\}) = c_+(P; Q) - c_-(P; Q).$$

$$\mathbf{C}(\{\mathbf{stha}_p(P), \ldots, \mathbf{stha}_0(P)\}) = \#\{\alpha \in \mathbb{R} : P(\alpha) = 0\}.$$

In particular, the number of real roots of P is determined exactly by the signs of the last $p-1$ determinants $\mathbf{stha}_i(P)$ (the first two ones are $\operatorname{lcof}(P)$ and $p \operatorname{lcof}(P)$ with $\operatorname{lcof}(P)$ denoting the leading coefficient of P).

The definition of Sturm–Habicht sequence through determinants allows to perform computations dealing with real roots in a generic way: if P and Q are two polynomials with parametric coefficients whose degrees do not change after specialization then the Sturm–Habicht sequence for P and Q can be computed without specializing the parameters and the result is always good after specialization (modulo the condition over the degrees). This is not true when using Sturm sequences (the computation of the euclidean remainders makes to appear denominators which can vanish after specialization) or negative polynomial remainder sequences (with fixed degree for P the sequence has not always the same number of elements).

For the concrete problem considered here, the using of the results presented in this section allows to characterize when the characteristic polynomial $f(\lambda) = \det(\lambda A + B)$ has a fixed number of real roots. In order to deal with the sign of these real roots, it is needed to use the sign determination scheme (together with Theorem 1.1) which is next presented.

1.2 The sign determination scheme

Let P and Q be polynomials in $\mathbb{R}[x]$. The problem to solve by the so called "sign determination scheme" is the determination of the signs of the evaluation of Q on the real roots of P in a purely formal way without requiring the knowledge of the real roots of P. Denote

$$\mathcal{V}(P, Q) = \mathbf{C}(\{\mathbf{stha}_p(P, Q), \ldots, \mathbf{stha}_0(P, Q)\})$$

and according to Theorem 1.1:

$$\mathcal{V}(P, Q) = c_+(P; Q) - c_-(P; Q). \tag{1}$$

Since $\mathcal{V}(P, 1) = c_+(P; 1) - c_-(P; 1) = c_+(P; 1)$ agrees with the number of real roots of P then

$$\mathcal{V}(P, 1) = c_0(P; Q) + c_+(P; Q) + c_-(P; Q) \tag{2}$$

because if α is a real root of P then $Q(\alpha) = 0$ or $Q(\alpha) > 0$ or $Q(\alpha) < 0$. Applying again Theorem 1.1,

$$\mathcal{V}(P, Q^2) = c_+(P; Q^2) - c_-(P; Q^2) = c_+(P; Q^2) - 0 = c_+(P; Q^2) = c_+(P; Q) + c_-(P; Q) \tag{3}$$

because if α is a real root of P such that $Q^2(\alpha) > 0$ then $Q(\alpha) > 0$ or $Q(\alpha) < 0$.

Putting together equations (1), (2) and (3), it is obtained

$$\begin{aligned} c_0(P; Q) + c_+(P; Q) + c_-(P; Q) &= \mathcal{V}(P, 1) \\ c_+(P; Q) - c_-(P; Q) &= \mathcal{V}(P, Q) \\ c_+(P; Q) + c_-(P; Q) &= \mathcal{V}(P, Q^2) \end{aligned}$$

and the matricial identity

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & -1 \\ 0 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} c_0(P;Q) \\ c_+(P;Q) \\ c_-(P;Q) \end{bmatrix} = \begin{bmatrix} \mathcal{V}(P,1) \\ \mathcal{V}(P,Q) \\ \mathcal{V}(P,Q^2) \end{bmatrix} \tag{4}$$

allowing to compute $c_0(P;Q)$, $c_+(P;Q)$ and $c_-(P;Q)$ once $\mathcal{V}(P,1)$, $\mathcal{V}(P,Q)$ and $\mathcal{V}(P,Q^2)$ are known. But these integer numbers are directly obtained from the Sturm–Habicht sequences of P and 1, Q and Q^2 by applying the **C** function as shown by Theorem 1.1 and Definition 1.4.

When P and Q have no common roots, then $c_0(P;Q) = 0$ and the the matricial identity in (4) reduces to

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} c_+(P;Q) \\ c_-(P;Q) \end{bmatrix} = \begin{bmatrix} \mathcal{V}(P,1) \\ \mathcal{V}(P,Q) \end{bmatrix}. \tag{5}$$

More information about the sign determination scheme including historical remarks and the generalization to more than one polynomial can be found in [1].

1.3 The study of the signs of the real roots of the characteristic polynomial

The shown techniques presented in subsections 1.1 and 1.2 are going to be applied here to give a condition characterizing that the polynomial

$$P = x^4 + ax^3 + bx^2 + cx + d$$

has two real positive roots, in terms of the coefficients a, b, c and d.

First, the non trivial principal Sturm–Habicht coefficients associated to P are determined:

$$\begin{aligned}
\mathbf{stha}_2(P) &= -8b + 3a^2, \\
\mathbf{stha}_1(P) &= -8b^3 + 2a^2b^2 + 32bd + 28cab - 12a^2d - 6ca^3 - 36c^2 \\
\mathbf{stha}_0(P) &= -27d^2a^4 - 4a^3c^3 + 18a^3dcb - 6a^2c^2d - 4a^2b^3d + 144a^2bd^2 + a^2c^2b^2 - 80ab^2cd - 192ad^2c \\
&\quad +18ac^3b - 128d^2b^2 + 144c^2bd - 27c^4 + 256d^3 - 4b^3c^2 + 16db^4.
\end{aligned}$$

Next, in order to study the sign of the real roots of the polynomial $P = x^4 + ax^3 + bx^2 + cx + d$, the polynomials P and $Q = x$ are considered. Thus, the principal Sturm–Habicht coefficients associated to P and Q are computed:

$$\begin{aligned}
\mathbf{stha}_3(P,Q) &= -4a, \\
\mathbf{stha}_2(P,Q) &= 4ba^2 + 12ac - 16b^2, \\
\mathbf{stha}_1(P,Q) &= -12da^3b + 16a^3c^2 - 4b^2ca^2 + 28dca^2 + 48dab^2 + 64ad^2 - 72ac^2b - 192dcb + 16b^3c + 108c^3, \\
\mathbf{stha}_0(P,Q) &= 72d^2a^3bc - 16da^3c^3 - 16d^2a^2b^3 + 4b^2c^2da^2 + 576bd^3a^2 - 24d^2c^2a^2 - 320d^2ab^2c - 108a^4d^3 \\
&\quad +72dabc^3 - 768ad^3c + 64b^4d^2 - 16b^3c^2d - 512b^2d^3 + 576d^2c^2b + 1024d^4 - 108dc^4
\end{aligned}$$

Since the integer numbers $\mathcal{V}(P,1)$ and $\mathcal{V}(P,Q)$ depend, only and respectively, on the signs of

1. $\{\mathbf{stha}_4(P,1), \mathbf{stha}_3(P,1), \mathbf{stha}_2(P,1), \mathbf{stha}_1(P,1), \mathbf{stha}_0(P,1)\}$, and
2. $\{\mathbf{stha}_4(P,Q), \mathbf{stha}_3(P,Q), \mathbf{stha}_2(P,Q), \mathbf{stha}_1(P,Q), \mathbf{stha}_0(P,Q)\}$,

therefore, in order to get the values of $\mathcal{V}(P,1)$ and $\mathcal{V}(P,Q)$, it is enough to study the six polynomials:

$$\begin{aligned}
p_1 &= -8b + 3a^2, \\
p_2 &= -4b^3 + a^2b^2 + 16bd + 14cab - 6a^2d - 3ca^3 - 18c^2, \\
p_3 &= -27d^2a^4 - 4a^3c^3 + 18a^3dcb + a^2c^2b^2 + 144a^2bd^2 - 6a^2c^2d - 4a^2b^3d - 192ad^2c \\
&\quad -80ab^2cd + 18ac^3b - 27c^4 + 144c^2bd + 256d^3 - 128d^2b^2 + 16db^4 - 4b^3c^2, \\
q_1 &= a, \\
q_2 &= ba^2 + 3ac - 4b^2, \\
q_3 &= 4a^3c^2 - 3da^3b + 7dca^2 - b^2ca^2 + 12dab^2 - 18ac^2b + 16ad^2 + 27c^3 - 48dcb + 4b^3c.
\end{aligned}$$

{1}	[[4, 2], [1]]	{2}	[[4, 2], [4]]	{3}	[[4, 2], [5]]	{4}	[[4, 2], [7]]
{5}	[[4, 2], [8]]	{6}	[[4, 2], [9]]	{7}	[[4, 2], [11]]	{8}	[[4, 2], [13]]
{9}	[[4, 2], [14]]	{10}	[[4, 2], [15]]	{11}	[[4, 2], [16]]	{12}	[[4, 2], [17]]
{13}	[[4, 2], [18]]	{14}	[[4, 2], [21]]	{15}	[[4, 2], [23]]	{16}	[[4, 2], [24]]
{17}	[[4, 2], [25]]	{18}	[[4, 2], [26]]	{19}	[[4, 2], [27]]	{20}	[[3, 2], [36]]
{21}	[[3, 2], [37]]	{22}	[[3, 2], [42]]	{23}	[[3, 2], [45]]	{24}	[[3, 2], [48]]
{25}	[[3, 2], [49]]	{26}	[[3, 2], [50]]	{27}	[[3, 2], [51]]	{28}	[[3, 2], [54]]

{1}	[1,1,1,1,1,1]	{2}	[1,1,1,1,0,1]	{3}	[1,1,1,1,0,0]	{4}	[1,1,1,1,-1,1]
{5}	[1,1,1,1,-1,0]	{6}	[1,1,1,1,-1,-1]	{7}	[1,1,1,0,1,0]	{8}	[1,1,1,0,0,1]
{9}	[1,1,1,0,0,0]	{10}	[1,1,1,0,0,-1]	{11}	[1,1,1,0,-1,1]	{12}	[1,1,1,0,-1,0]
{13}	[1,1,1,0,-1,-1]	{14}	[1,1,1,-1,1,-1]	{15}	[1,1,1,-1,0,0]	{16}	[1,1,1,-1,0,-1]
{17}	[1,1,1,-1,-1,1]	{18}	[1,1,1,-1,-1,0]	{19}	[1,1,1,-1,-1,-1]	{20}	[1,1,0,1,-1,-1]
{21}	[1,1,0,0,1,1]	{22}	[1,1,0,0,0,-1]	{23}	[1,1,0,0,-1,-1]	{24}	[1,1,0,-1,1,-1]
{25}	[1,1,0,-1,0,1]	{26}	[1,1,0,-1,0,0]	{27}	[1,1,0,-1,0,-1]	{28}	[1,1,0,-1,-1,-1]

Table 1: Sign conditions for the polynomials p_1, p_2, p_3, q_1, q_2, q_3 implying the separation of the ellipsoids.

In the concrete case considered here, the polynomial P represents the characteristic polynomial of the pencil $\lambda A + B$ once it has been transformed into a monic polynomial $P(\lambda) = -\frac{f(\lambda)}{k}$ with $k > 0$.

There are $3^6 = 729$ possibilities of sign conditions in the polynomial sequence $\{p_1, p_2, p_3, q_1, q_2, q_3\}$. The sign determination scheme (see, for example, [1, 3]) produces a list $[[a, b], [n]]$ $1 \leq n \leq 729$, indicating that in the n–th element of the list, P has a total of a different real roots and b of them are positive. For example, $[[3, 1], [5]]$ means that the fifth case P has 3 different real roots and just one is positive. Taking into account that the characteristic polynomial P of the pencil $\lambda A + B$ has always two negative roots (counting multiplicities) at least and that two ellipsoids are separated by a plane if and only if P has two distinct positive roots, the cases to be considered are only those producing $[[4, 2], [n]]$ and $[[3, 2], [n]]$.

This process, completely automatized by using the Computer Algebra System `Maple`, produces the following 28 possibilities (see Table 1) which completely characterize the separation of the two considered ellipsoids. A simple inspection allows to check that all elements of

$$\begin{aligned}
&[1, 1, 1, *, -1, *] := \{[1, 1, 1, n, -1, m], n \in \{-1, 0, 1\}, m \in \{-1, 0, 1\}\},\\
&[1, 1, 0, -1, 0, *] := \{[1, 1, 0, -1, 0, n], n \in \{-1, 0, 1\}\},\\
&[1, 1, 0, *, -1, -1] := \{[1, 1, 0, n, -1, -1], n \in \{-1, 0, 1\}\},
\end{aligned}$$

are included in Table 1. In fact, denoting $[1, 1, 1, n \neq 0, 1, n \neq 0] := \{[1, 1, 1, n, 1, n], n \neq 0\}\}$, and $[1, 1, 1, n \neq 0, 0, m \neq -1] := \{[1, 1, 1, n, 0, m], n \neq 0, m \neq -1\}\}$, the 28 cases are included in

$$[1, 1, 1, *, -1, *] \cup [1, 1, 0, -1, 0, *] \cup [1, 1, 0, *, -1, -1] \cup$$

$$\cup [1, 1, 1, n \neq 0, 1, n \neq 0] \cup [1, 1, 1, n \neq 0, 0, m \neq -1] \cup [1, 1, 0, -1, 1, -1].$$

In other words, if $P = x^4 + ax^3 + bx^2 + cx + d$ represents the characteristic polynomial of the pencil $\lambda A + B$ (once turned monic) then the ellipsoids $\mathcal{A}$ and $\mathcal{B}$ are separated if and only if (a, b, c, d) verifies one of the following six conditions (matrix rows):

$$\begin{array}{cccccc}
p_1 > 0 & p_2 > 0 & p_3 > 0 & & q_2 < 0 & \\
p_1 > 0 & p_2 > 0 & p_3 = 0 & q_1 < 0 & q_2 = 0 & \\
p_1 > 0 & p_2 > 0 & p_3 = 0 & & q_2 < 0 & q_3 > 0 \\
p_1 > 0 & p_2 > 0 & p_3 = 0 & q_1 < 0 & q_2 > 0 & q_3 > 0 \\
p_1 > 0 & p_2 > 0 & p_3 > 0 & q_1 \neq 0 & q_2 > 0 & q_3 \neq 0 \\
p_1 > 0 & p_2 > 0 & p_3 > 0 & q_1 \neq 0 & q_2 = 0 & q_3 \geq 0
\end{array}$$

2 On the relative position of two parametric ellipsoids

It is worth to remark that all the results obtained in the previous section can be applied to study the case of two ellipsoids depending on one parameter. Given two moving ellipsoids $\mathcal{A}(t) : X^T A(t) X = 0$ and $\mathcal{B}(t) : X^T B(t) X = 0$ under rational motions $M_A(t)$ and $M_B(t)$, $t \in [0,1]$, respectively, $\mathcal{A}(t)$ and $\mathcal{B}(t)$ are said to be *collision-free* if $\mathcal{A}(t)$ and $\mathcal{B}(t)$ are separated for all $t \in [0,1]$; otherwise $\mathcal{A}(t)$ and $\mathcal{B}(t)$ collide.

The characteristic equation of $\mathcal{A}(t)$ and $\mathcal{B}(t)$, $t \in [0,1]$

$$f(\lambda; t) := \det(\lambda A(t) + B(t)) = 0$$

is a degree four polynomial in λ with real coefficients depending on the parameter t.

At any time $t_0 \in [0,1]$, if $\mathcal{A}(t_0)$ and $\mathcal{B}(t_0)$ are separated $f(\lambda; t_0)$ has two distinct positive roots; otherwise $\mathcal{A}(t_0)$ and $\mathcal{B}(t_0)$ are either touching externally or overlapping, and $f(\lambda; t_0)$ has a double positive root or no positive roots, respectively.

In order to determine the relative position of the ellipsoids, the study of the sign behaviour of the roots of the characteristic polynomial for all the possible values of the parameter t is required. This is accomplished by using the techniques presented in Section 1.3 where the analysis of the possible sign conditions verified by six polynomials in the coefficients of $f(t;\lambda)$ (as polynomial in λ) produces in an automatic manner (and in terms of t) which is the behaviour of the sign of the real roots of $f(t;\lambda)$.

Example 2.1.
Let $\mathcal{A}(t)$ and $\mathcal{B}(t)$ be two spheres, depending on $t \in \mathbb{R}$, defined by the equations

$$x^2(t^2+1) + y^2(t^2+1) + z^2 = 1, \quad (x-t)^2 + y^2 + z^2 = 1.$$

$\mathcal{A}(t)$ is the set of concentric spheres of radius less or equal to 1 and $\mathcal{B}(t)$ is a (radius 1) sphere whose centre moves along the axis x.

The matrices associated to $\mathcal{A}(t)$ and $\mathcal{B}(t)$ are in this case:

$$A(t) = \begin{pmatrix} t^2+1 & 0 & 0 & 0 \\ 0 & t^2+1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}, \quad B(t) = \begin{pmatrix} 1 & 0 & 0 & -t \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -t & 0 & 0 & t^2-1 \end{pmatrix}$$

and the characteristic polynomial of the pencil $\lambda A(t) + B(t)$:

$$f(t;\lambda) = \det(\lambda A(t) + B(t)) = (-2t^2-1-t^4)\lambda^4 + (t^6-5t^2-4)\lambda^3 + (2t^4-4t^2-6+t^6)\lambda^2 + (-4-t^2+t^4)\lambda - 1.$$

Turning $f(t;\lambda)$ into a monic polynomial (with respect to λ) produces the following coefficients

$$a = \frac{t^6-5t^2-4}{-2t^2-1-t^4}, \quad b = \frac{2t^4-4t^2-6+t^6}{-2t^2-1-t^4}, \quad c = \frac{-4-t^2+t^4}{-2t^2-1-t^4}, \quad d = \frac{-1}{-2t^2-1-t^4}.$$

According to the results in subsection 1.3, the sign behaviour of the real roots of $f(t;\lambda)$ is determined by the sign conditions verified by the polynomials

$$\begin{aligned}
p_1 &:= t^2(3t^6+2t^4-5t^2-8)/(t^2+1)^2, \\
p_2 &:= t^6(t^{10}+t^8-3t^6-7t^4-7t^2-4)/(t^2+1)^4, \\
p_3 &:= t^{14}(t^3+2t^2+2t+2)(t^3-2t^2+2t-2)/(t^2+1)^6, \\
q_1 &:= -(-4-t^2+t^4)/(t^2+1), \\
q_2 &:= -t^2(t^{10}+3t^8-5t^6-12t^4-t^2+8)/(t^2+1)^3, \\
q_3 &:= t^6(-4-t^2+t^4)(t^{10}-5t^6-7t^4-4t^2-2)/(t^2+1)^6.
\end{aligned}$$

In the concrete problem considered here, once denominators and those factors without real roots and constant sign are removed the following polynomials are obtained

$$\begin{aligned}
&3t^6+2t^4-5t^2-8, t^{10}+t^8-3t^6-7t^4-7t^2-4, t^3-2t^2+2t-2, \\
&4+t^2-t^4, -t^10-3t^8+5t^6+12t^4+t^2-8, (-4-t^2+t^4)(t^10-5t^6-7t^4-4t^2-2).
\end{aligned}$$

Next the real roots of these polynomials are computed producing the following results:

- The real roots of $3t^6 + 2t^4 - 5t^2 - 8$ is 1.240967508.
- The real roots of $t^{10} + t^8 - 3t^6 - 7t^4 - 7t^2 - 4$ is 1.52066394.
- The real root of $t^3 - 2t^2 + 2t - 2$ is 1.543689012.
- The real root of $4 + t^2 - t^4$ is 1.600485180.
- The real root of $-t^{10} - 3t^8 + 5t^6 + 12t^4 + t^2 - 8$ are 0.8540956701 and 1.424253130.
- The real root of $(-4 - t^2 + t^4)(t^{10} - 5t^6 - 7t^4 - 4t^2 - 2)$ are 1.600485180 and 1.684484014.

together with the following description for the separation problem, in terms of t:

1. If $t \in (0, 1.600485180)$ then the ellipsoids $\mathcal{A}(t)$ and $\mathcal{B}(t)$ are not separated.
2. If $t \in (1.600485180, \infty)$ then the ellipsoids $\mathcal{A}(t)$ and $\mathcal{B}(t)$ are separated.

This information is obtained by just determining the number of real roots of $f(t; \lambda)$ when t belongs to each of the intervals defined by the real roots of the polynomials p_i and q_i.

3 Conclusions

A closed formulae requiring only the evaluation of six polynomials has been presented for characterizing the separation of two ellipsoids specially well suited when the considered ellipsoids depend on one parameter. Further analysis for the treatment of the involved polynomials here presented is required to consider the case of two moving ellipsoids under rational motions since the size of the involved polynomials require “ad–hoc techniques” for their study (see [2, 8]).

References

[1] S. Basu, R. Pollack, M.-F. Roy: *Algorithms in Real Algebraic Geometry.* Algorithms and Computations in Mathematics **10**, Springer–Verlag (2003).

[2] Y.–K. Choi, M.–S. Kim, W. Wang: *Exact collision detection of two moving ellipsoids under rational motions.* Proceedings of the 2003 IEEE International Conference on Robotics & Automation, 349–354, 2003.

[3] F. Etayo, L. Gonzalez–Vega, N. del Rio: *A new approach to characterizing the relative position of two ellipses depending on one parameter* . Computer Aided Geometric Design **23**, 324–350 (2006).

[4] L. Gonzalez–Vega, H. Lombardi, T. Recio, M.–F. Roy: *Specialisation de la suite de Sturm et sous–resultants.* (I): Informatique Theorique et Applications **24**, 561–588 (1990). (II): Informatique Theorique et Applications **28**, 1–24 (1994).

[5] L. Gonzalez–Vega, H. Lombardi, T. Recio, M.–F. Roy: *Determinants and real roots of univariate polynomials.* Quantifier Elimination and Cylindrical Algebraic Decomposition (Caviness B. and Johnson J. eds), Texts and Monographs in Symbolic Computation, 300–316, Springer–Verlag (1998).

[6] W. Wang, R. Krasauskas: *Interference analysis of conics and quadrics.* Contemporary Mathematics **334**, 25–36, AMS (2003).

[7] W. Wang, J. Wang, M.-Soo Kim: *An algebraic condition for the separation of two ellipsoids.* Computer Aided Geometric Design **18**, 531–539 (2001).

[8] W. Wang, Y.-K. Choi, B. Chan, M.-S. Kim, J. Wang: *Efficient collision detection for moving ellipsoids using separating planes.* Geometric modelling. Computing 72, 1-2, 235–246 (2004).

DETERMINING WHEN PROJECTIONS ARE THE ONLY HOMOMORPHISMS

DAVID CASPERSON

1. Introduction

Herein, an algebra is meant in the sense of universal algebra, that is a set equipped with operations that interpret the symbols of some fixed language.

Some properties of entire families of algebras are equivalent to the existence or non-existence of term functions with certain properties. For instance, a variety of algebras is congruence permutable if and only if there exists a ternary term $p(x, y, z)$ with the property that $p(x, x, y) = y$ and $p(x, y, y) = x$. As being congruence permutable implies the existence of a Jordan-Hölder-like theorem, we know that groups have a Jordan-Hölder theorem because the term $p(x, y, z) = xy^{-1}z$ establishes that they are congruence permutable.

For a finite algebra $\mathbf{A}$, the number of ternary term functions is at most $|\mathrm{A}|^{|\mathrm{A}|^3}$. This means that the question of whether or not the question of whether or not the variety of algebras generated by a single finite algebra is congruence permutable *can be reduced to a computation.*

In this paper we establish a similar kind of result for a different question about quasivarieties of algebras.

2. The Question

Here, we consider the following question. Given a finite algebra $\mathbf{M}$, we call a homomorphism from $\mathbf{A}$, a subalgebra of a finite power of $\mathbf{M}$, to $\mathbf{M}$ itself, a (*partial*) *algebraic operation.* The question that we ask is: for what fixed $\mathbf{M}$ is every partial algebraic operation the restriction of a projection?

3. The Answer

Definition 1. Suppose that $\mathbf{M}$ is a finite algebra. If, for all finite index sets I, all $\mathbf{A} \leq \mathbf{M}^I$ and all $h \in \hom(\mathbf{A}, \mathbf{M})$, we have $h = \pi_i$ for some $i \in I$, then we say that $\mathbf{M}$ is a *projection algebra.*

Definition 2. Fix an integer $k \geq 1$. If, for all finite index sets I and all at-most k-generated $\mathbf{A} \leq \mathbf{M}^I$ and all $h \in \hom(\mathbf{A}, \mathbf{M})$, we have $h = \pi_i$, we then say that $\mathbf{M}$ is a *k-projection algebra.*

An algebra $\mathbf{M}$ is a projection algebra if and only if it is k-projection algebra for all finite k. The following propositions show that we can reduce the question of whether or not an algebra is a projection algebra from a question concerning an entire quasi-variety of algebras to the existence of certain term functions, and hence make the question *computable.*

Proposition 3. *A finite algebra $\mathbf{M}$ is a k-projection algebra if and only if for every $\vec{c} \in \mathbf{M}^k$ there are k-ary terms $\sigma_{\vec{c}}(\vec{x})$ and $\tau_{\vec{c}}(\vec{x})$ such that*

$$\sigma_{\vec{c}}(\vec{x}) \neq \tau_{\vec{c}}(\vec{x}) \qquad \textit{if and only if} \qquad \vec{c} = \vec{x}.$$

Lemma 4. *A finite algebra $\mathbf{M}$ that is a 3-projection algebra and a k-projection algebra is a $(k+1)$-projection algebra.* Consequently:

Theorem 5. *The finite algebra $\mathbf{M}$ is projection algebra if for every $\langle a, b, c\rangle \in \mathbf{M}^3$ there are ternary terms $\sigma_{a,b,c}(x, y, z)$ and $\tau_{a,b,c}(x, y, z)$ such that*

$$\sigma_{a,b,c}(x, y, z) \neq \tau_{a,b,c}(x, y, z) \qquad \textit{if and only if} \qquad \langle a, b, c\rangle = \langle x, y, z\rangle.$$

4. Computability and algorithmic questions

By Theorem 5 the question of whether or not a given finite algebra is a projection algebra can be reduced to searching the subset $\mathbf{M}^{M^3}$ generated by the projections to see whether or not for each $\vec{c} \in \mathbf{M}^3$ there is a pair of terms $(\sigma_{\vec{c}}, \tau_{\vec{c}})$ with the required property. This gives an exponential *upper bound* on the complexity of determining whether $\mathbf{M}$ is a projection algebra. Though still exponential, we can substantially reduce this bound if there is a positive answer to the following unsolved problem.

Problem 6. *Is every* 2*-projection algebra a projection algebra?*

The upper bounds suggested above can likely be improved. When the subset $\mathbf{M}^{M^3}$ generated by the projections is large, $\hom(\mathbf{M}^{M^3}, \mathbf{M})$ is relatively speaking small, and it may be possible to compute directly that this consists of projections only (which suffices for reason found in the proof of Proposition 4).

Thus, we have shown that the question of whether or not a finite algebra is a projection algebra is computable, and indicated directions to pursue with regard to finding computationally tractable algorithms and heuristics.

Department of Computer Science, University of Northern British Columbia, Prince George, BC V2N 4Z9, Canada

E-mail address: `casper@unbc.ca`

Automatic Variable Order Selection for Polynomial System Solving

Mark Giesbrecht[1], John May[2], Marc Moreno Maza[3],
Daniel Roche[1], Yuzhen Xie[1]

[1]David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada, N2L 3G1
Email: {mwg,droche,yxie}@cs.uwaterloo.ca

[2]Maplesoft
615 Kumpf Drive
Waterloo, Ontario, Canada, N2V 1K8
Email: jmay@maplesoft.com

[3]Department of Computer Science
University of Western Ontario
London, Ontario, Canada, N6A 5B7
Email: moreno@csd.uwo.ca

Abstract

The goal of a general purpose solver is to allow a user to compute the solutions of a system of equations with minimal interactions. Modern tools for polynomial system solving, namely triangular decomposition and Gröbner basis computation, can be highly sensitive to the ordering of the variables. Our goal is to examine the structure of a given system and use it to compute a variable ordering that will cause the solving algorithm to complete quickly (or alternately, to give compact output). We explore methods based on the dependency graph of coincident variables and terms between the equations. Desirable orderings are gleaned from connected components and other topological properties of these graphs, under different weighting schemes.

Acknowledgement
All authors acknowledge the continuing support of Waterloo Maple Inc., and the Mathematics of Information Technology and Complex Systems (MITACS). Giesbrecht, Moreno Maza, and Xie acknowledge the support of the Natural Sciences and Engineering Research Council (NSERC) of Canada.

On the Verification of Polynomial System Solvers

Changbo Chen, Marc Moreno Maza, Wei Pan and Yuzhen Xie
University of Western Ontario, London, Ontario, Canada

We discuss the verification of mathematical software solving polynomial systems symbolically by way of triangular decomposition. Given a polynomial system F and a set of components $C_1, \ldots, C_e$, it is hard, in general, to tell whether the union of $C_1, \ldots, C_e$ corresponds exactly to the solution set $\mathbf{V}(F)$ or not. Solving this verification problem is generally (at least) as hard as solving the system F itself. In addition, different solvers can produce different but all valid triangular decompositions for the same input system.

Because of the high complexity of symbolic solvers, developing verification algorithms and reliable verification software tools is a clear need. However, this verification problem has received little attention in the literature. Checking whether $C_1, \ldots, C_e$ corresponds exactly to the solution set $\mathbf{V}(F)$ of F can be done by means of Gröbner bases computations. This verification method is quite simple, but highly expensive.

In this poster, we exhibit a new approach which manipulates constructible sets represented by regular systems. We assume that each component of the solution set $\mathbf{V}(F)$ is given by a so-called *regular system*. This is a natural assumption in symbolic computations, well-developed in the literature under different terminologies. In broad terms, a regular system consists of several polynomial equations with a triangular shape

$$p_1(x_1) = p_2(x_1, x_2) = \cdots = p_i(x_1, x_2, \ldots, x_n) = 0$$

and a polynomial inequality

$$h(x_1, \ldots, x_n) \neq 0$$

such that there exists (at least) one point $(a_1, \ldots, a_n)$ satisfying the above equations and inequality. Note that these polynomials may contain parameters.

Let us consider now an arbitrary input system F and a set of components $C_1, \ldots, C_e$. The usual approach for verifying that $C_1, \ldots, C_e$ correspond exactly to the solution set $\mathbf{V}(F)$ is as follows.

(1) First, one checks that each candidate component C_i is actually contained in $\mathbf{V}(F)$. This essentially reduces to substitute the coordinates of the points given by C_i into the polynomials of F: if all these polynomials vanish at these points, then C_i is a component of $\mathbf{V}(F)$, otherwise C_i is not a component of $\mathbf{V}(F)$.

(2) Secondly, one checks that $\mathbf{V}(F)$ is contained in the union of the candidate components $C_1, \ldots, C_e$ by:

   (2.1) computing a polynomial system G such that $\mathbf{V}(G)$ corresponds exactly to $C_1, \ldots, C_e$, and

   (2.2) checking that every solution of $\mathbf{V}(F)$ cancels the polynomials of G.

Steps (2.1) and (2.2) can be performed using standard techniques based on computations of Gröbner bases. These calculations are very expensive, as shown by our experimentation.

The main idea of our new approach is as follows. Instead of comparing a candidate set of components $C_1, \ldots, C_e$ against the input system F, we compare it against the output $D_1, \ldots, D_f$ produced by another solver. Both this solver and the comparison process are assumed to be validated. Hence, the candidate set of components $C_1, \ldots, C_e$ corresponds exactly to the solution set $\mathbf{V}(F)$ if and only if the comparison process shows that $D_1, \ldots, D_f$ and $C_1, \ldots, C_e$ define the same solution set. Checking that these two sets of components encode the same solution set boils down to compute the differences of two constructible sets.

Assume that we have at hand a reliable solver computing triangular decompositions of polynomial systems. We believe that this reliability can be acquired over time by combining several features.

- Checking the solver with a verification tool based on Gröbner bases for input systems of moderate difficulty.
- Using the solver for input systems of higher difficulty where the output can be verified by theoretical arguments.
- Involving the library supporting the solver in other applications.
- Making the solver widely available to potential users.

We provide a relatively simple, but efficient, procedure for computing the set theoretical differences between two constructible sets. We also perform comparative benchmarks of different verification procedures applied to four solvers for computing triangular decomposition of polynomial systems:

- the command `Triangularize` of the *RegularChains* library in MAPLE
- the `triade` solver of the *BasicMath* library in ALDOR
- the commands `RegSer` and `SimSer` of the *Epsilon* library in MAPLE.

We have run these four solvers on a large set of well-known input systems. For those systems for which this is feasible, we have successfully verified their computed triangular decompositions with a verification tool based on Gröbner bases computations. Then, for each input system, we have compared all its computed triangular decompositions by means of our new verification tool. Our experimental results demonstrate the high efficiency of our new approach. We are able to verify triangular decompositions of polynomial systems which are not easy to solve. In particular, our new verification tool can verify the solution set of all test polynomial systems that at least two of the four solvers can solve. Therefore, this tests indicates that the four solvers are solving tools with a high probability of correctness.

References

[1] P. Aubry, D. Lazard, and M. Moreno Maza. On the theories of triangular sets. *J. Symb. Comp.*, 28(1-2):105–124, 1999.

[2] P. Aubry and M. Moreno Maza. Triangular sets for solving polynomial systems: A comparative implementation of four methods. *J. Symb. Comp.*, 28(1-2):125–154, 1999.

[3] L. Donati and C. Traverso. Experimenting the Gröbner basis algorithm with the ALPI system. In *Proc. ISSAC'89*, pages 192–198. ACN Press, 1989.

[4] M. Kalkbrener. A generalized euclidean algorithm for computing triangular representations of algebraic varieties. *J. Symb. Comp.*, 15:143–167, 1993.

[5] The Computational Mathematics Group. *The BasicMath library* NAG Ltd, Oxford, UK. http://www.nag.co.uk/projects/FRISCO.html, 1998.

[6] F. Lemaire, M. Moreno Maza, and Y. Xie. The *RegularChains* library. In Ilias S. Kotsireas, editor, Maple Conference 2005, pages 355–368, 2005.

[7] Montserrat Manubens and Antonio Montes. Improving dispgb algorithm using the discriminant ideal, 2006.

[8] M. Moreno Maza. On triangular decompositions of algebraic varieties. Technical Report TR 4/99, NAG Ltd, Oxford, UK, 1999. http://www.csd.uwo.ca/∼moreno.

[9] J. O'Halloran and M. Schilmoeller. Gröbner bases for constructible sets. *Journal of Communications in Algebra*, 30(11), 2002.

[10] W. Sit. Computations on quasi-algebraic sets. In R. Liska, editor, *Electronic Proceedings of IMACS ACA'98*.

[11] *The SymbolicData Project*. http://www.SymbolicData.org, 2000–2006.

[12] D. Wang. Computing triangular systems and regular systems. *J. Symb. Comp.*, 30(2):221–236, 2000.

[13] D. M. Wang. *Epsilon 0.618*. http://www-calfor.lip6.fr/∼wang/epsilon.

[14] D. M. Wang. Decomposing polynomial systems into simple systems. *J. Symb. Comp.*, 25(3):295–314, 1998.

A Note on the Functional Decomposition of Symbolic Polynomials

Stephen M. Watt
Ontario Research Centre for Computer Algebra
Department of Computer Science, University of Western Ontario
London Ontario, CANADA N6A 5B7
watt@uwo.ca

It often arises that the general form of a polynomial is known, but the particular values for the exponents are unknown. For example, we may know a polynomial is of the form $3X^{(n^2+n)/2} - Y^{2m} + 2$, where n and m are integer-valued parameters. We consider the case where the exponents are multivariate integer-valued polynomials with coefficients in $\mathbb{Q}$ and call these "symbolic polynomials." Earlier work has presented algorithms to factor symbolic polynomials and compute GCDs [9, 10]. Here, we extend the notion of univariate polynomial decomposition to symbolic polynomials and presents an algorithm to compute these decompositions. For example, the symbolic polynomial $f(X) = 2X^{n^2+n} - 4X^{n^2} + 2X^{n^2-n} + 1$ can be decomposed as $f = g \circ h$ where $g(X) = 2X^2 + 1$ and $h(X) = X^{n^2/2+n/2} - X^{n^2/2-n/2}$.

Definition 1 (Multivariate integer-valued polynomial)**.** For an integral domain D with quotient field K, the (multivariate) integer-valued polynomials over D in variables $X_1, \ldots, X_n$, denoted $\mathrm{Int}_{[X_1,\ldots,X_n]}(D)$, are defined as $\mathrm{Int}_{[X_1,\ldots,X_n]}(D) = \{f \mid f \in K[X_1, \ldots, X_n] \text{ and } f(a) \in D, \text{ for all } a \in D^n\}$.

Integer-valued polynomials have been studied for many years [5, 6]. Definition 1 is the obvious multivariate generalization.

Definition 2 (Symbolic polynomial)**.** The ring of symbolic polynomials in $X_1, ..., X_v$ with exponents in $n_1, ..., n_p$ over the coefficient ring R is the ring consisting of finite sums of the form $\sum_i c_i X_1^{e_{i1}} X_2^{e_{i2}} \cdots X_v^{e_{iv}}$, where $c_i \in R$ and $e_{ij} \in \mathrm{Int}_{[n_1,n_2,...,n_p]}(\mathbb{Z})$. Multiplication is defined by $bX_1^{e_1} \cdots X_v^{e_v} \times cX_1^{f_1} \cdots X_v^{f_v} = bc\, X_1^{e_1+f_1} \cdots X_v^{e_v+f_v}$ and distributivity. We denote this ring $R[n_1, ..., n_p; X_1, ..., X_v]$.

If a univariate polynomial is regarded as a function of its variable, then we may ask whether the polynomial is the composition of two polynomial functions of lower degree. This can be useful in simplifying expressions, solving polynomial equations exactly or determining the dimension of a system. Polynomial decomposition has been studied for quite some time, with early work by Ritt and others [1, 4, 7, 8]. Algorithms for polynomial decomposition have been proposed and refined for use in computer algebra systems. Generalizations of this problem include decomposition of rational functions and algebraic functions. The relationship between polynomial composition and polynomial systems has also been studied [2, 3].

Unlike polynomial rings, symbolic polynomial rings are not closed under functional composition. For example, if $g(X) = X^n$ and $h(X) = X + 1$ then $g(h(X)) = \sum_{i=0}^{n} \binom{n}{i} X^i$ cannot be expressed in finite terms of group ring operations. We therefore make the following definition.

Definition 3 (Composition of univariate symbolic polynomials)**.** Let $g, h \in \mathcal{P} = R[n_1, ..., n_p; X]$. The composition $g \circ h$ of g and h, if it exists, is the finite sum $f = \sum_i c_i X^{e_i} \in \mathcal{P}$ such that $\phi f = \phi g \circ \phi h$ under all evaluation maps $\phi : \{n_1, ..., n_p\} \to \mathbb{Z}$.

We may now state the problem we wish to solve:

Problem 1. *Let $f \in R[n_1, ..., n_p; X]$. Determine whether there exist symbolic polynomials $g_1, ..., g_\ell \in R[n_1, ..., n_p; X]$ not of the form $c_1 X + c_0 \in R[X]$, such that $f = g_1 \circ \cdots \circ g_\ell$ and, if so, find them.*

We restrict our attention to the case where the coefficient ring is $\mathbb{C}$. This allows roots of unity when required and avoids technicalities arising when the characteristic of the coefficient field divides the degree an outer composition factor. This so-called "wild" case is less important with symbolic polynomials because degrees are not always fixed values. We then have the following result.

Theorem 1. *Let $g(X) = \sum_{i=1}^{R} g_i X^{p_i}$ and $h(X) = \sum_{i=1}^{S} h_i X^{q_i}$ be symbolic polynomials in $\mathcal{P} = \mathbb{C}[n_1, ..., n_p; X]$, with $g_i \neq 0$, $h_i \neq 0$, and with the p_i all distinct and the q_i all distinct. The functional composition $g \circ h$ exists in $\mathcal{P}$ if and only if at least one of the following conditions hold:*

Condition 1. *h is a monomial and $g \in \mathbb{C}[X, X^{-1}]$,*

Condition 2. *h is a monomial with coefficient h_1 a d-th root of unity, where d is a fixed divisor of all p_i,*

Condition 3. *$g \in \mathbb{C}[X]$.*

Based on this theorem, we may compute a decomposition of a symbolic polynomial as follows.

Algorithm 1 (Symbolic polynomial decomposition)**.**

Input: $f = \sum_{i=1}^{T} f_i X^{e_i} \in \mathcal{P} = \mathbb{C}[n_1, ..., n_p; X]$

Output: If there exists a decomposition $f = g \circ h$, $g, h \in \mathcal{P}$ not of the form $c_1 X + c_0 \in \mathbb{C}[X]$, then output **true**, g and h. Otherwise output **false**.

Step 1. *Handle the case of monomial h.*
Let $q :=$ primitive part of $\gcd(e_1, ..., e_T)$, $k := \gcd(\text{max fixed divisor } e_1, \ldots, \text{max fixed divisor } e_T)$.
If $kq \neq 1$, let $g = \sum_{i=1}^{T} f_i X^{e_i/(kq)}$ and $h = X^{kq}$. Return $(\mathbf{true}, g, h)$

Step 2. *Remove fractional coefficients that occur in f.*
Let L be smallest integer such that $Le_1, ..., Le_T \in \mathbb{Z}[n_1, ..., n_p]$. Construct $f' = \rho f \in \mathcal{P}$, using the substitution $\rho : X \mapsto X^L$.

Step 3. *Convert to multivariate problem.* Construct $f'' = \gamma f' \in \mathbb{C}[X_{0\ldots0}, ..., X_{d\ldots d}]$, using the correspondence $\gamma : X^{{n_1}^{i_1} \cdots {n_p}^{i_p}} \mapsto X_{i_1 \ldots i_p}$.

Step 4. *Determine possible degrees.* Let D be the total degree of f''. The possible degrees of the composition factors are the integers that divide D.

Step 5. *Try uni-multivariate decompositions.* For each integer divisor r of D, from largest to smallest until a decomposition is found or there are no more divisors, try a uni-multivariate Laurent polynomial decomposition $f'' = g \circ h''$ where g has degree r. If no decomposition is found, return **false**.

Step 6. *Compute h.* Invert the substitutions to obtain $h = \rho^{-1} \gamma^{-1} h''$.

Step 7. *Return* $(\mathbf{true}, g, h)$.

It may be possible to further decompose g and h. If $g \in \mathbb{C}[X]$, the standard polyomial decomposition algorithms may be applied. If $h = X^{a \times b}$, then h may be decomposed as $X^a \circ X^b$.

Some interesting problems remain open to future investigation: One is to decompose symbolic polynomials over fields of finite characteristic. Another is to compute the functional decomposition of extended symbolic polynomials, where elements of the coefficient ring may have symbolic exponents.

[1] D. R. Barton and R. E. Zippel. A polynomial decomposition algorithm. In *Proc. 1976 ACM Symposium on Symbolic and Algebraic Computation*, pages 356–358. ACM Press, 1976.

[2] H. Hong. Subresultants under composition. *J. Symbolic Computation*, 23:355–365, 1997.

[3] H. Hong. Groebner basis under composition I. *J. Symbolic Computation*, 25:643–663, 1998.

[4] D. Kozen and S. Landau. Polynomial decomposition algorithms. *J. Symbolic Computation*, 22:445–456, 1989.

[5] A. Ostrowski. Über ganzwertige Polynome in algebraischen Zahlköpern. *J. Reine Angew. Math.*, 149:117–124, 1919.

[6] G. Pólya. Über ganzwertige Polynome in algebraischen Zahlköpern. *J. Reine Angew. Math.*, 149:97–116, 1919.

[7] J. Ritt. Prime and composite polynomials. *Trans. American Math. Society*, 23(1):51–66, 1922.

[8] J. von zur Gathen, J. Gutierrez, and R. Rubio. Multivariate polynomial decomposition. *Applied Algebra in Engineering, Communication and Computing*, 14:11–31, 2003.

[9] S. Watt. Making computer algebra more symbolic. In *Proc. Transgressive Computing 2006: A conference in honor of Jean Della Dora*, pages 43–49, 2006.

[10] S. Watt. Two families of algorithms for symbolic polynomials. In I. Kotsireas and E. Zima, editors, *Computer Algebra 2006: Latest Advances in Symbolic Algorithms – Proceedings of the Waterloo Workshop*, pages 193–210. World Scientific, 2007.

A Preliminary Report on the Set of Symbols Occurring in Engineering Mathematics Texts

Stephen M. Watt
Ontario Research Centre for Computer Algebra
Department of Computer Science
University of Western Ontario
London Ontario, CANADA N6A 5B7
Stephen.Watt@uwo.ca

Certain forms of mathematical expression are used more often than others in practice. We propose that a quantitative understanding of actual usage can provide information to improve the accuracy of software for the input of mathematical expressions from scanned documents or handwriting and allow more natural forms of presentation of mathematical expressions by computer algebra systems. Earlier work [1] has examined this question for the diverse set of articles from the mathematics preprint archive `arXiv.org`. That analysis showed showed the variance between mathematical areas. The present work analyzes a particular mathematical domain more deeply. We have chosen to examine second year university engineering mathematics as taught in North America as the domain. This syllabus typically includes linear algebra, complex analysis, Fourier analysis, vector calculus, and ordinary and partial differential equations. We have analyzed the set of expressions occurring in the most popular textbooks, weighted by popularity. Assuming that early training influences later usage, we take this as a model of the set of mathematical expressions used by the population of North American engineers. We present a preliminary empirical analysis of the individual symbols and of sequences of n symbols (n-grams) occurring in these expressions.

Corpus Selection The first step in our approach was to identify the most popular textbooks in the area of second year engineering mathematics. US college and university bookstore sales for spring for 2006 to fall 2006 show the most demanded texts to be Kreyszig [2] (72%), Greenberg [3] (13%), O'Neil [4] (7%), Jeffrey (5%), Harman (2%). From this we see that three titles account for more than 90% of the textbook use. We therefore built our model based on these three titles.

TEX Sources For each of the three textbooks, we obtained TEX sources for all the mathematical expressions, and then constructed MathML from the TEX. For the texts by Greenberg and O'Neil, the author and publisher (respectively) were highly cooperative and provided the TEX sources directly. The sources for the text by O'Neil corresponded to the published version in use today. The sources for the text by Greenberg had somewhat diverged from the published text but not so much as to materially affect the analysis in our opinion. For the text by Kreyszig, the publisher and author declined to provide access to the source files. To obtain the mathematical expressions of the text in electronic form, we first scanned the entire book and used the Infty system [5] to produce TEX. In most cases the TEX produced had to be edited by hand to correct errors. This was a highly labour intensive activity that spanned several months. In the end we had a TEX representation for all the mathematical expressions in all three texts.

MathML Conversion Naïve examination of TEX sources does not give the mathematical expressions of a document. This is for two reasons: The first reason is that typical TEX document markup makes use of a number of macro packages, as well as author-defined macros. These macros have to be expanded to reveal the mathematical expression. The second reason that TEX sources do not give *expressions* directly is that the TEX representation of mathematics is not grouped as required. For example, most authors would write `$a + b c$` rather than `$a + {b c}$`. We used our TEX to MathML converter [6, 7] to expand the TEX macros and properly group the expressions. We then performed our analysis on the resulting MathML. The resulting expressions treated were (for the most part) complete, well formed, and grouped appropriately. We describe the conversion process in more detail elsewhere [1, 8].

Analysis We grouped the chapters of each text into general subject categories (ODEs, PDEs, vector calculus, *etc*) and analyzed the mathematical expressions for each subject/author combination, for each author with subjects combined (weighted by author emphasis), and for each subject with authors combined (weighted by sales volume). In each case, we computed the individual symbol frequencies (normalized to total 1) and n-gram frequencies for $n = 2, 3, 4, 5$. To compute the n-grams, we converted the expressions to strings by traversing the frontier of the expression trees in writing order. The resulting strings were over the alphabet of leaf symbols extended by `<sub>`, `</sub>`, `<sup>`, `</sup>`, `<frac/>` and `<root/>`. These symbols captured transitions from the expression baseline to subscripts and superscripts as well as built up fractions and radicals. The n-grams were then tallied using sliding windows over these strings.

Results Tables 1 and 2 show extracts of the preliminary results of our analysis. Table 1 shows the frequencies of the most commonly occurring symbols in the entire corpus. These are presented with the absolute symbol count for each author and as a percentage of all symbols, weighted by author. The relative weights used were (72, 13, 7). We see that the most popular symbols were common among all the authors, although the rank of the symbols varied somewhat from author to author. The total number of mathematical symbols occurring in the texts were 368,267, 467,044 and 391,602. Table 1 also shows the most commonly occurring symbols for two representative areas. We see that the declining relative frequency is similar between the areas, with a few outlying points (such as z being very popular for complex analysis). This same pattern was observed for all subject areas. The cumulative frequency of symbols is shown in Figure 1 with one curve for each subject and one for the weighted combination. From the log plot it is possible to see that the symbol frequencies follow an approximately exponential distribution.

Table 2 shows a preliminary count of the most popular 5-grams for the three corpus authors as well as from two comparison texts. The n-grams have a qualitatively similar declining frequency pattern as the symbols, but this time in a much larger space. The total number of n-grams (for each n) was 479,388 (Kreyszig), 562,297 (Greenberg) and 477,268 (O'Neil). The total number of *different* bigrams was 5,992 (Kreyszig), 7,056 (Greenberg) and 5,442 (O'Neil). The total number of *different* 5-grams was 140,306 (Kreyszig), 146,507 (Greenberg), 126,232 (O'Neil). Figure 1 shows the cumulative frequency for all distinct n-grams occurring in the text by Kreyszig. The highest curve is for $n = 2$ and they are in order to the lowest curve for $n = 5$. We find it remarkable that even though the ranking of the particular n-grams is different for the each author, the cumulative n-gram frequency curves are almost identical from author to author.

By analyzing the population of symbols and n-grams that occur in the corpus, we are able to determine the most popular symbols and n-grams by subject. The exponential drop in number of occurrences, from the highest ranked symbols and n-grams to the lowest, means that a compact database can contain most of the frequently occurring items. Thus applications, even those for portable devices, could use these statistics to guide their recognition.

Acknowledgments We thank Michael Greenberg, Peter O'Neil, Prentice-Hall and Thomson-Nelson for the use of their materials. We also thank Robert Lopez and Maplesoft for additional materials. We thank Jeliazko Polihronov for assistance in gathering the data and Elena Smirnova for work on the n-gram analysis software. This work was supported in part by grants from the NSERC Canada, Microsoft and Maplesoft.

References

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

[2] Erwin Kreyszig, *Advanced Engineering Mathematics, 8th edition*, John Wiley & Sons 1999.

[3] Michael Greenberg, *Advanced Engineering Mathematics, 2nd edition*, Prentice Hall 1998.

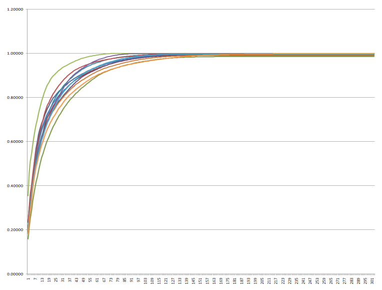
[4] Peter O'Neil, *Advanced Engineering Mathematics, 5th edition*, Thomson-Nelson 2003.

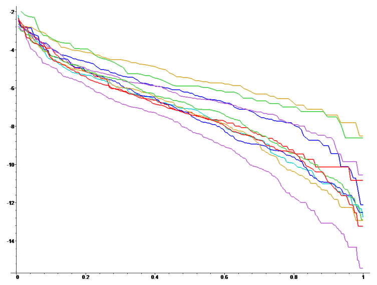
[5] M.Suzuki, F.Tamari, R.Fukuda, S.Uchida, T.Kanahori, *Infty—an Integrated OCR System for Mathematical Documents*, Proceedings of ACM Symposium on Document Engineering 2003, Grenoble, 2003, pp.95-104.

[6] ORCCA. *On-line TeX to MathML Translator.* `http://www.orcca.on.ca/MathML/texmml/textomml.html`

[7] S.M.Watt. *Implicit Mathematical Semantics in Conversion between TEX and MathML.* TUGBoat, **23**(1) 2002.

[8] E. Smirnova and S.M. Watt, *Context-Sensitive Mathematical Character Recognition.* International Conference on Frontiers in Handwriting Recognition (ICFHR 2008), (accepted).


Symbols
X: Symbol rank number
Y: Cumulative frequency


Symbols
X: Symbol rank number
Y: Log frequency

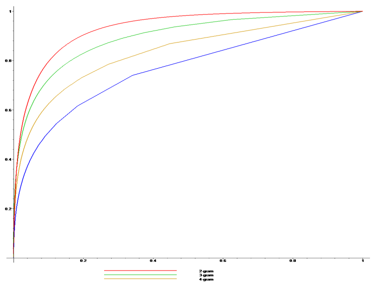

2-, 3-, 4- and 5-grams
X: $n-$gram rank number
Y: Cumulative frquencey

Figure 1: **Symbol and n-gram frequencies**

All areas combined				
Symbol	Weighted Freq. (%)	Symbol Counts Kreyszig	Greenberg	O'Neil
1	6.16415	24519	23209	20345
2	6.15918	24436	22613	21886
$=$	5.89883	22906	26202	19275
0	5.13055	20436	19623	16164
$($	5.08432	18162	26262	27777
$)$	5.08387	18158	26257	27804
x	4.97402	18271	28243	17918
$-$	3.82436	14609	15625	17152
$+$	3.12976	11906	14648	11711
y	2.94812	11400	13191	9996
$,$	2.53506	9796	12571	6784
n	2.11526	8016	9681	8577
z	1.88590	7447	7238	6593
3	1.87252	7225	7603	7706
$\frac{\square}{\square}$	1.73059	6386	7715	9163
t	1.71003	5771	9800	11446
$.$	1.62134	6234	4510	10083
4	1.42027	5694	4119	6097
f	1.30925	4926	6522	4874

Complex Analysis	
Symbol	Weighted Freq. (%)
z	11.28007
$=$	6.19577
$)$	5.76133
$($	5.75744
1	5.59297
2	5.21226
$-$	4.02399
0	3.88584
$+$	3.71409
i	2.95919
n	2.94910
$\vert$	2.78406
x	2.45995
f	1.98821
$,$	1.69837
y	1.60176
π	1.30730
C	1.18192
3	1.13346

PDEs	
Symbol	Weighted Freq. (%)
$=$	7.22187
x	7.04832
$($	6.44756
$)$	6.43967
2	5.54914
0	4.82981
u	4.28608
1	3.35806
n	3.33931
t	3.10607
y	2.63211
$-$	2.38819
$+$	2.02753
$,$	2.00038
c	1.68841
r	1.67920
π	1.66333
f	1.38602
∂	1.32067

Table 1: **Most popular symbols, by weighted frequency, for entire corpus and two sample areas**

Kreyszig		**Greenberg**		**O'Neil**		**Lopez**		**MSKit**	
Freq%	**Sequence**	**Freq%**	**Sequence**	**Freq%**	**Sequence**	**Freq%**	**Sequence**	**Freq%**	**Sequence**
0.00104	(x,y)	0.00142	$\int\langle\text{sub}\rangle 0\langle/\text{sub}\rangle\langle\text{sup}\rangle$	0.00152	(x,y)	0.00284	00000	0.00442	$lim\langle\text{sub}\rangle x$
0.00095	$y\langle\text{sup}\rangle {''}\langle/\text{sup}\rangle$	0.00130	(x,y)	0.00149	$\int\langle\text{sub}\rangle 0\langle/\text{sub}\rangle\langle\text{sup}\rangle$	0.00136	(x,y)	0.00406	$im\langle\text{sub}\rangle x \rightarrow$
0.00082	$x\langle\text{sub}\rangle 1\langle/\text{sub}\rangle +$	0.00077	$0\langle/\text{sub}\rangle\langle\text{sup}\rangle\infty\langle/\text{sup}\rangle$	0.00106	$y\langle\text{sup}\rangle {''}\langle/\text{sup}\rangle$	0.00120	$x\langle\text{sup}\rangle 2\langle/\text{sup}\rangle +$	0.00320	$x\langle\text{sup}\rangle 2\langle/\text{sup}\rangle +$
0.00081	$f(x) =$	0.00077	$x\langle\text{sup}\rangle 2\langle/\text{sup}\rangle +$	0.00102	$0\langle/\text{sub}\rangle\langle\text{sup}\rangle\infty\langle/\text{sup}\rangle$	0.00104	$\int\langle\text{sub}\rangle 0\langle/\text{sub}\rangle\langle\text{sup}\rangle$	0.00285	$dy\langle\text{frac/}\rangle dx$
0.00080	$\int\langle\text{sub}\rangle 0\langle/\text{sub}\rangle\langle\text{sup}\rangle$	0.00071	(x,t)	0.00100	$\sum\langle\text{sub}\rangle n = 1$	0.00096	$f(x) =$	0.00200	$f(x) =$
0.00073	$0\langle/\text{sub}\rangle\langle\text{sup}\rangle\infty\langle/\text{sup}\rangle$	0.00070	$1\langle/\text{sub}\rangle,\ldots,$	0.00100	$\langle\text{sub}\rangle n = 1\langle/\text{sub}\rangle$	0.00074	$x\langle\text{sup}\rangle 2\langle/\text{sup}\rangle -$	0.00200	$sin(x$
0.00072	$\langle/\text{sub}\rangle\langle\text{sup}\rangle {'}\langle/\text{sup}\rangle =$	0.00067	$\langle\text{sub}\rangle 1\langle/\text{sub}\rangle,\ldots$	0.00100	$n = 1\langle/\text{sub}\rangle\langle\text{sup}\rangle$	0.00070	$(x,y,$	0.00190	$x\langle\text{sup}\rangle 2\langle/\text{sup}\rangle -$
0.00072	$x\langle\text{sup}\rangle 2\langle/\text{sup}\rangle +$	0.00062	$y(x) =$	0.00094	$1\langle/\text{sub}\rangle\langle\text{sup}\rangle\infty\langle/\text{sup}\rangle$	0.00066	$\langle\text{sup}\rangle 2\langle/\text{sup}\rangle + 1$	0.00188	$in(x)$
0.00071	$\langle\text{sup}\rangle {''}\langle/\text{sup}\rangle +$	0.00060	$\langle\text{sub}\rangle 0\langle/\text{sub}\rangle\langle\text{sup}\rangle\infty$	0.00093	$= 1\langle/\text{sub}\rangle\langle\text{sup}\rangle\infty$	0.00065	$,\ldots,$	0.00154	$2x\langle\text{sup}\rangle 2\langle/\text{sup}\rangle$
0.00064	$-z\langle\text{sub}\rangle 0\langle/\text{sub}\rangle$	0.00057	$\langle/\text{sub}\rangle(x) =$	0.00090	$)sin($	0.00064	$f(x,y$	0.00141	$cos(x$
0.00060	$.....$	0.00056	$(0) = 0$	0.00086	$x\langle\text{sup}\rangle 2\langle/\text{sup}\rangle +$	0.00064	$+y\langle\text{sup}\rangle 2\langle/\text{sup}\rangle$	0.00133	$os(x)$
0.00057	$\langle\text{sub}\rangle 1\langle/\text{sub}\rangle\langle\text{sup}\rangle {'}$	0.00056	$f(x,y$	0.00084	$\langle\text{sup}\rangle {''}\langle/\text{sup}\rangle +$	0.00061	x,y,z	0.00131	$x\langle\text{sup}\rangle 3\langle/\text{sup}\rangle +$
0.00055	$z\langle\text{sub}\rangle 0\langle/\text{sub}\rangle)$	0.00052	$\langle\text{sub}\rangle 1\langle/\text{sub}\rangle(x$	0.00084	$(-1)\langle\text{sup}\rangle$	0.00060	$,y,z)$	0.00124	$\langle\text{sup}\rangle 2\langle/\text{sup}\rangle + 1$
0.00055	$y\langle\text{sub}\rangle 1\langle/\text{sub}\rangle\langle\text{sup}\rangle$	0.00052	$1\langle/\text{sub}\rangle(x)$	0.00082	$sin(n$	0.00059	$2x\langle\text{sup}\rangle 2\langle/\text{sup}\rangle$	0.00122	$log\langle\text{sub}\rangle a$
0.00054	$y(0) =$	0.00052	$1\langle/\text{sub}\rangle\langle\text{sup}\rangle\infty\langle/\text{sup}\rangle$	0.00076	$y\langle\text{sup}\rangle {'}\langle/\text{sup}\rangle =$	0.00058	$\langle\text{sup}\rangle 2\langle/\text{sup}\rangle + y$	0.00122	$og\langle\text{sub}\rangle a\langle/\text{sub}\rangle$
0.00054	$1\langle/\text{sub}\rangle\langle\text{sup}\rangle {'}\langle/\text{sup}\rangle$	0.00051	x,y,z	0.00073	$y\langle\text{sup}\rangle {'}\langle/\text{sup}\rangle +$	0.00058	$)sin($	0.00117	$y\langle\text{frac/}\rangle dx =$
0.00051	$\langle\text{sub}\rangle 2\langle/\text{sub}\rangle\langle\text{sup}\rangle {'}$	0.00050	$f(x) =$	0.00072	$,\ldots,$	0.00058	$\langle/\text{sup}\rangle cos($	0.00116	$\langle\text{root/}\rangle 2x\langle\text{sup}\rangle 2$
0.00051	$z - z\langle\text{sub}\rangle 0$	0.00049	$(x,y,$	0.00070	(x,t)	0.00057	$\langle/\text{sup}\rangle sin($	0.00116	$(x\langle\text{sup}\rangle 2\langle/\text{sup}\rangle$
0.00050	$,y\langle\text{sub}\rangle 2\langle/\text{sub}\rangle$	0.00044	$\langle\text{sub}\rangle n\langle/\text{sub}\rangle(x$	0.00068	$-1)\langle\text{sup}\rangle n$	0.00055	$0\langle/\text{sub}\rangle\langle\text{sup}\rangle\infty\langle/\text{sup}\rangle$	0.00116	$du\langle\text{frac/}\rangle dx$

Table 2: **Most popular 5-grams**

Triangular Decompositions for Solving Parametric Polynomial Systems

Changbo Chen[1], Marc Moreno Maza[1], Bican Xia[2] and Lu Yang[3]
[1]: University of Western Ontario, London, Ontario, Canada
[2]: Peking University, Beijing, China
[3]: East China Normal University, Shangai, China

Triangular decompositions, like lexicographical Gröbner bases, are natural candidates for studying parametric polynomial systems. However, these tools need to be equipped with additional concepts and algorithms in order to answer the usual questions arising with these systems. In many applications, see for instance [2, 9] one wants to determine the number of complex or real roots depending on the parameters. Thus, algorithms for solving parametric systems need to take into account the fact that two different groups of variables are involved: the *unknowns* and the *parameters*. Comprehensive Gröbner bases [10, 11, 6, 5] and the use of block term ordering in Gröbner basis calculations, as in [4], are techniques to meet this requirement.

For triangular decompositions, several approaches have been proposed: the use of *regular systems* and *simple systems* in [8, 7], *decomposition trees* and *refined covers* in [3], *border polynomials* in [12] and *comprehensive triangular decomposition (CTD)* [1]. In all these works, except [12], the authors study parametric constructible sets whereas in [12] parametric semi-algebraic sets are the object of study. Another distinction between theses works is the following. In [8, 7, 3, 1] the goal is to provide a representation of the unknowns as functions of the parameters; this representation is a triangular decomposition in the case of [8, 7, 3] and a family of triangular decompositions (indexed by a partition of the parameter space) in [1].

In [12], the emphasis is on determining necessary and sufficient conditions for the input parametric semi-algebraic system to have a prescribed number of real solutions. Moreover the computation of these conditions is incremental: one obtains first the conditions on the parameters corresponding to components of maximum dimension. In practice, this incremental approach can provide information on the input system whereas a non-incremental approach could be stuck in some huge intermediate calculation. The algorithm presented is freely available in the form of a MAPLE library, called `Discoverer`.

The CTD offers several attractive features. First, it relies on concepts, such as the *discriminant constructible set of the input system*, which are independent of the algorithms that compute them. Moreover, the experimental results reported in [1] show that the CTD code outperforms the performances of software solvers with comparable specification and implementation environment, namely MAPLE. However, the notion of a CTD, as introduced in [1] was limited to a system of parametric equations (without equations or inequalities). Based on these observations, the contributions of the present article naturally extend the work of [12] and [1].

Our first contribution is a notion of CTD for a parametric constructible set, together with an algorithm for computing it. A first by-product is an algorithm for complex root counting, depending on parameters. A second by-product of this extension is the fact that we can compute the image (or the pre-image) of a constructible set by a rational map. These applications, and others, are reported in a forthcoming paper. Examples of a CTD, together with a decomposition computed by `Discoverer`, are included.

Our second contribution, is a notion of CTD for a parametric semi-algebraic set and, here again, we provide an algorithm for computing it. In broad terms, the CTD of a basic parametric semi-algebraic set $\mathcal{S}$ is a "refined triangular decomposition" followed by a so-called "connected semi-algebraic decomposition". The motivation of our design is to avoid *cylindrical algebraic decomposition* (CAD) and instead rely only on *partial CAD*. Under this constraint and standard hypotheses on the input $\mathcal{S}$, we provide an algorithmic solution to the following real root counting problem: given a positive integer number n, describe the set of parameter values for which $\mathcal{S}$ has n distinct real points. While borrowing some ideas from `Discoverer`, our strategy is fairly different from it. Experimental comparison between the two approaches is work in progress and will be reported in another article.

A third contribution of this paper is a comparison between different notions used for parametric polynomial system solving. More precisely, the poster will show some relations between the notions of border polynomial [12], discriminant set [1] and minimal discriminant variety [4]. In particular, we show that for a parametric basic constructible set CS, the discriminant set of CS is contained in its minimal discriminant variety. Moreover, we show that, for a parametric regular system R, the zero set of the border polynomial is the minimal discriminant variety of the zero set of R.

References

[1] C. Chen, O. Golubitsky, F. Lemaire, M. Moreno Maza, and W. Pan. *Comprehensive Triangular Decomposition*, volume 4770 of *Lecture Notes in Computer Science*, pages 73–101. Springer Verlag, 2007.

[2] F. Chen and D. Wang, editors. *Geometric Computation*. Number 11 in Lecture Notes Series on Computing. World Scientific Publishing Co., Singapore, New Jersey, 2004.

[3] X.S. Gao and D.K. Wang. Zero decomposition theorems for counting the number of solutions for parametric equation systems. In *Proc. ASCM 2003*, pages 129–144, World Scientific, 2003.

[4] D. Lazard and F. Rouillier. Solving parametric polynomial systems. *J. Symb. Comput.*, 42(6):636–667, 2007.

[5] M. Manubens and A .Montes. Improving dispgb algorithm using the discriminant ideal, 2006.

[6] A .Montes. A new algorithm for discussing gröbner bases with parameters. *J. Symb. Comput.*, 33(2):183–208, 2002.

[7] D. Wang. Computing triangular systems and regular systems. *Journal of Symbolic Computation*, 30(2):221–236, 2000.

[8] D. M. Wang. Decomposing polynomial systems into simple systems. *J. Symb. Comp.*, 25(3):295–314, 1998.

[9] D.M. Wang and B. Xia. Stability analysis of biological systems with real solution classification. In *Proc. 2005 International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 354–361, New York, 2005. ACM Press.

[10] V. Weispfenning. Comprehensive Gröbner bases. *J. Symb. Comp.*, 14:1–29, 1992.

[11] V. Weispfenning. Canonical comprehensive Gröbner bases. In *ISSAC 2002*, pages 270–276. ACM Press, 2002.

[12] L. Yang, X. Hou, and B. Xia. A complete algorithm for automated discovering of a class of inequality-type theorems. *Science in China, Series* **F**, 44(6):33–49, 2001.

Milestones in Computer Algebra
MICA 2008

A Conference in Honour of Keith Geddes' 60th Birthday

1948 1968 1988 2008

Stonehaven Bay, Trinidad and Tobago, 1 - 3 May 2008

Keith Geddes received his PhD in 1973 from the University of Toronto under the supervision of John C. Mason. Since that time, as a professor at the University of Waterloo, his research has spanned the areas of numerical approximation, algebraic algorithms for symbolic computation, hybrid symbolic-numeric computation and the design and implementation of computer algebra systems. He is perhaps best known as co-founder of the Maple computer algebra system. Through his teaching, research and software, the work of Keith Geddes has touched millions of individuals.

Program Committee

Jacques Carette *(McMaster U., Canada)*
Robert Corless *(U. Western Ontario, Canada)*
James Davenport *(U. Bath, UK)*
Jürgen Gerhard *(Maplesoft, Canada)*
Mark Giesbrecht, Chair *(U. Waterloo, Canada)*
Laureano Gonzalez Vega *(U. Cantabria, Spain)*
Hoon Hong *(North Carolina State U., USA)*
Erich Kaltofen *(North Carolina State U., USA)*
George Labahn *(U. Waterloo, Canada)*
Ziming Li *(AMSS Academia Sinica, China)*

Invited Speakers

Jim Cooper *(Maplesoft, Canada)*
Gaston Gonnet *(ETH Zurich, Switzerland)*
Michael Monagan *(Simon Fraser U., Canada)*
Joel Moses *(MIT, USA)*
Peter Paule *(RISC Linz, Austria)*
B. David Saunders *(U. Delaware, USA)*
David Stoutemyer *(U. Hawaii, USA)*
Jan Verschelde *(U. Illinois at Chicago, USA)*

Organizing Committee

General Chair:
Stephen Watt *(U. Western Ontario, Canada)*
Program Committee Chair:
Mark Giesbrecht *(U. Waterloo, Canada)*
Sponsorship Chair:
George Labahn *(U. Waterloo, Canada)*
Publicity Chair:
Ilias Kotsireas *(Wilfrid Laurier U., Canada)*
Treasurer:
Jim Cooper *(Maplesoft, Canada)*
Local Arrangements:
Bal Swaroop Bhatt *(U. West Indies, Trinidad and Tobago)*
David Jeffrey *(U. Western Ontario, Canada)*
Electronic Proceedings:
Marc Moreno Maza, Stephen Watt *(U. Western Ontario, Canada)*

http://www.orcca.on.ca/conferences/mica2008

acm Maplesoft command the brilliance ORCCA University of Waterloo Western UWI