Message from the SIGSAM Chair

Ilias S. Kotsireas
Wilfrid Laurier University
Waterloo, ON, Canada

Dear SIGSAM Members,

It is a great honor to serve as Chair of ACM SIGSAM, a historical organization with numerous and long-lasting contributions to Computer Algebra and Symbolic Computation. I would like to take this opportunity to thank all candidates at the recent elections. Everyone is a valuable member of the community and I would encourage them to take part in the next election as well and/or to continue to be actively involved in SIGSAM activities and projects. SIGSAM is a volunteer-based organization and therefore it is important to involve as many volunteers as possible. I would also like to welcome the three elected members of the new SIGSAM executive, namely:

- Vice Chair: Jean-Guillaume Dumas (Université Joseph Fourier, France), `VC_SIGSAM@acm.org`
- Secretary: Ziming Li (Institute of Systems Science, China), `Secretary_SIGSAM@acm.org`
- Treasurer: Agnes Szanto (North Carolina State University, USA), `Treasurer_SIGSAM@acm.org`

I am looking forward to working with all three of them. I am also looking forward to work with those of you that would be willing to take some time off your busy schedules to work on some projects that I plan to pursue. To be more specific, I plan to intensify work on the following projects:

- Update and re-organize the SIGSAM webpage `http://www.sigsam.org/`
- Design and implement a campaign to increase SIGSAM membership. This will include reviewing all benefits of SIGSAM membership and discussing ways to increase the value of membership for all of those interested in computer algebra and related fields
- Enlarge and maintain coverage of CCA in on-line publication databases, i.e. DBLP, MathSciNet

The annual SIGSAM business meeting was held during the ISSAC 2013 conference in Boston and it was reported that:

1. SIGSAM is financially healthy with a balance of more than $\$60,000$.
2. SIGSAM sponsorship of ISSAC paper prizes and Jenks award funded by endowed accounts which also remain healthy
3. Item 1 enabled SIGSAM to reduce the fee from ACM for ISSAC sponsorship from 16% to 10% which was the same as when ISSAC was sponsored by INRIA at Grenoble.
4. ACM has revised its publishing policies. This includes authorizer which allows authors to post official copies of their ACM publications. It also allows ACM published proceedings to be available to everyone for one year. I.E. we can post the table of contents of ISSAC proceedings with links to all papers for one year on the ISSAC or SIGSAM website.
5. Past SIGSAM Chair, Prof. Jeremy Johnson, Drexel University, presented the SIGSAM Program Review on March 14, 2013. SIGSAM received renewed viability from ACM SGB for another 3 years. The official ACM statement is:
   SIGSAM
   The SGB EC congratulates SIGSAM on their continuing importance to the community, but has

concerns about submissions and attendance at the ISSAC conference and finds SIGSAM viable to continue its status for the next 3 years.

The preparations for the ISSAC 2014 conference are well underway, under the leadership of general co-chairs Kosaku Nagasaka (Kobe) and Franz Winkler (RISC-Linz). The conference will be held July 23–25 in Kobe, Japan. Please consult the website `http://www.issac-symposium.org/2014/` for more information. ISSAC 2014 is in cooperation with ACM. It is worthwhile to note that ISSAC 2014 is a satellite conference of the International Congress of Mathematicians (ICM 2014) that will be held August 13–21 in Seoul, Korea, `http://www.icm2014.org/` .

The venue for the ISSAC 2015 conference was selected during the business meeting at ISSAC 2013 in Boston. There were four candidate cities and the winning city was Bath, UK. The representative of one of the four candidate cities expressed concerns over the preparation and the logistics of the bidding presentations. I witnessed some aspects of the entire process first hand and I believe the concerns were justified and valid. In order to address these concerns, I believe that SIGSAM and the ISSAC steering committee should investigate the entire bidding process and possibly offer recommendations on how to systematize it and streamline it, so as to hopefully avoid this type of concerns at future bids for ISSAC conferences.

SIGSAM sponsored the following ISSAC 2013 awards:

- Distinguished Student Author Awards: Pierre Lairez for Creative Telescoping for Rational Functions Using the Griffiths-Dwork Method (with Alin Bostan and Bruno Salvy) and Qingdong Guo for Computing Rational Solutions of Linear Matrix Inequalities (with Mohab Safey El Din and Lihong Zhi).
- Distinguished Paper Award: Jingguo Bi, Qi Cheng, and J. Maurice Rojas. Sub-Linear Root Detection and New Hardness Results for Sparse Polynomials Over Finite Fields.
- Distinguished Poster Awards: Jeremy Johnson and Lingchuan Meng. Towards Parallel General-Size Library Generation for Polynomial Multiplication. James Wan. Hypergeometric Generating Functions and Series for $\frac{1}{\pi}$.
- Distinguished Software Presentation Award: Fredrik Johansson. Arb: a C Library for Ball Arithmetic.
- The 2013 Richard Dimick Jenks Memorial Prize for Excellence in Software Engineering applied to Computer Algebra was awarded to the William Stein for the Sage Project.

I would like to thank CCA editor Manuel Kauers (RISC-Linz, Austria) and Information Director William J. Turner (Wabash College, USA) along with the editorial board for making sure that the four yearly issues of CCA are published in a timely manner on-line and in the printed version. Timely publication of CCA is an important aspect of the periodic viability review for SIGSAM. I would also like to thank Clément Pernet (Université Jopseph Fourier, Grenoble, France) for serving as the SIGSAM representative on the Editorial Board of ACM Transactions on Mathematical Software (TOMS) `http://toms.acm.org/` .

Finally I would like to thank the extremely efficient ACM staff for their help and support during my past ten years as Editor of CCA and on other occasions such as conference organization. Special thanks go to Irene Frawley, Program Coordinator, SIG Activities, for her enthusiasm and dedication. I hope to continue to work with all of them and be productive in my new role as SIGSAM Chair.

Ilias S. Kotsireas
SIGSAM Chair `chair_SIGSAM@acm.org`
Wilfrid Laurier University
Waterloo, ON, Canada

ACM Communications in Computer Algebra, Vol. 47, No. 3, Issue 185, September 2013

Asymptotic series of Generalized Lambert W Function

Tony C. Scott[1,2], Greg Fee[3] and Johannes Grotendorst[4]

[1] Institut für Physikalische Chemie, RWTH Aachen University, 52056 Aachen, Germany
email: scott@pc.rwth-aachen.de

[2] Zephyr Health Inc, 589 Howard Street, 3rd floor, San Francisco CA 94105, USA
email: tony@zephyrhealthinc.com

[3] Centre for Experimental and Constructive Mathematics (CECM), Simon-Fraser University, Burnaby, BC V5A 1S6 Canada
email: gjfee@cecm.ca

[4] Institute for Advanced Simulation, Jülich Supercomputing Centre, Forschungszentrum Jülich, 52425 Jülich, Germany
email: j.grotendorst@fz-juelich.de

Abstract

Herein, we present a sequel to earlier work on a generalization of the Lambert W function. In particular, we examine series expansions of the generalized version providing computational means for evaluating this function in various regimes and further confirming the notion that this generalization is a natural extension of the standard Lambert W function.

AMS Numbers: 33E30, 01-01, 01-02
Also related to: 70B05, 81Q05, 83C47, 11A99

1 Introduction

The Lambert W function satisfying $W(t)e^{W(t)} = t$ provides an exact solution to:

$$e^{-cx} = a_o\ (x - r_1) \tag{1}$$

with $x = r_1 + \frac{1}{c}W(c\ e^{-cr_1}/a_o)$. The Lambert W function appears in a myriad number of applications. In particular, it appears in the "lineal" gravity two-body problem [1, 2] as a solution to the Einstein Field equations in $(1 + 1)$ dimensions. The Lambert W function appears as a solution for the case when the two-bodies have exactly the same mass. However, the case of unequal masses required a *Generalization* of Lambert's function [1, eq.(81)].

$$e^{-cx} = a_o\ (x - r_1)(x - r_2) \tag{2}$$

This generalization originally appeared from the (quantum-mechanical) Double Well Dirac Delta Potential model [3], a one-dimensional version of a special case of the quantum-mechanical three-body system known

as the *Hydrogen Molecular Ion* (and also appears in quantum gravity [2]). For this problem, specifically $r_1 = 1$, $r_2 = \lambda$, $c = 2\ R$ where R is the internuclear distance. $a_o = \frac{1}{\lambda}$ and λ was treated formally as real perturbative parameter (the case at $\lambda = 1$ allows eq. (2) to factor into (1) which is solvable in terms of the standard Lambert W function). In its original form, this equation was written in a more complicated form, namely a *pseudo-quadratic*: with two solutions for x [3–6]:

$$x_\pm(\lambda) \ = \ \tfrac{1}{2}(\lambda+1) \pm \tfrac{1}{2}\left\{(1+\lambda)^2 - 4\,\lambda[1 - e^{-c x_\pm(\lambda)}]\right\}^{1/2}$$

where $E_\pm = -x_\pm/2$ are the quantum state energies (for respectively the two distinct solutions $x_\pm$). All these quantities including the energies were real though we do not rule out a generalization to the complex plane.

A difficulty encountered by Byers-Brown and Scott *et al.* is that Physical Chemists followed a conventional practice of starting with the case $\lambda = 0$ whose solution is $x_0 = 1$ as a starting point and considering a series expansion about x_0 of eq. (1) in powers of λ. This was called the "polarization expansion" for the range $0 < \lambda < 1$ and proves very difficult to sum, necessitating the use of Padé-Hermite Approximants [3]. This slow convergence became aggravated for larger but similar molecular systems like the Hydrogen Molecular Ion requiring much discussion (and calculation) to sort out the convergence of the eigenstates and related quantities once and for all [7, 8].

Subsequently, it was realized that eq. (2) could be further generalized to the case of a rational polynomial [9]:

$$e^{-c\,x} \ = \ \frac{P_N(x)}{Q_M(x)} \tag{3}$$

where $c > 0$ is a constant as before and $P_N(x)$ and $Q_M(x)$ are polynomials in x of respectively orders N and M. Eq. (3) expresses the solution for the energy eigenvalues of the three-dimensional (and realistic) version of the Hydrogen molecular ion. These generalizations were found to express solutions to a huge class of fundamental problems and were found to be natural extensions of the standard W function requiring merely a formal nesting of the standard Lambert W function [10] and thus economical conceptually in terms of mathematical resources. Some exact solutions were even found for some special cases for eq. (2) [10].

Herein, we examine the more pragmatic matter of obtaining series expansions for eq. (2) for analytical and computational purposes. In the process, we will show how closely they relate to the series expansions of the standard W function. We will examine three series expansions which apply to three different regimes. Though eq. (2) is not the full generalization in eq (3) it already embodies a link between gravity theory and quantum mechanics albeit in lower dimensions [2] and is therefore instructive as a special case beyond the standard W function. Finally, some concluding remarks are made at the end. Since we are dealing with applications in Physics, the input parameters c, a_o and the polynomial roots r_i where $i = 1, 2, \ldots$ are assumed to be real.

2 Series Expansions

2.1 Taylor series in r_d

By a series of manipulations, eq. (1) can be brought in the familiar standard form:

$$x_0 \ = \ W(x_0)e^{W(x_0)} \quad \text{where} \quad x_0 \ = \ \frac{c\ e^{-c r_1}}{a_o} \tag{4}$$

Using very similar manipulations and defining respectively the mean and difference of the roots r_1 and r_2:

$$r_m \;=\; \frac{r_1 + r_2}{2} \quad \text{and} \quad r_d \;=\; \frac{r_1 - r_2}{2} \;, \tag{5}$$

and by *completing the square* for the quadratic on the right of eq. (2):

$$(x - r_1)(x - r_2) \;=\; (x - r_m)^2 - r_d^2$$

and defining $W(r_d) = x - r_m$, eq. (2) can be rewritten as:

$$e^{-c(W(r_d)+r_m)} \;+\; a_o\, r_d^2 \;=\; a_o\, W(r_d)^2 \;. \tag{6}$$

The above can be viewed as the intersection between an exponential of the form $Ae^{-c\,x}$ and a "simple harmonic oscillator" of the form Bx^2. Potentially, there can be two and as much as three intersections (in the real plane), in some cases, roots of the same sign. To obtain real solutions, we constrain $a_o > 0$. It is very similar to eq. (1) the equation governing the standard Lambert W function with the mean of the roots r_m playing the role of the r_1 in the monomial on the right side of eq. (1), the difference in the roots r_d representing a departure from the form of eq. (1). This makes perfect since because when $r_d = 0$, then $r_1 = r_2$ and eq. (2) can be factored into the form of eq. (1) bringing us back to the standard W function. We define:

$$z_0 = \frac{1}{2}\sqrt{\frac{c^2}{a_o}}e^{-cr_m/2} = \frac{1}{2}\frac{c}{\sqrt{a_o}}e^{-cr_m/2} \tag{7}$$

where it is understood that $W(0)$ is/are the solution(s) when $r_d = 0$:

$$W(0) \;=\; \frac{2}{c}\; W(\pm z_0) \;=\; \frac{2}{c}\; W_0 \tag{8}$$

and where $W(\pm z_0)$ on the right side of eq. (8) is the *standard* Lambert W function. For real results, in particular for the parameters mentioned for the Double Well Dirac potential mentioned just below eq. (2), we are interested in real results and make use of the main branch of the standard W function. In this case, $c > 0$ helps ensure $|z_0| < 1/e$ (although $W(-z_0)$ could have a real result on a different branch for c sufficiently small). Implicit differentiation on both sides of eq. (6) yields:

$$\frac{\partial W(r_d)}{\partial r_d} \;=\; \frac{2r_d}{\frac{ce^{-c(W(r_d)+r_m)}}{a_o} + 2W(r_d)} \;=\; \frac{2r_d}{c\; W(r_d)^2 - c\; r_d^2 + 2\; W(r_d)} \tag{9}$$

Naturally successive derivatives with respect to r_d yields the Taylor series in r_d. Its radius of convergence will be obtained from the disk about the point of expansion $r_d = 0$ (assuming it is regular at the point of expansion) bounded by the closest singularity or branch point in the complex plane namely when the denominator of this derivative and all successive derivatives is zero, with $W(r_d)$ simultaneously satisfying eq. (6). Note that the expression on the right most side of eq. (9), obtained by virtue of eq. (6), does not formally depend on a_o nor r_m but only on c and r_d. Even though this is a quadratic in $W(r_d)$, only one solution satisfies eq. (6), namely:

$$W(r_{d\;crit}) \;=\; \frac{-1 + \sqrt{1 + c^2\; r_{d\;crit}^{\;2}}}{c} \tag{10}$$

The critical radius in the complex plane is:

$$r_{d\;crit} \;=\; \pm\frac{1}{c}\sqrt{2\; W(-2\; z_0^2) + W(-2\; z_0^2)^2} \;. \tag{11}$$

Here $W_0 = W(\pm z_0)$ is the standard W function and the radius is $|r_{d\ crit}|$. Note that when $z_0 = 0$, $W(z_0) = W(-2z_0^2) = 0$ (on the main branch) and the radius of convergence is also zero even though $z_0 = 0$ is analytic on the main branch for the (standard) Lambert W function. The series in r_d is thus:

$$\begin{aligned} W(r_d) &= 2\,\frac{W_0}{c} + \frac{1}{4}\,\frac{c\,r_d^2}{W_0(W_0+1)} + \frac{1}{64}\,\frac{c^3\,r_d^4\,(2\,W_0^2-1)}{W_0^3(W_0+1)^3} \\ &+ \frac{1}{1536}\,\frac{c^5\,r_d^6\,(8W_0^4-12W_0^2+3-4W_0^3)}{W_0^5(W_0+1)^5} \\ &+ \frac{1}{49152}\,\frac{c^7\,r_d^8\,(48W_0^6-132W_0^4+90W_0^2-15-64W_0^5+40W_0^3)}{W_0^7(W_0+1)^7} + O(c^9 r_d^{10}) \end{aligned} \tag{12}$$

which is a series in r_d^2 for $x = W(r_d) + r_m$ with x governed by eq. (2) and the radius of convergence is provided by the magnitude of (11). Within its radius of convergence, it converges rapidly. Note that when argument of z_0 is such that $W_0 = 0$ (which happens when e.g. $z_0 = 0$ on the main branch) or $W_0 + 1 = 0$ (which happens when $z_0 = -e^{-1}$ which is a branch point on the main branch), the individual series coefficients are confronted with divisions by zero, a result consistent, for the case $W_0 = 0$, with a radius of convergence of zero as given by eq.(11).

The validity of this series is demonstrated with some numerical tests. To reiterate the earlier problem, for a relatively high value of $\lambda = 0.8$ and an internuclear distance near the bond length $R = 2$, we have:

$$a_o = \frac{5}{4}, \quad c = 4, \quad r_d = \frac{1}{10}, \quad r_m = \frac{9}{10}$$

The solution of eq. (2) is $x = 1.0485$ obtained to within 4 decimals using the series in eq. (12) to within and including order $O(r_d^{10})$ using $W_0 = W(z_0)$ as the lead term. Similarly, the other solution $x = 0.6248$ is obtained using $W_0 = W(-z_0)$ as the lead term. The convergence of this series is much more rapid than the original "polarization expansion" mentioned in the introduction. Furthermore, this series is not limited to the real plane. For $\lambda = \frac{9}{10} - \frac{1}{10}i$

$$a_o = \frac{45}{41} + \frac{5}{41}i, \quad c = 4, \quad r_d = \frac{1}{20} + \frac{1}{20}i, \quad r_m = \frac{19}{20} - \frac{1}{20}i$$

The series to (and including) order $O(r_d^{14})$ yields $x = 1.0651408 - 0.0281742\,i$ to within 7 decimals for $W_0 = W(z_0)$ and similarly $x = 0.72818558 - 0.0876039\,i$ for $W_0 = W(-z_0)$ This series expansion is valid for small differences in the roots r_d, so clearly an asymptotic expansion valid for large r_d is also needed.

It would seem that in the case of three real roots, that we would only recover at most two out of three solutions. However, when two roots appear for e.g. $x > 0$ and the third root appears for $x < 0$, the latter can be recovered by reflection symmetry on the parameters. Let $x \to -x$, $c \to -c$, $r_i \to -r_i$ and these same formula can be used to recover that third solution.

2.2 Reversion of Power Series

To get an asymptotic series valid for large r_d, we further transform eq. (6) with the following variable transformations:

$$\begin{aligned} W(r_d)^2 &= \left(\frac{2}{c}\right)^2 (U^2 + d^2) \\ d &= c\,r_d/2 \end{aligned} \tag{13}$$

and $x = W(r_d) + r_m$ as before. Following the procedure for the standard W function [11], we start from:

$$z_0 \;=\; f(U) = Ue^{\pm\sqrt{U^2+d^2}} \quad \text{where} \quad z_0 = \frac{1}{2}\frac{c\; e^{-cr_m/2}}{\sqrt{a_o}} \tag{14}$$

where the sign $\pm$ takes into account that the negative square root is also possible. When $d = 0$, eq. (14) reduces to the form of the standard W function. Eq. (14) has the form:

$$z = f(U)$$

and we seek to reverse the power series to obtain:

$$U = g(z);$$

Defining $\phi(U) = U/f(U) = e^{\mp\sqrt{U^2+d^2}}$ and noting that $\phi(0) \neq 0$, we use a specialized version of the Lagrange-Bürmann [12] formula:

$$U(z) \;=\; z\phi(0) \;+\; \sum_{k=1}^{\infty} \frac{z^{k+1}}{(k+1)!}\frac{\partial^k \phi(U)^{k+1}}{\partial U^k}\Big]_{U=0} \tag{15}$$

Implicit differentiation of eq. (14) w.r.t. z yields:

$$\frac{\partial U(z)}{\partial z} \;=\; \frac{\sqrt{U(z)^2+d^2}\; e^{\mp\sqrt{U(z)^2+d^2}}}{U(z)^2+\sqrt{U(z)^2+d^2}} \tag{16}$$

We can see that the square root term dominates the functional form of the derivatives and the branch structure $U(z)$ in the complex plane much in accordance with the findings of Byers-Brown [5,6]. Note that eq. (16) has no explicit dependence on z and thus there is no need to verify its consistency with (14). To get the radius of convergence, we need to consider both the branch structure of the square root term in the denominator of (16) and values of $U(z)$ in the complex plane about the region $z = 0$ which would make this denominator zero. Thus the radius of convergence is limited by either:

$$|U_{crit}| < |d|$$

or

$$|U_{crit}| < \frac{1}{2}|\sqrt{2 \pm 2\sqrt{1+4d^2}}| \tag{17}$$

whichever is smaller. We obtain:

$$U(z) \;=\; ze^{\mp d} \;\mp\; \frac{1}{2}\frac{z^3 e^{\mp 3d}}{d} \;\pm\; \frac{1}{8}\frac{z^5(\pm 5d+1)e^{\mp 5d}}{d^3} \;+\; O(z^7 e^{\pm 7d}) \tag{18}$$

$$\;=\; \frac{1}{2}\frac{c\; e^{-\frac{1}{2}cr_\pm}}{\sqrt{a_o}} \;\mp\; \frac{1}{8}\frac{c^2\; e^{-\frac{3}{2}cr_\pm}}{a_o^{3/2} r_d} \;\pm\; \frac{1}{64}\frac{c^2\; e^{-\frac{5}{2}cr_\pm}(\pm 5\, c\, r_d + 2)}{a_o^{5/2} r_d^3} \;+\; O(e^{-\frac{7}{2}r_\pm c}) \tag{19}$$

where $r_+ = r_1$, $r_- = r_2$, $x = r_m + (2/c)\sqrt{U^2+d^2}$ as x given in eq. (2). This series would have growing exponential terms of the form $\exp(-k * d)$ unless $k > 0$ and consequenly this necessitates the requirement that $c\; r_\pm > 0$. Thus, we obtain a valid asymptotic expansion valid for large d or equivalently large r_d.

As in the previous section, we also get two kinds of solutions, respectively for positive and negative d, but they do not necessarily relate at all to the solutions of the previous section. The first section involved a

series expansion in r_d^2 where $r_d = (2/c)d$ and invariant with respect to the sign of d. Here we are dealing with a situation where the difference between the roots r_d is very large and thus quite possibly only one intersection between the exponential term on the left side of eq. (2) and its right side namely a quadratic in x, and thus only one solution.

As a numerical check and departing from the earlier physical chemistry problem in the earlier section, consider these particular values:

$$a_o \;=\; 1, \quad c \;=\; 2, \quad r_1 \;=\; 2, \qquad r_2 \;=\; 1 \quad \Rightarrow \quad d \;=\; r_d \;=\; \frac{1}{2}, \quad r_m \;=\; \frac{3}{2}\,.$$

The asymptotic series in eq. (19) with only the first 3 terms up to and including $O(1/d^3)$ yields the solution $x = 2.01739$ to within 4 decimals. Another test case, this time with some complex values:

$$a_o \;=\; 1, \quad c \;=\; 1, \quad r_1 \;=\; 2-i, \qquad r_2 \;=\; 1+i \quad \Rightarrow \quad d = \frac{1}{4} - \frac{1}{2}i, \quad r_d = \frac{1}{2} - i, \quad r_m = \frac{3}{2}$$

This same series with only 3 terms gives us $x = 1.9703 - 0.9430\,i$ to within 4 decimals. Thus, though initially motivated for the case of real numbers, these expansions can be used in the complex plane within certain restrictions.

2.3 Asymptotic series for large argument

The question arises what happens if we decide the left side z_0 of eq. (14) is large? For the principal branch when $z > 0$, taking logs of both sides of the equation governing the standard Lambert W function i.e. $We^W = z$ yields:

$$\ln[W(z)] \;=\; \ln(z) \;-\; W(z) \tag{20}$$

Recursive substitution yields successively:

$$\begin{array}{l} \ln(z) \\ \ln(z) - \ln(\ln(z)) \\ \ln(z) - \ln(\ln(z) - \ln(\ln(z))) \\ \ldots \end{array}$$

By taking logs on both sides of eq. (14) for the positive square root case only:

$$\ln(z) - \ln(U) = \sqrt{U^2 + d^2} \quad \text{or} \quad (\ln(z) - \ln(U))^2 = U^2 + d^2 \tag{21}$$

Thus, we consider two types of recursion.

$$U \;\rightarrow\; \sqrt{(\ln(z) - \ln(U))^2 - d^2} \tag{22}$$

$$U^2 \;\rightarrow\; \frac{1}{4}(-2\ln(z) + \ln(U^2) - 2d)(-2\ln(z) + \ln(U^2) + 2d) \tag{23}$$

The second recursion avoids the square root (and its messy consequences for recursion) and looks like a factored form involving a combination of asymptotic formulae for the standard W function. By successive substitution, we obtain:

$$U \approx \sqrt{\left(\ln(z) - \ln\left(\sqrt{\left(\ln(z) - \ln\left(\sqrt{\ldots \ln\left(\ln(z) - \ln\left(\sqrt{(\ln(z) - \ln(U))^2 - d^2}\right)\right)^2 \ldots - d^2}\right)\right)^2 - d^2}\right)\right)^2} \tag{24}$$

Table 1: Non-Linear transformations applied to Taylor series of eq.(12) for $r_d = 0.8$

no. of terms	$W(r_d)$ Taylor Series	Shanks	Levin t
1	-0.9999999996	-0.9999999996	-0.9999999996
2	-1.6400000000	-1.6400000000	-2.7777777780
3	-1.4352000000	-1.4848484850	-1.5213977230
4	-1.6099626670	-1.5294964030	-1.5192810810
5	-1.4421905070	-1.5246574640	-1.5243445560
6	-1.6265161880	-1.5271424650	-1.5267037510
7	-1.4108133840	-1.5280997520	-1.5277557490
⋮	⋮	⋮	⋮
	-1.528554071	-1.528554071	-1.528554071

and:

$$\begin{aligned} U^2 &\approx \frac{1}{4}\left(-2\ln(z) + \ln\left(\frac{1}{4}\left(-2\ln(z) + \ln(\ldots\right.\right. \\ &+ \ln\left(\frac{1}{4}\left(-2\ln(z) + \ln(U^2) - 2d)(-2\ln(z) + \ln(U^2) + 2d\right)\right) + \ldots + 2d\right)\Big)\Big) + 2d\Big) \end{aligned} \tag{25}$$

However, we find from experience that the argument z has to be *very large* indeed for these asympotic formulations to converge. This exercise is more to demonstrate the resemblence with the counterpart expansion for the standard W function, namely eq. (21). For computational value, sections 2.1 and 2.2 are more useful. Nonetheless, the very large z_0 argument is tractable.

2.4 Summation techniques

Finally, the series summation can be accelerated even *beyond* the radii of convergence using non-linear transformations as mentioned in the introduction. These transformations are applied to the sequence of partial sums and are capable of accelerating the convergence of a series and even sum divergent series (e.g. see the work of [13, 14]). We take the point of view that a Taylor or asymptotic series has all the desired "information", getting numbers from the series is a matter of a summation technique. For the series in r_d of the first section for both $W(\pm z_0)$, it was found that the series, when oscillating in r_d, could indeed be extended beyond their radius of convergence. This is demonstrated for the test case:

$$a_o \;=\; 1, \quad c \;=\; 1, \quad r_m \;=\; 1\;.$$

Here, the asymptotic solution of eq. (19) matches the extrapolated Taylor series of solution about $W_0(z_0)$ of (12) in 4 decimal places. Here $r_{d\ crit} \;\approx\; 0.64$ and we consider the regime when $r_d > r_{d\ crit}$, the alternating Taylor series is divergent. This Taylor series to order $O(r_d^{12})$ (6 terms in powers of r_d^2) is used for the t transformation of Levin [15] and the Shanks transformation [16]. To demonstrate agreement between the Taylor series and the outcome of the non-linear transformations, tables 1 and 2 compares the Taylor series of eq. (12) and the outcome of the Shanks and Levin t transformations for respectively $r_d = 0.8$ and $r_d = 1.5$. At the bottom of each table is listed what exact solution to the number of digits shown. The Taylor series of eq. (12) diverges violently when $r_d = 1.5$ but the non-linear transformations

Table 2: Non-Linear transformations applied to Taylor series of eq.(12) for $r_d = 1.5$

no. of terms	$W(r_d)$ Taylor Series	Shanks	Levin t
1	0.38889448	0.3888944774	0.3888944774
2	2.81078092	2.8107809190	-0.0743922833
3	-3.02541152	1.0991727410	-5.1438626370
4	24.71693722	1.7964086140	1.7380384290
5	-139.85949420	1.3876539200	1.5296581130
6	953.20098980	1.5894954440	1.5167708910
7	-6823.99405600	1.4791930140	1.5165517370
⋮	⋮	⋮	⋮
	1.516240428	1.516240428	1.516240428

converge nicely. Three terms of the asymptotic expansion in eq. (19) for $r_d = 1.5$, yield $x = 1.516240673$ which agrees with the exact solution starting from $W_0(z_0)$ to within 7 decimals. This demonstrates that the solutions of section 2.2 can match one of the solutions of section 2.1.

3 Conclusions

Previously [10] we had inferred a canonical form for a generalization as expressed by (2) and (3) and given both mathematical and physical justifications for it. Herein, we formulated Taylor series and asymptotic series useful for analysis and computation. We find that the results are similar to those governing the standard W function and represent a natural extension though the branch structure in the complex plane may differ.

This approach could be extended to higher order polynomials fitting the pattern of eq. (3). For example, when the right side of eq. (3) we can *complete the cube* in some special cases, i.e. for

$$x^3 \; + \; a\, x^2 \; + \; b\, x \; + \; c \; = \; \left(x \; + \; \frac{1}{3} \right)^3 \; - \; \left(\frac{1}{27}\, a^3 \; - \; c \right)^3 \quad \text{when} \quad b \; = \; \frac{a^2}{3}$$

which can allow a special case of eq.(3) and create a cubic relation counterpart of eq. (14):

$$\frac{e^{-c r_m}}{a_o} \; = \; Y^3 \; e^{c \; (Y^3 + d_3)^{1/3}} \tag{26}$$

where $(x - r_m)^3 \; = \; Y^3 + d_3$ and $d_3 \; = \; \frac{a^3}{27} - c$ and $r_m = -a/3$. However, for larger order polynomials and rational polynomials, this approach is quickly exhausted and one has to rely on numerical techniques which is very feasible.

Finally, the Taylor series summation can be accelerated even *beyond* the radii of convergence using non-linear transformations known as the Levin or Shanks transformations allowing a matching between the Taylor series and the asymptotic series. The resulting series can be converted into FORTRAN or C code using the interface between Maple and these languages [18].

References

[1] R. B. Mann and T. Ohta, *Exact solution for the metric and the motion of two bodies in* $(1+1)$*-dimensional gravity*, Phys. Rev. D. **55**, (1997), 4723-4747.

[2] P. S. Farrugia, R. B. Mann, and T. C. Scott, *N-body Gravity and the Schrödinger Equation*, Class. Quantum Grav. **24**, (2007), 4647-4659.

[3] T. C. Scott, J. F. Babb, A. Dalgarno, and J. D. Morgan III, *The Calculation of Exchange Forces: General Results and Specific Models*, J. Chem. Phys. **99**, (1993), 2841-2854.

[4] A. A. Frost, *Delta-Function Model. I. Electronic Energies of Hydrogen-Like Atoms and Diatomic Molecules*, J. Chem. Phys. **25**, (1956), 1150-1154.

[5] P. R. Certain and W. Byers Brown, *Branch Point Singularities in the Energy of the Delta-Function Model of One-Electron Diatoms*, Intern. J. Quantum Chem. **6**, (1972), 131-142.

[6] W. N. Whitton and W. Byers Brown, *The Relationship Between the Rayleigh-Schrödinger and Asymptotic Perturbation Theories of Intermolecular Forces*, Int. J. Quantum Chem. **10**, (1976), 71-86.

[7] T. C. Scott, A. Dalgarno, and J. D. Morgan III, *Exchange Energy of* H_2^+ *Calculated from Polarization Perturbation Theory and the Holstein-Herring Method*, Phys. Rev. Lett. **67**, pp. 1419-1422 (1991).

[8] T. C. Scott, J. F. Babb, A. Dalgarno, and J. D. Morgan III, *Resolution of a Paradox in the Calculation of Exchange Forces for* H_2^+, Chem. Phys. Lett. **203**, pp. 175-183 (1993).

[9] T. C. Scott, M. Aubert-Frécon and J. Grotendorst, *New approach for the electronic energies of the hydrogen molecular ion*, Chem. Phys. **324**, (2006), 323-338.

[10] T. C. Scott, R. B. Mann and R. E. Martinez II, *General Relativity and Quantum Mechanics: Towards a Generalization of the Lambert W Function*, AAECC, **17**, (2006) 41-47.

[11] (a) R. Corless, G. Gonnet, D. E. G. Hare and D. Jeffrey, *Lambert's W Function in Maple*, MapleTech **9**, ed. T. C. Scott, (Spring 1993); (b) R. Corless, G. Gonnet, D. E. G. Hare, D. Jeffrey and D. Knuth, *On the Lambert W Function*, Advances in Computational Mathematics, **5**, (1996), 329-359.

[12] A. C. Dixon, *On Burmann's Theorem*, Proc. London Math. Soc. **34**, pp. 151-153, (1902).

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

[14] E. J. Weniger, *Nonlinear Sequence Transformations for the Acceleration of Convergence and the Summation of Divergent Series*, Comput. Phys. Rep. **10**, (1989), 189-371.

[15] D. Levin, *Development of non-linear transformations of improving convergence of sequences*, Internat. J. Comput. Math. **B 3**, (1973), 371-388.

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

[17] B. W. Char, K. O. Geddes, G. H. Gonnet, B. L. Leong, M. B. Monagan and S. M. Watt, *First Leaves: A Tutorial Introduction to Maple V*, Springer-Verlag, New York, (1992).

[18] C. Gomez and T. C. Scott, *Maple programs for generating efficient FORTRAN code for serial and vectorised machines*, Comput. Phys. Commun., Thematic issue: Computer Algebra in Physics Research, **115**, pp. 548-562 (1998).

NSF Funding Opportunities for Symbolic Computation

Communicated by Erich Kaltofen and Alexey Ovchinnikov

The NSF CCF Core Programs, Algorithmic Foundation, is to fund as one of its areas research symbolic computation. Please, note the change of deadlines of this major funding opportunity this year:

- Medium projects: September 24, 2013 – October 15, 2013
- Large projects: November 4, 2013 – November 19, 2013
- Small projects: January 2, 2014 – January 17, 2014

For further information, please, refer to the program web page:

`http://www.nsf.gov/funding/pgm_summ.jsp?pims_id=503299&org=CISE`

and (new) solicitation:

`http://www.nsf.gov/funding/pgm_summ.jsp?pims_id=503220&org=CISE`

where you can find further details, NSF contact information for further questions, and what has been recently funded under the Core Programs.

ACM Communications in Computer Algebra, Vol. 47, No. 3, Issue 185, September 2013

The Computational Geometry Algorithms Library CGAL*

Efi Fogel† Monique Teillaud‡

Abstract

The Computational Geometry Algorithms Library (CGAL) is an open source software library that provides industrial and academic users with easy access to reliable implementations of efficient geometric algorithms.

Usage. CGAL is used in a diverse range of domains requiring geometric computation such as computer graphics, scientific visualization, computer aided design and modeling, geographic information systems, molecular biology, medical imaging, and many more. Since CGAL provides a wide range of components, we restrict ourselves to mentioning just a few here.

As an example application of CGAL, a series of packages are provided which are useful in robotics and automation: Minkowski sums, offset polygons, Boolean operations on curved regions. The high precision of CGAL allows users to solve geometric problems involving motion in restricted environments, such as those arising in assembly planning.

The robustness and efficiency of components such as the Delaunay triangulation and mesh construction and manipulation packages makes CGAL attractive for simulations, in particular those involving proteins, particle physics, fluid dynamics, medical modeling, biophysics, geophysics, and astronomy. Indeed, the aforementioned components are largely used in these areas.

Some support for manipulations of polynomials and for solving univariate polynomial equations and bivariate polynomial systems is also provided, as well as handling for convex quadratic programs.

History of the CGAL **Open Source Project.** Several European research groups started to develop their own small geometry libraries in the early 90's. In 1996, a consortium of eight sites was created to gather the work of these groups into a single software library, namely CGAL. Their main goal was to promote research in computational geometry and to translate the results into **robust** software suitable for industrial applications.

Around this time the Computational Geometry Impact Task Force Report [C+96, C+99] made a series of recommendations. Amongst these recommendations, the production and distribution of usable (and useful) geometric software, and the need to establish a reward structure for software implementations in academia, were key.

On November 2003, when version 3.0 was released, CGAL officially became an Open Source project, allowing new contributors to join the project.

License. CGAL is distributed under the GPL license (apart from a few basic parts, which are distributed under the LGPL license). In particular, it is publicly and freely available for academic use. Commercial licenses are offered by Geometry Factory, a company founded in 2003 mainly for this purpose.

*`http://www.cgal.org`

†Tel-Aviv University `http://acg.cs.tau.ac.il/people/efifogel`

‡INRIA Sophia Antipolis-Méditerranée `http://www-sop.inria.fr/members/Monique.Teillaud/`

Editorial board. The CGAL editorial board was created in 2001. It currently consists of thirteen members. The main task of the editorial board is to assure the quality of CGAL. It is also responsible for making decisions about technical matters and coordinating communication and promotion of CGAL.

All new packages must be submitted to the Editorial Board to be reviewed before they can be accepted and integrated into the library, in a process that is very similar to the standard review process for papers published in conference proceedings or journals. More information about the submission process is available at `http://www.cgal.org/review_process_rules.html`.

Style and Techniques. CGAL is a unique library both in general and within the field of computational geometry in particular, as it consists of a large number of components with a homogeneous API (Application Programming Interface). Careful choices in design and programming style have made CGAL the *de facto* standard in the field of applied computational geometry. Its development started whilst the standardization process of C++ and the STL (Standard Template Library) was taking place. Indeed, the programming style is very close to the programming style of STL; it rigorously adheres to the generic programming paradigm—a discipline that consists of the gradual lifting of concrete algorithms abstracting over details, while retaining the algorithm semantics and efficiency. The programming style of CGAL also facilitates the process of interfacing with third party software.

Each package comes with header files consisting not only of the interface, but also the generic implementation of the package code, comprehensive and didactic on-line documentation, a set of non-interactive standalone example programs, and an optional interactive demo with a graphical user interface.

Robustness. CGAL follows the exact geometric-computation paradigm, which simply amounts to ensuring that errors in predicate evaluations do not occur; it guarantees robustness of the applied algorithms.

We additionally remark that every package also includes a collection of function and regression test. [1] The tests provided by each package are combined into one place to form the CGAL test suite. This test suite is run daily and its results are automatically assembled, analyzed, and reported.

Impact. Measuring the impact of software is a difficult task, especially in the Open Source software community. Even if some hard numbers can be found, they can be difficult to interpret. The following facts may shed some light on the impact of CGAL:

- There are roughly 1000 downloads per month from `http://gforge.inria.fr/`
- CGAL is included in various software distribution channels, such as Fedora, Debian/Ubuntu, and Macports.
- The range of uses of CGAL is very broad, as shown by the sample list of projects using CGAL, which is available at `http://www.cgal.org/projects.html`. In addition, many projects shown in `http://acg.cs.tau.ac.il/projects` use CGAL or even describe the development of a CGAL component.
- The CGAL triangulation packages were integrated in Matlab 2009a.[2]
- Springer has published a book entitled "CGAL Arrangements and Their Applications" authored by some of the developers of the *2D Arrangements* package and its derivatives.
- Concerning the public mailing lists, there are currently

[1] However these are not distributed as part of the public releases.

[2] Watch the video at `http://www.mathworks.com/products/demos/shipping/matlab/New-MATLAB-Mathematics-Features-in-R2009a.html`

- 4000 subscribers to the announcement list `cgal-announce@lists-sop.inria.fr`
- 1500 to the public discussion list `cgal-discuss@lists-sop.inria.fr`, with high traffic: users are free to ask questions, which are often rapidly answered by the developers or other users.

Acknowledgments
We thank Ross Hemsley for helping us distilling the text.

References

[C+96] Bernard Chazelle et al. Application challenges to computational geometry: CG impact task force report. Technical Report TR-521-96, Princeton Univ., April 1996.

[C+99] Bernard Chazelle et al. Application challenges to computational geometry: CG impact task force report. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, volume 223 of *Contemporary Mathematics*, pages 407–463. American Mathematical Society, Providence, 1999.

ACM Communications in Computer Algebra, Vol 47, No. 3, Issue 185, September 2013

ISSAC 2013 Poster Abstracts

Communicated by Alin Bostan

Revisiting QRGCD and Comparison with ExQRGCD*

Kosaku Nagasaka† and Takaaki Masui‡
†Kobe University, 3-11 Tsurukabuto, Nada-ku, Kobe 657-8501 JAPAN
‡Kobe-Takatsuka High School, 9-1 Mikatadai, Nishi-ku, Kobe 651-2277 JAPAN
† nagasaka@main.h.kobe-u.ac.jp, ‡ masui.takaaki@gmail.com

1 Introduction

In this poster, we are interested in computing "approximate polynomial GCD": for the input polynomials $f(x), g(x) \in \mathbb{R}[x]$, we call the polynomial $d(x) \in \mathbb{R}[x]$ "approximate polynomial GCD" of tolerance $\varepsilon \in \mathbb{R}_{\geq 0}$ if it satisfies

$$f(x) + \Delta_f(x) = f_1(x)d(x),\ g(x) + \Delta_g(x) = g_1(x)d(x)$$

for some polynomials $\Delta_f(x), \Delta_g(x), f_1(x), g_1(x) \in \mathbb{R}[x]$ such that $\deg(\Delta_f) \leq \deg(f)$, $\deg(\Delta_g) \leq \deg(g)$, $\|\Delta_f\|_2 < \varepsilon \|f\|_2$ and $\|\Delta_g\|_2 < \varepsilon \|g\|_2$ where $\|\cdot\|_2$ denotes the 2-norm. Although there are many studies, we revisit the QRGCD algorithm[2] which is one of algorithms based on matrix decompositions and is also implemented as a part of the SNAP package of Maple. It is notable that the QRGCD algorithm is very simple and has been used as the benchmark algorithm for newly proposed algorithms. The framework of the QRGCD algorithm is as follows. For details, please refer the original paper[2].

1. Compute the QR decomposition of $Syl(f, g)$: $Syl(f, g) = QR$.
2. Find the gap between the k-th and $(k+1)$-th row vectors $\vec{r}_k, \vec{r}_{k+1}$ of R and form the polynomial with coefficients $\vec{r}_k$, which is an approximate polynomial GCD (or its factor).
3. Apply the same procedures to the reversal polynomials of cofactors since R may not have the approximate common divisor whose roots are outside the unit circle in the complex plane.

However, since QRGCD was proposed in the early stage of approximate GCD, its theoretical background is not enough analyzed from the current theoretical point of view and the official implementation is different from the paper. For example, Bini and Boito[1] reported that QRGCD failed to recognize the correct degree of GCD for polynomials with small leading coefficients. This result is caused by the preconditioning routine in the official implementation hence QRGCD works well for such polynomials. Therefore, our aim consists of two parts: 1) verifying the efficiency of QRGCD with much theoretical considerations, and 2) improving the framework and algorithm to be more accurate and able to satisfy the given tolerance.

2 Notable Facts on QRGCD and its Implementation in SNAP

Recently, the concept of "structured perturbation" is important in the theory of approximate GCD. However, at the time of QRGCD proposed, this concept is not widely discussed hence there are unclear statements in the original paper from this point of view. Analyzing their theory from this concept could be interesting. For example, any relationship between the QR factoring and structured perturbation, the reason that the QR factoring can not detect the roots outside the unit circle and so on.

*This work was supported in part by JSPS KAKENHI Grant Number 22700011.

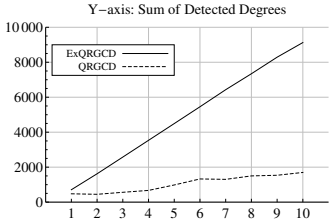
Moreover, we found the 4 significant differences between the original algorithm and the SNAP implementation. According to our personal conversations, some of them are implemented by the original authors and others may be by H. Kai, the person implemented it in the SNAP package. The differences are 1) the preconditioning routine "find non-zero terms", 2) the matrix norm used, 3) the polynomials to be applied to the algorithm "Split", and 4) the fail-safe retry loop. The first one may be the cause that many people think QRGCD is weak for polynomials with small leading coefficients. Without this, QRGCD works well for such polynomials. Other differences seem to be some techniques to make QRGCD working well.

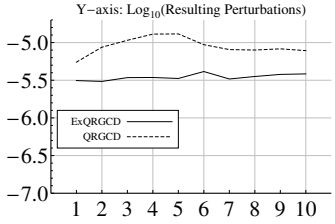
3 Improved QRGCD Algorithm (ExQRGCD)

We refine the framework of QRGCD from the different approach with recent theoretical results of approximate polynomial GCD and propose the improved algorithm called "ExQRGCD". The most notable difference is that our algorithm detects a row vector of R by estimating relative distance from the expected approximate GCD while QRGCD detects by estimating absolute distance. As a result, ExQRGCD works more accurately. For example, it works for the following polynomials (QRGCD does not work well for this kind of polynomials unfortunately). For $i = 1, \ldots, 10$, we have generated 100 pairs of (f, g) such that

$$f(x) = d(x)\prod_{j=1}^{2i}(x-\omega_{f,j})\prod_{j=1}^{2i}(x-\hat{\omega}_{f,j}),\ g(x) = d(x)\prod_{j=1}^{2i}(x-\omega_{g,j})\prod_{j=1}^{2i}(x-\hat{\omega}_{g,j})$$

where $d(x) = \prod_{j=1}^{3i}(x-\omega_{d,j})\prod_{j=1}^{3i}(x-\hat{\omega}_{d,j})$, $\omega_{\cdot,j} = O(10^{-2}), \hat{\omega}_{\cdot,j} = O(10^2)$ is randomly chosen, $f(x), g(x)$ are normalized (i.e. $\|f(x)\|_2 = \|g(x)\|_2 = 1$) and rounded with $Digits := 10$. We computed with tolerance 10^{-5}. Figure 1 shows the result that ExQRGCD is explicitly better than QRGCD though as for computing time, ExQRGCD is 39.8 times slower than QRGCD (note that QRGCD outputs failure for 62% pairs so computing time is very fast for the rest easy cases). The average of resulting perturbations of ExQRGCD is also better. For other random generated examples, ExQRGCD is almost 2 times slower than QRGCD since ExQRGCD is more conservative than QRGCD for detecting approximate GCD hence it computes QR decompositions several times (this is more than that of QRGCD in general).





Number of fail events or wrong Δ_* (QRGCD)											
i=1	2	3	4	5	6	7	8	9	10	Δ_f	Δ_g
29	63	68	71	64	60	66	66	69	68	110	119

Number of fail events or wrong Δ_* (ExQRGCD)											
i=1	2	3	4	5	6	7	8	9	10	Δ_f	Δ_g
0	0	0	0	0	0	0	0	0	0	0	0

Figure 1: Sum of Detected Degrees and Resulting Perturbations (failure is not counted)

We note that our preliminary implementations on Maple and written in C, and generated polynomial data are available: "`http://wwwmain.h.kobe-u.ac.jp/~nagasaka/research/snap/issac2013/`".

References

[1] D. A. Bini and P. Boito. A fast algorithm for approximate polynomial GCD based on structured matrix computations. In *Numerical methods for structured matrices and applications*, volume 199 of *Oper. Theory Adv. Appl.*, pages 155–173. Birkhäuser Verlag, Basel, 2010.

[2] R. M. Corless, S. M. Watt, and L. Zhi. *QR* factoring to compute the GCD of univariate approximate polynomials. *IEEE Trans. Signal Process.*, 52(12):3394–3402, 2004.

Schützenberger's factorization on $q-$stuffle Hopf algebra

C. Bui[♭], G. H. E. Duchamp[♯], Hoang Ngoc Minh[◊,♯]
[♭]Hué University - College of sciences, 77 - Nguyen Hue street - Hué city, Viêt Nam
[♯]Institut Galilée, LIPN - UMR 7030, CNRS - Université Paris 13, F-93430 Villetaneuse, France,
[◊]Université Lille II, 1, Place Déliot, 59024 Lille, France

Schützenberger's monoidal factorization [9] has been introduced and plays a central role in the renormalization [7] of associators which are formal power series in non commutative variables[1]. The coefficients of these power series are polynomial at positive integral multi-indices of Riemann's zêta function[2] [5, 10] and they satisfy quadratic relations [1] which can be explained through Lyndon words. These relations can be obtained by identification of the local coordinates on a bridge equation connecting the Cauchy and Hadamard algebras of polylogarithmic functions and use the factorizations of the non commutative generating series of polylogarithms [6] and of harmonic sums [7]. This equation is mainly a consequence of the double isomorphy between these structures to respectively the shuffle [6] and stuffle [3] algebras both admitting the Lyndon words as a transcendence basis.

Symbolic computation allows us to introduce a formal variable q in order to better understand the mechanisms of the shuffle and to obtain algorithms on stuffles. We will then examine the q-stuffle interpolating between the shuffle [9], stuffle [8] and minus-stuffle [3]. In particular, we will give an effective construction of pair of bases in duality. It uses essentially an adapted version of the Eulerian projector in order to obtain the primitive elements of the q-stuffle Hopf algebra and they are obtained thanks to the computation of the logarithm of the diagonal series. This study completes the treatment for the stuffle [7] and boils down to the shuffle [9].

More precisely, let $Y = \{y_s\}_{s\geq 1}$ be an alphabet with the total order $y_1 > y_2 > \cdots$. Let also $\mathbf{k}$ be a unitary $\mathbb{Q}$-algebra containing q. One defines the q-stuffle, or its dual co-product, as follows, for any $y_s, y_t \in Y$ and $u, v \in Y^*$,

$$u \sqcup\!\sqcup_q 1_{Y^*} = 1_{Y^*} \sqcup\!\sqcup_q u = u \quad \text{and} \quad y_s u \sqcup\!\sqcup_q y_t v = y_s(u \sqcup\!\sqcup_q y_t v) + y_t(y_s u \sqcup\!\sqcup_q v) + q y_{s+t}(u \sqcup\!\sqcup_q v), \tag{1}$$

$$\Delta_{\sqcup\!\sqcup_q}(1_{Y^*}) = 1_{Y^*} \otimes 1_{Y^*} \quad \text{and} \quad \Delta_{\sqcup\!\sqcup_q}(y_s) = y_s \otimes 1_{Y^*} + 1_{Y^*} \otimes y_s + q \sum_{s_1+s_2=s} y_{s_1} \otimes y_{s_2}. \tag{2}$$

This product is commutative, associative and unital. With the co-unit defined by, $\epsilon(P) = \langle P \mid 1_{Y^*}\rangle$, for $P \in \mathbf{k}\langle Y\rangle$, one gets $\mathcal{H}_{\sqcup\!\sqcup_q} = (\mathbf{k}\langle Y\rangle, \mathtt{conc}, 1_{Y^*}, \Delta_{\sqcup\!\sqcup_q}, \epsilon)$ and $\mathcal{H}^{\vee}_{\sqcup\!\sqcup_q} = (\mathbf{k}\langle Y\rangle, \sqcup\!\sqcup_q, 1_{Y^*}, \Delta_{\mathtt{conc}}, \epsilon)$ which are mutually dual bialgebras and, in fact, Hopf algebras because they are $\mathbb{N}$-graded by the weight.

Group-like elements, redefined below, form a group for which the log-exp correspondence is explained by as follows

Lemma 1 (q-extended Friedrichs criterium) *Let $S \in \mathbf{k}\langle\langle Y\rangle\rangle$ (for 2., we suppose in addition that $\langle S \mid 1_{Y^*}\rangle = 1$).*

1. *S is* primitive*, i.e. $\Delta_{\sqcup\!\sqcup_q} S = S \otimes 1_{Y^*} + 1_{Y^*} \otimes S$, if and only if, for any $u, v \in Y^+$, $\langle S \mid u \sqcup\!\sqcup_q v\rangle = 0$.*
2. *S is* group-like*, i.e. $\Delta_{\sqcup\!\sqcup_q} S = S \otimes S$, if and only if, for any $u, v \in Y^+$, $\langle S \mid u \sqcup\!\sqcup_q v\rangle = \langle S \mid u\rangle\langle S \mid v\rangle$.*
3. *S is group-like if and only if $\log S$ is primitive.*

Proposition 1 *Let $\mathcal{D}_Y = \sum_{w\in Y^*} w \otimes w$ be the diagonal series over Y. Then*

1. $\log \mathcal{D}_Y = \sum_{w\in Y^+} w \otimes \pi_1(w)$, *where* $\pi_1(w) = w + \sum_{k\geq 2} \frac{(-1)^{k-1}}{k} \sum_{u_1,\ldots,u_k\in Y^+} \langle w \mid u_1 \sqcup\!\sqcup_q \ldots \sqcup\!\sqcup_q u_k\rangle u_1 \ldots u_k$.
2. *For any $w \in Y^*$, we have* $w = \sum_{k\geq 0} \frac{1}{k!} \sum_{u_1,\ldots,u_k\in Y^+} \langle w \mid u_1 \sqcup\!\sqcup_q \ldots \sqcup\!\sqcup_q u_k\rangle \pi_1(u_1)\ldots\pi_1(u_k)$.

[1]These associators were introduced in quantum field theory by Drinfel'd and the universal associator, *i.e.* Φ_{KZ}, was obtained with explicit coefficients which are polyzêtas and regularized polyzêtas [5].
[2]These values are usually abbreviated MZV's by Zagier [10] and are also called polyzêtas by Cartier [1].

Let $\mathcal{P} = \{P \in \mathbb{Q}\langle Y\rangle \mid \Delta_{\sqcup\!\sqcup_q} P = P \otimes 1 + 1 \otimes P\}$ be the set of primitive polynomials. Since, in virtue of $\Delta_{\sqcup\!\sqcup_q}\pi_1(w) = \pi_1(w) \otimes 1 + 1 \otimes \pi_1(w)$, $\mathrm{Im}(\pi_1) \subseteq \mathcal{P}$, we can state the following

Theorem 1 ([2]) *1. Let $\{\Pi_l\}_{l\in\mathcal{L}ynY}$ be defined by, for any $y_k \in Y, \Pi_{y_k} = \pi_1(y_k)$ and for any $l \in \mathcal{L}ynX$ of standard factorization $l = (s, r), \Pi_l = [\Pi_s, \Pi_r]$. Then $\{\Pi_l\}_{l\in\mathcal{L}ynY}$ forms a basis of $\mathcal{P}$.*

2. Let $\{\Pi_w\}_{w\in Y^*}$ be defined by, for any $w \in Y^*$ such that $w = l_1^{i_1}\dots l_k^{i_k}, l_1 > \dots > l_k, l_1\dots, l_k \in \mathcal{L}ynY$, $\Pi_w = \Pi_{l_1}^{i_1}\dots\Pi_{l_k}^{i_k}$. Then $\{\Pi_w\}_{w\in Y^*}$ forms a basis of $\mathbf{k}\langle Y\rangle$.

3. Let $\{\Sigma_w\}_{w\in Y^*}$ be the family of the quasi-shuffle algebra obtained by duality with $\{\Pi_w\}_{w\in Y^*}$. Then $\{\Sigma_w\}_{w\in Y^*}$ generates freely the quasi-shuffle algebra.

4. The family $\{\Sigma_l\}_{l\in\mathcal{L}ynY}$ forms a transcendence basis of $(\mathbf{k}\langle Y\rangle, \sqcup\!\sqcup_q)$.

We now give formulas which permit to compute the basis $\{\Sigma_w\}_{w\in Y^*}$ without inverting a huge Gram matrix.

Theorem 2 (q-extended Schützenberger's factorization, [2]) *1. For any $y \in Y$, $\Sigma_y = y$.*

2. For any $y_{s_1}\dots y_{s_k} \in \mathcal{L}ynX$, $\displaystyle\Sigma_{y_{s_1}\dots y_{s_k}} = \sum_{\substack{\{s'_1,\cdots,s'_i\}\subset\{s_1,\cdots,s_k\}, l_1\geq\cdots\geq l_n\in\mathcal{L}ynY \\ (y_{s_1}\cdots y_{s_k})\overset{*}{\leftarrow}(y_{s'_1},\cdots,y_{s'_n},l_1,\cdots,l_n)}} \frac{q^{i-1}}{i!} y_{s'_1+\cdots+s'_i}\Sigma_{l_1\cdots l_n}.$

3. For any $w = l_1^{i_1}\dots l_k^{i_k}$, with $l_1, \dots, l_k \in \mathcal{L}ynY$ and $l_1 > \dots > l_k$, $\displaystyle\Sigma_w = \frac{\Sigma_{l_1}^{\sqcup\!\sqcup_q i_1} \sqcup\!\sqcup_q \dots \sqcup\!\sqcup_q \Sigma_{l_k}^{\sqcup\!\sqcup_q i_k}}{i_1!\dots i_k!}.$

4. $\displaystyle\mathcal{D}_Y = \sum_{w\in Y^*} \Sigma_w \otimes \Pi_w = \prod_{l\in\mathcal{L}ynY}^{\searrow} \exp(\Sigma_l \otimes \Pi_l).$

Theorems 1.1 and 2.2 are based mainly on respectively the logarithm of the diagonal series $\mathcal{D}_Y$ and the standard sequences [9, 2] and lead to simplified algorithms getting bases in duality as shown in the following

Example 1

$$
\begin{aligned}
\Pi_{y_2} &= y_2 - \tfrac{q}{2}y_1^2,\\
\Pi_{y_2y_1} &= y_2y_1 - y_1y_2,\\
\Pi_{y_3y_1y_2} &= y_3y_1y_2 - \tfrac{q}{2}y_3y_1^3 - qy_2y_1^2y_2 + \tfrac{q^2}{4}y_2y_1^4 - y_1y_3y_2 + \tfrac{q}{2}y_1y_3y_1^2 + \tfrac{q}{2}y_1^2y_2^2 - \tfrac{q^2}{2}y_1^2y_2y_1^2 - y_2y_3y_1\\
&+ \tfrac{q}{2}y_2^2y_1^2 + y_2y_1y_3 + \tfrac{q}{2}y_1^2y_3y_1 - \tfrac{q}{2}y_1^3y_3 + \tfrac{q^2}{4}y_1^4y_2,\\
\Pi_{y_3y_1y_2y_1} &= y_3y_1y_2y_1 - y_3y_1^2y_2 - \tfrac{q}{2}y_2y_1^2y_2y_1 - y_1y_3y_2y_1 + y_1y_3y_1y_2 + \tfrac{q}{2}y_1^2y_2^2y_1 - \tfrac{q}{2}y_1^2y_2y_1y_2 - y_2y_1y_3y_1\\
&+ \tfrac{q}{2}y_2y_1y_2y_1^2 + y_2y_1^2y_3 + y_1y_2y_3y_1 - \tfrac{q}{2}y_1y_2^2y_1^2 - y_1y_2y_1y_3 + \tfrac{q}{2}y_1y_2y_1^2y_2.\\
\Sigma_{y_2} &= y_2,\\
\Sigma_{y_2y_1} &= y_2y_1 + \tfrac{q}{2}y_3,\\
\Sigma_{y_3y_2y_1} &= y_3y_1y_2 + y_3y_2y_1 + qy_3^2 + \tfrac{q}{2}y_4y_2 + \tfrac{q^2}{3}y_6 + \tfrac{q}{2}y_5y_1,\\
\Sigma_{y_3y_1y_2y_1} &= 2y_3y_2y_1^2 + qy_3y_2^2 + y_3y_1y_2y_1 + \tfrac{3q}{2}y_3^2y_1 + \tfrac{q}{2}y_3y_1y_3 + \tfrac{q^2}{2}y_3y_4 + \tfrac{q}{2}y_4y_2y_1 + \tfrac{q^2}{4}y_4y_3 + qy_5y_1^2 + \tfrac{q^2}{2}y_5y_2 + \tfrac{q^2}{2}y_6y_1 + \tfrac{q^3}{8}y_7.
\end{aligned}
$$

In conclusion, since the pioneering works of Schützenberger and Reutenauer [9], the question of computing bases in duality (maybe at the cost of a more involved procedure, but without inverting a Gram matrix) remained open in the case of cocommutative deformations of the shuffle product. We have given such a procedure allowing a great simplification for an interpolation between shuffle and stuffle. In the next framework, this product will be continuously deformed, in the most general way while remaining commutative [4].

References

[1] P. Cartier.– *Fonctions polylogarithmes, nombres polyzêtas et groupes pro-unipotents*, Sém BOURBAKI, 53ème 2000-2001, n°885.

[2] C. Bui, G. H. E. Duchamp, V. Hoang Ngoc Minh.– *Schützenberger's factorization on the (completed) Hopf algebra of q−stuffle product.* arXiv:1305.4450

[3] C. Costermans and Hoang Ngoc Minh.– *Noncommutative algebra, multiple harmonic sums and applications in discrete probability*, J. of Sym. Comp. (2009), pp. 801-817.

[4] J-Y. Enjalbert, Hoang Ngoc Minh.– *Combinatorial study of Hurwitz colored polyzêtas*, Discrete Mathematics, 1. 24 no. 312 (2012), p. 3489-3497.

[5] T.Q.T. Lê and J. Murakami.– *Kontsevich's integral for Kauffman polynomial*, Nagoya Math., pp 39-65, 1996.

[6] Hoang Ngoc Minh, M. Petitot and J. Van der Hoeven.– *Computation of the monodromy of generalized polylogarithms*, ISSAC'98, Rostock, Allemagne, Aug. 1998.

[7] Hoang Ngoc Minh.– *On a conjecture by Pierre Cartier about a group of associators*, to appear (2013).

[8] M. Hoffman.– *Quasi-shuffle products*, J. of Alg. Cominatorics, 11 (2000), pages 49-68.

[9] C. Reutenauer.– *Free Lie Algebras*, Lon. Math. Soc. Mono, New Series-7, Oxford Sc. Pub., 1993.

[10] D. Zagier.– *Values of zeta functions and their applications*, in "First European Congress of Mathematics", vol. 2, Birkhäuser (1994), pp. 497-512.

Fast parallel GCD algorithm of many integers

Sidi M. SEDJELMACI
LIPN, CNRS UMR 7030
University of Paris-Nord
Av. J.-B. Clément, 93430 Villetaneuse, France
sms@lipn.univ-paris13.fr

Abstract: We present a new parallel algorithm which computes the GCD of n integers of $O(n)$ bits in $O(n/\log n)$ time with $O(n^{2+\epsilon})$ processors, for any $\epsilon > 0$ on CRCW PRAM model.

The computation of the GCD of two integers is not known to be in the NC parallel class, nor it is known to be P-complete [1]. The best parallel performance was first obtained by Chor and Goldreich [2], then by Sorenson [7] and Sedjelmaci [5] since they propose, with different approaches, parallel integer GCD algorithms which can be achieved in $O(n/\log n)$ time with $O(n^{1+\epsilon})$ number of processors, for any $\epsilon > 0$, in PRAM CRCW model. A naive approach, using a binary tree computation to compute the GCD of n integers of $O(n)$ bits would require $O(n)$ parallel time with $O(n^{2+\epsilon})$ processors. One may also use the existing parallel GCD algorithms of two integers and try to adapt them to design a GCD for many integers. However, it is not obvious how to find a parallel GCD for n integers which conserve the same $O(n/\log n)$ time, with $O(n^{2+\epsilon})$ processors, which is roughly the bit-size of all the n input integers. In this paper, we prove that we can compute the GCD of n integers of $O(n)$ bits, in only $O(n/\log n)$ parallel time with $O(n^{2+\epsilon})$ processors, for any $\epsilon > 0$ on CRCW PRAM model, in the worst case. Another probabilistic approach is given in [3]. To our knowledge, it is the first deterministic algorithm which computes the GCD of many integers with this parallel performance and polynomial work. Our algorithm, called Δ-GCD is the following:

Input: A set $A = \{ a_0, a_1, \cdots, a_{n-1} \}$ of n distinct positive integers, $a_i < 2^n$, with $n \geq 4$.
Output: $\gcd(a_0, a_1, \cdots, a_{n-1})$.

$\alpha := a_0\,;\ I := 0\,;\ p := n\,;$
While $(\alpha > 1)$ **Do**
  For $(i = 0)$ **to** $(n-1)$ **ParDo**
    If $(0 < a_i \leq 2^n/p)$ **Then** $\{\ \alpha := a_i\,;\ I := i\,;\ \}$
  Endfor
  If $(\alpha > 2^p/n)$ **Then** /* Compute in parallel I, J and α */
    $\alpha := \min \{\, | a_i - a_j | > 0 \,\} = a_I - a_J$; $a_I := \alpha$;
  Endif
  For $(i = 0)$ **to** $(n-1)$ **ParDo** /* Reduce all the a_i's */
    If $(i \neq I)$ **Then** $a_i := a_i \bmod \alpha$;
  Endfor /* $\forall\, i\,,\ 0 \leq a_i \leq \alpha$ */
  If $(\forall\, i \neq I\,,\ a_i = 0\,)$ **Then Return** α ; /* Here $\alpha = \gcd(a_0, \cdots, a_{n-1})$ */
  $p := np\,;$
Endwhile
Return α.

We use a weak version of the function min based the pigeonhole principle, where only the $O(\log n)$ leading bits of the integers are considered. The integer α is, at each while iteration, $O(\log n)$ bits less. More details for the computations of I, J and α are given in [6], as well as a first C program checking the correctness of the Δ-GCD algorithm.

Theorem : The $\Delta - GCD$ algorithm computes in parallel the GCD of n integers of $O(n)$ bits in length, in $O(n/\log n)$ time using $O(n^{2+\epsilon})$ processors on CRCW PRAM model, with $\epsilon > 0$.

Proof: (Sketch, see [6]). The algorithm terminates after $O(n/\log n)$ loop iterations. Let t_i be the time cost at iteration i, $1 \leq i \leq N$, with $N = O(n/\log n)$. Let k_i be the maximum bit length of all the quotients $q_j = \lfloor a_j/\alpha \rfloor$, with $\sum_{i=1}^{N} k_i \leq n$. We prove that $t_i = O(\min\{\frac{k_i}{\log n}, \log n\})$. The total number of processors is $n \times O(n^{1+\epsilon}) = O(n^{2+\epsilon})$ and the parallel time is then $t(n) = \sum_{i=1}^{N} t_i = \sum_{i=1}^{N} \min(\{\frac{k_i}{\log n}, \log n\}) = \sum_{k_i < \log n} 1 + \sum_{\log n < k_i < \log^2 n} \frac{k_i}{\log n} + \sum_{k_i > \log^2 n} \log n = O(n/\log n)$. □

A Blankinship-like algorithm can be easily designed to compute Extended GCD, and an upper bound of the multipliers [4] could be considered as well. A slightly modified Rosser's algorithm (pivoting with α) can be used to solve linear Diophantine equations. Moreover, a $O(n^2/\log n)$ sequential version of Δ-GCD should be considered with precomputed lookup tables for arithmetic operations on $O(\log n)$ bit integers.

References

[1] A. Borodin, J. von zur Gathen and J. Hopcroft, Fast parallel matrix and GCD computations, *Information and Control,* 52, 3, 1982, 241–256

[2] B. Chor and O. Goldreich, An improved parallel algorithm for integer GCD, Algorithmica, 5, 1990, 1-10

[3] G. Cooperman, S. Feisel, J. von zur Gathen and G. Havas, GCD of many integers, *Lect. Notes in Comp. Sci., Springer-Verlag, Berlin,* 1627 (1999), 310–317

[4] G. Havas, S. Majewski, Extended gcd calculation, *Congressus Numerantium, 111,* 1627 (1998), 104-114

[5] S.M. Sedjelmaci, On A Parallel Lehmer-Euclid GCD Algorithm, in Proc. of the International Symposium on Symbolic and Algebraic Computation (ISSAC'2001), 2001, 303-308

[6] S.M. Sedjelmaci, Fast Parallel GCD algorithm of many integers, lipn.univ-paris13.fr/∼sedjelmaci, Rapport interne, LIPN, April, 2013

[7] J. Sorenson, Two Fast GCD Algorithms, J. of Algorithms, 16, 1994, 110-144

ACM Communications in Computer Algebra, Vol 47, No. 3, Issue 185, September 2013

Gelfand-Kirillov dimensions of differential difference modules via Gröbner bases

Xiangui Zhao
Department of Mathematics, University of Manitoba
Winnipeg, Canada, R3T 2N2
umzha493@cc.umanitoba.ca

Introduction. Differential-difference algebras were defined by Mansfield and Szanto in [5], which arose from the calculation of symmetries of discrete systems (c.f., [2]). Mansfield and Szanto developed the Gröbner basis theory of differential difference algebras over a field by using a special kind of left admissible orderings (which they called differential difference orderings). We generalize the main results of [5] to any left admissible ordering, and apply the generalized results to compute the Gelfand-Kirillov dimensions of cyclic differential difference modules.

Definition of differential difference algebras. Let k be a field, R be a k-algebra and integers $m, n \geq 1$. Suppose that $R[D; \mathrm{id}, \delta] = R[D_1; \mathrm{id}, \delta_1] \cdots [D_n; \mathrm{id}, \delta_n]$ and $R[S; \sigma, 0] = R[S_1; \sigma_1, 0] \cdots [S_m; \sigma_m, 0]$ are two Ore algebras ([5]) such that $\sigma_i \circ \delta_j = \delta_j \circ \sigma_i$ for $1 \leq i \leq m, 1 \leq j \leq n$. Furthermore, suppose that each $\sigma_i : R \to R$, $1 \leq i \leq m$, can be extended to a k-algebra automorphism $\sigma_i : R[D; \mathrm{id}, \delta] \to R[D; \mathrm{id}, \delta]$ such that $\sigma_i(D_j) = \sum_{l=1}^{n} a_{ijl} D_l$, $a_{ijl} \in R$. Let F be the free R-R bi-module with basis $\{S_1, \ldots, S_m, D_1, \ldots, D_n\}$, T be the tensor algebra on F over R, and K be the two-sided ideal in T generated by the set of the following elements of T:
(1) $D_i r - r D_i - \delta_i(r), 1 \leq i \leq n, r \in R$; (2) $S_i r - \sigma_i(r) S_i, 1 \leq i \leq m, r \in R$;
(3) $S_i S_j - S_j S_i, 1 \leq i, j \leq m$; (4) $D_i D_j - D_j D_i, 1 \leq i, j \leq n$;
(5) $D_i S_j - S_j \sigma_j(D_i), 1 \leq i \leq n, 1 \leq j \leq m$.
Then the R-algebra T/K, denoted by $R[D; \mathrm{id}, \delta][S; \sigma, 0]$, is called a *differential difference algebra* of type (m, n), or DD-algebras for short.

DD-algebras are generalizations of commutative polynomial algebras, Ore extensions, skew polynomials of derivation (or automorphism) type, and quantum planes. Since elements in S do not commute with those in D in general, DD-algebras are different from difference-differential rings (see, e.g., [6]). The following example distinguishes DD-algebras from algebras of solvable type [3], or PBW extensions [1], or G-algebras [4].

Example. Let $A = k[D; \mathrm{id}, 0][S; \sigma, 0]$ be a DD-algebra of type $(1, 2)$ with $\sigma_1(D_1) = D_2$ and $\sigma_1(D_2) = D_1$. Then $D_1 S_1 = S_1 D_2$ and $D_2 S_1 = S_1 D_1$. Hence A is not an algebra of solvable type (or a PBW extension, or a G-algebra).

Gröbner bases of DD-algebras. We only consider the special case when $R = k$. From now on, let $A = k[D; \mathrm{id}, \delta][S; \sigma, 0]$ be a DD-algebra. Then, it is easy to see that $\delta = 0$ and $\sigma|_k = \mathrm{id}$. Thus $A = k[D; \mathrm{id}, 0][S; \sigma, 0]$ and $\sigma|_k = \mathrm{id}$. One can prove that the set $\mathcal{M} = \{S^\alpha D^\beta : \alpha \in \mathbb{N}^m, \beta \in \mathbb{N}^n\}$ is a k-basis of A. Let $u = S^\alpha D^\beta \in \mathcal{M}$, $\alpha = (\alpha_1, \ldots, \alpha_m) \in \mathbb{N}^m$ and $\beta = (\beta_1, \ldots, \beta_n) \in \mathbb{N}^n$. Then the *(total) degree* of u is defined as $\deg(u) = \alpha_1 + \cdots + \alpha_m + \beta_1 + \cdots + \beta_n$, and the degree of u with respect to S_i (D_j, respectively) is defined as $\deg_{S_i} = \alpha_i$ ($\deg_{D_j} = \beta_j$, respectively).

For any given well ordering on $\mathcal{M}$ and $f = c_1u_1 + \cdots + c_tu_t \in A$ ($0 \neq c_i \in k$, $u_i \in \mathcal{M}$, $1 \leq i \leq t$) with $u_1 > \cdots > u_t$, the *leading monomial* of f is denoted by $\mathrm{lm}(f) = u_1$. A *DD-monomial ordering* on $\mathcal{M}$ is a well ordering $>$ on $\mathcal{M}$ such that if $S^\alpha D^\beta > S^{\alpha'} D^{\beta'}$ and $f \in A \setminus k$, then $\mathrm{lm}(fS^\alpha D^\beta) > \mathrm{lm}(fS^{\alpha'} D^{\beta'})$. Note that DD-monomial orderings are more general than differential difference orderings defined in [5].

Let $f, g \in A$. If there exists $h \in A$ such that $f = hg$, we say that f is *right divisible* by g.

Let $>$ be a DD-monomial ordering on $\mathcal{M}$ and I be a left ideal of A. A finite set $G \subseteq A$ is called a (finite) *left Gröbner basis* of I with respect to $>$ if G satisfies: (i) G generates I as a left ideal of A; and (ii) For any $0 \neq f \in I$, there exists $g \in G$ such that $\mathrm{lm}(f)$ is right divisible by $\mathrm{lm}(g)$.

Similarly as in [5], we can define reductions and S-polynomials. Then the reduction algorithm and the left Gröbner basis algorithm still work under a DD-monomial ordering. We have

Theorem 1 *Let $G \subseteq A$ be a finite set and I be the left ideal of A generated by G. Then G is a left Gröbner basis of I if and only if* $\mathrm{Spoly}(g_1, g_2) \to_G 0$ *for any $g_1, g_2 \in G$.*

It can be proved that the Hilbert basis theorem is valid for DD-algebras: every left ideal of A is finitely generated. Thus we have

Theorem 2 *Every left ideal of a DD-algebra $k[D; \mathrm{id}, \delta][S; \sigma, 0]$ has a (finite) left Gröbner basis.*

Gelfand-Kirillov dimension of cyclic A-modules. For convenience, let $x_i = S_i, x_{m+j} = D_j$ for $1 \leq i \leq m, 1 \leq j \leq n$ and let $l = m + n$. Denote $X^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_l^{\alpha_l}$ for $\alpha = (\alpha_1, \ldots, \alpha_l) \in \mathbb{N}^l$. Then $\mathcal{M} = \{X^\alpha : \alpha \in \mathbb{N}^l\}$. For $u = X^\alpha \in \mathcal{M}$ and $p \in \mathbb{N}$, define $\mathrm{top}_p(u) = \{i : 1 \leq i \leq l, \alpha_i \geq p\}$ and $\mathrm{sh}_p(u) = X^\beta$, where $\beta_i = \min\{p, \alpha_i\}, 1 \leq i \leq l$.

Then we have the following theorem which computes the Gelfand-Kirillov dimension of a cyclic DD-module.

Theorem 3 *Let I be a left ideal of A and G be a left Gröbner basis of I with respect to a total degree DD-monomial ordering. Set $p = \max\{\deg_{x_i}(\mathrm{lm}(g)) : g \in G, 1 \leq i \leq l\}$. Then*

$$\mathrm{GKdim}(M) = \max\{|\,\mathrm{top}_p(u)| : \mathrm{sh}_p(u) = u\}.$$

References

[1] A. D. Bell and K. R. Goodearl. Uniform rank over differential operator rings and Poincaré-Birkhoff-Witt extensions. *Pacific J. Math*, 131(1):13–37, 1988.

[2] P. E. Hydon. Symmetries and first integrals of ordinary difference equations. *Proceedings of the Royal Society of London (series A)*, 456:2835–2855, 2000.

[3] A. Kandri-Rody and V. Weispfenning. Non-commutative Gröbner bases in algebras of solvable type. *Journal of Symbolic Computation*, 9(1):1–26, 1990.

[4] V. Levandovskyy. *Non-commutative Computer Algebra for polynomial algebras: Gröbner bases, applications and implementation.* PhD thesis, University of Kaiserslautern, 2005.

[5] E. L. Mansfield and A. Szanto. Elimination theory for differential difference polynomials. In *Proceedings of the 2003 international symposium on symbolic and algebraic computation*, pages 191–198. ACM, 2003.

[6] Meng Zhou and Franz Winkler. Gröbner bases in difference-differential modules. In *Proceedings of the 2006 international symposium on Symbolic and algebraic computation*, pages 353–360. ACM, 2006.

Newton-like Iteration for Determinantal Systems and Structured Low Rank Approximation

Éric Schost and Pierre-Jean Spaenlehauer
Western University, Department of Computer Science
London, Ontario, Canada
eschost@uwo.ca, pierre-jean.spaenlehauer@m4x.org

Problem statement. Let $\mathcal{M}_{p,q}(\mathbb{R})$ be the space of $p \times q$ matrices with real entries, $r \in \mathbb{N}$ be an integer, $V_r \subset \mathcal{M}_{p,q}(\mathbb{R})$ be the determinantal variety of matrices of rank at most r and E be a linear (or affine) subspace of $\mathcal{M}_{p,q}(\mathbb{R})$ (*e.g.* Toeplitz, Hankel, Sylvester matrices). Given a matrix $M \in E$, the goal is to compute a close matrix in $E \cap V_r$. More precisely, we want a numerical algorithm computing a function $\varphi : E \longrightarrow E$ such that, if M is close enough to $E \cap V_r$, then the sequence defined by $M_0 = M$, $M_{i+1} = \varphi(M_i)$ converges quadratically towards a matrix $M_\infty \in E \cap V_r$. As shown in [5], this problem which is also known as *Structured Low-Rank Approximation* (SLRA) is central in data fitting or in numerical analysis. It is also underlying classical symbolic-numeric problems.

Main results. We propose a Newton-like algorithm (`NewtonSLRA`) which answers the above specification and appears to converge quadratically. The main principle of this algorithm is close to Cadzow's algorithm [1] which proceeds by a sequence of Singular Value Decompositions (SVD) and orthogonal projections on E. However, we choose a direction of projection which is tangent to the determinantal variety in order to ensure quadratic convergence. Each iteration of the algorithm `NewtonSLRA` computes a function $\varphi(M)$ in three main steps: (1) compute a rank r approximation $\widetilde{M}$ of M; (2) from the left and right kernels of $\widetilde{M}$, compute a set of generators of the tangent space $T_{\widetilde{M}}V_r$; (3) compute the point in $E \cap T_{\widetilde{M}}V_r$ which minimizes the distance to $\widetilde{M}$ (this is achieved by solving a linear least squares problem). Computing the best rank r approximation with respect to the Frobenius norm is achieved by the SVD. It also provides an orthonormal basis (for the scalar product $\langle M_1, M_2 \rangle = \mathtt{tr}(M_1 \cdot {}^T M_2)$) of the normal space $N_{\widetilde{M}}V_r = \mathrm{Ker}_L(\widetilde{M}) \otimes \mathrm{Ker}_R(\widetilde{M})$ which is used for computing the projection on E. The most expensive step is the SVD which is achieved in $O(pq \min(p,q))$ operations in fixed precision. The main theoretical result lies in the following theorem which ensures the local quadratic convergence towards a matrix $M_\infty \in V_r \cap E$ near the optimal solution, under conditions on the dimensions of $\dim(E)$ and $\dim(V_r)$. To the best of our knowledge, this is the first proof of quadratic convergence of an iterative method for the SLRA problem:

Theorem 1. *If* $\dim(E) = \dim(V_r)$ *and* $\dim(E) + \dim(V_r) > pq$*, then the algorithm* `NewtonSLRA` *computes a function* $\varphi : E \to E$ *verifying the following property: for all* $\mu > 1$ *and for all* $\hat{M} \in V_r \cap E$ *such that* V_r *and* E *verify mild transversality conditions at* $\hat{M}$*, there exists* $\epsilon > 0$ *such that for all* M_0 *with* $\|M_0 - \hat{M}\| < \epsilon$*, the sequence* $M_{i+1} = \varphi(M_i)$ *converges towards a matrix* $M_\infty \in V_r \cap E$ *and*

$$\|M_i - M_\infty\| \leq \left(\frac{1}{2}\right)^{2^i - 1} \|M_0 - M_\infty\| \quad \textit{and} \quad \|M_0 - M_\infty\| \leq \mu \|M_0 - \hat{M}\|.$$

The proof relies on tools from Smale's α-theory, slightly modified to take into account the properties of this Newton-like iteration.

Application to univariate approximate GCD. Approximate GCD computation is a symbolic-numeric example of SLRA problem: a degree condition on the GCD of univariate polynomials amounts to

a rank condition on their Sylvester matrix. In this setting, the algorithm takes as input two floating-point polynomials f, g of degrees m and n, and an integer $d \in \mathbb{N}$; it outputs three floating-point polynomials a, b, h of respective degrees $m-d$, $n-d$, d such that $\|f-ah\|^2 + \|g-bh\|^2$ is small. Here, E is the linear space of truncated Sylvester matrices (see *e.g.* [6]) and V_r is the variety of rank deficient matrices of sizes $(m+n-d+1) \times (m+n-2d+2)$. We compare in Table 1 our `Maple` implementation of `NewtonSLRA` with the `Maple` implementation of `GPGCD` [6], which is a state-of-the art algorithm dedicated to the computation of approximate GCDs. Instances are constructed by generating two random polynomials $\widetilde{f}$, $\widetilde{g}$ such that $\deg(\mathrm{GCD}(\widetilde{f},\widetilde{g})) = d$ and by adding a random error polynomial f_ϵ, g_ϵ such that the relative noise $\sqrt{\|f_\epsilon\|^2 + \|g_\epsilon\|^2}/\sqrt{\|\widetilde{f}\|^2 + \|\widetilde{g}\|^2}$ is equal to a fixed parameter ϵ. The column "perturbation" gives the relative distance between the output and the input of the algorithms. Notice that `NewtonSLRA` performs almost as well as `GPGCD`, which relies on optimization techniques to minimize the function $\|f-ah\|^2+\|g-bh\|^2$. In comparison, `NewtonSLRA` does not converge to the minimum of this function, but we see in Table 1 that the distance to the optimum is small. Also, experimental results indicate that `NewtonSLRA` converges quadratically (although $\dim(E)$ and $\dim(V_r)$ do not verify the assumptions of theorem 1), whereas `GPGCD` converges linearly (see the right part of table 1 for an example). We also tried to use directly the `QRGCD` routine from the package `SNAP` in `Maple` [3] but it failed to find an approximate GCD in our examples because of the high level of noise in the coefficients of the input polynomials.

	`NewtonSLRA`		`GPGCD`	
(m,n,d,ϵ)	time	perturbation	time	perturbation
(100, 100, 50, 0.001)	0.803s	4.838e-4	0.806s	4.742e-4
(500, 500, 250, 0.001)	37.5s	5.127e-4	45.4s	4.923e-4
(1000, 1000, 500, 0.001)	282s	5.781e-4	317s	5.155e-4
(2000, 2000, 1000, 0.0001)	1567s	5.104e-5	1161s	5.088e-5

	sizes of iteration steps	
iteration	`NewtonSLRA`	`GPGCD`
1	0.9e-1	0.9e-1
2	0.5e-3	0.5e-3
3	0.6e-8	0.2e-5
4	0.1e-17	0.8e-8
5	0.1e-36	0.4e-10

Table 1: Comparison between `GPGCD` [6] and `NewtonSLRA` for computing approximate GCDs

Other applications of SLRA in symbolic-numeric computations and future work. Several other algebraic problems are characterized by rank conditions on structured matrices, for which these techniques could lead to symbolic-numeric algorithms, *e.g.* solving bilinear systems, computing the minimal polynomial of algebraic power series or computing low degree Pade approximants. Moreover, there is still room for improvement: the most computationally-intensive step of this algorithm is the computation of the SVD, but the algorithm converges quadratically even when less precise rank-approximation techniques are used. Also, we plan to compare our method and implementation with other algorithms for SLRA (see *e.g.* [2] and references therein) and for computing approximate GCDs (see *e.g* [4], which relies on the *Structured Total Least Norm* approach). The main challenge is to extend theorem 1 by relaxing the restrictions on $\dim(E)$ and $\dim(V_r)$.

References

[1] J. A. Cadzow. Signal enhancement-a composite property mapping algorithm. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 36(1):49–62, 1988.

[2] M. T. Chu, R. E. Funderlic, and R. J. Plemmons. Structured Low Rank Approximation. *Linear algebra and its applications*, 366:157–172, 2003.

[3] R. M. Corless, S. M. Watt, and L. Zhi. QR factoring to compute the gcd of univariate approximate polynomials. *Signal Processing, IEEE Transactions on*, 52(12):3394–3402, 2004.

[4] E. Kaltofen, Z. Yang, and L. Zhi. Structured low rank approximation of a Sylvester matrix. In *Symbolic-numeric computation*, pages 69–83. Springer, 2007.

[5] I. Markovsky. Structured low-rank approximation and its applications. *Automatica*, 44(4):891–909, 2008.

[6] A. Terui. An iterative method for calculating approximate gcd of univariate polynomials. In *ISSAC 2009*, pp. 351–358.

A Verification Framework for *MiniMaple* Programs*

Muhammad Taimoor Khan and Wolfgang Schreiner
Dokotratskolleg Computational Mathematics (DK) and
Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
muhammad.khan@dk-compmath.jku.at
Wolfgang.Schreiner@risc.jku.at

In this poster, we present an overview of our ongoing work and results on the development of a verification framework for programs written in a (substantial) subset of the language of the computer algebra system Maple, which we call *MiniMaple*. The main goal here is to detect behavioral errors in such programs w.r.t. their specifications by static analysis. However, the task of the formal specification and verification of *MiniMaple* programs is complex as Maple supports various non-standard types of objects such as unevaluated expressions and also requires abstract data types to formalize computer algebra concepts and notions. To approach our goal, we have defined and formalized the syntax, semantics, type system and specification language for *MiniMaple*. For the verification, we translate an annotated *MiniMaple* program into the language Why3ML of the intermediate verification tool Why3 [1] developed at LRI, France. We generate verifi! cation conditions by the corresponding component of Why3 and then prove the correctness of these conditions by various automatic and interactive theorem provers supported by the Why3 back-end. The main test for our verification framework is the Maple package *DifferenceDifferential* [2] developed at our institute to compute bivariate difference-differential polynomials using relative Gröbner bases. All software (lexer, parser, type checker and translator) is open source and freely available from `http://www.risc.jku.at/people/mtkhan/dk10/`.

As a general overview of our verification framework, first any *MiniMaple* program is parsed to generate an abstract syntax tree (AST). Then the AST is type checked and annotated by type information and translated into a (presumably) semantically equivalent Why3ML program. From this program, Why3 generates verification conditions to be proved correct by its various back-end supported provers. All components of the framework may generate errors and information messages.

The syntax of *MiniMaple* [3] covers all the syntactic domains of Maple but supports fewer alternatives in each domain than Maple; in particular, Maple has many built-in expressions which are not supported in our language. We use the type annotations which Maple introduced for runtime checking for the purpose of the static type checking of *MiniMaple* programs; indeed we have defined a formal type system for *MiniMaple* as a decidable logic with various typing judgments. The type system requires that procedure parameters, procedure results and local variables are type annotated. However, global variables in Maple cannot be type annotated, such that values of arbitrary types can be assigned to them. To handle the correct semantics of such variables inside and outside of the body of procedures, we introduced *global* and *local* contexts. In the former, variables can be introduced by assignments and their types can change arbitrarily, while i! n the latter, variables can only be introduced by declarations and their types can only be specialized [3]. Another issue is the handling of dynamic type tests by the *MiniMaple* expression **type**(E,T). The use of a type test in a conditional may result in different type information for the same variable in different branches of the conditional; we use the type information introduced by the corresponding conditional branches to infer

*The research was funded by the Austrian Science Fund (FWF): W1214-N15, project DK10.

the possible type of a variable. We have applied the type checker to the package *DifferenceDifferential*; no crucial errors were found but some bad code parts, e.g. duplicate declaration of variables and global variables that are declared but not used.

Furthermore, we have defined a specification language for *MiniMaple* to formally describe mathematical theories (types, functions, axioms), behavior of procedures (pre- and post-conditions and other constraints), loops (invariants and termination terms) and commands (assertions). In addition to basic formulas, our specification language supports various forms of quantifiers, i.e. logical quantifiers (**forall** and **exists**), numerical quantifiers (**add**, **mul**, **max** and **min**) and sequential quantifiers (**seq**) to represent truth values, numeric values and sequences of values respectively. The language slightly extends the Maple syntax, e.g. logical quantifiers use typed variables and numerical quantifiers use logical conditions that filter values from the specified range of a variable. The language supports abstract data types to specify abstract mathematical concepts, e.g. polynomial rings. As an example, we have f! ormally specified a substantial part of the package *DifferenceDifferential*, e.g. difference-differential operators are formalized by a corresponding abstract data type.

To verify a *MiniMaple* program annotated with types and specifications, we translate this program to the language Why3ML of the intermediate verification tool Why3. We use the Why3 verification conditions generator to produce a set of verification conditions: the pre-conditions of called procedures, the post-conditions of defined procedures, the initial establishing of loop invariants, the preservation of loop invariants after every iteration and the decreasing of termination terms. We then prove their correctness by automated provers (e.g. Z3 and CVC3) and proof assistants (e.g. Coq) supported by the Why3 back-end. The wide range of proof support was one the reasons why we chose Why3, as we are, e.g., dealing with non-linear arithmetic which requires in general an interactive prover. For verification, we have defined the translation of *MiniMaple* into semantically equivalent constructs of Why3ML, e.g. the *MiniMaple* return statement is translated using t! he Why3 exception-handling mechanism, union types are translated to algebraic types and the corresponding type tests are translated using pattern matching. Using this approach, we have already verified most of the low level procedures of the package *DifferenceDifferential*, e.g. “gleicheterme” (comparing two difference-differential terms), “sigmamax” (computing a differential term with given constraints) and “ddsub” (subtraction of differential operators).

Currently, we are in the process of verifying higher level procedures with abstract (data type based) specifications: based on an example we experiment with appropriate proof strategies for such specifications using our verification framework. As a next step, a proof of the soundness of translation for selected *MiniMaple* constructs is planned. One of the reason for choosing Why3 was that it provides a formal (originally weakest precondition based, later also operational) semantics. We have correspondingly defined a formal denotational semantics of *MiniMaple* programs [3], e.g. the semantics of command execution is defined as a state relationship between pre- and post-states, i.e. the *MiniMaple* command semantics $[[C]](e)(s, s')$ states that in an environment e the execution of a command C in a pre-state s may result in a post-state s'. Based on these definitions, we plan to prove that our translation preserves the programs' semantics.

References

[1] F. Bobot and et al. Why3: Shepherd Your Herd of Provers. In *Boogie 2011: First International Workshop on Intermediate Verification Languages*, Wrocław, Poland, August 2011.

[2] Christian Dönch. Bivariate Difference-Differential Dimension Polynomials and Their Computation in Maple. Technical report, Research Institute for Symbolic Computation, University of Linz, 2009.

[3] M. T. Khan and W. Schreiner. Towards the Formal Specification and Verification of Maple Programs. In J. Jeuring and et al., editors, *Intelligent Computer Mathematics*, volume 7362 of *LNCS*, pages 231–247. Springer, 2012.

Relaxing Order Basis Computation

Pascal Giorgi and Romain Lebreton
LIRMM, CNRS-UM2 France
pascal.giorgi@lirmm.fr, romain.lebreton@lirmm.fr

The computation of an order basis (also called sigma basis in [3]) is a fundamental tool for linear algebra with polynomial coefficients. Such a computation is one of the key ingredients to provide algorithms which reduce to polynomial matrices multiplication. This has been the case for column reduction [3] or minimal nullspace basis [11] of polynomial matrix over a field. In this poster, we are interested in the application of order basis to compute minimal matrix generators of a linear matrix sequence (see [9]). In particular, we focus on the linear matrix sequence used in the Block Wiedemann algorithm [1].

As of today, the fast order basis algorithm PM-Basis from [3] suffers from two issues. In our applications, the bound σ on its degree may be pessimistic and therefore we need to use early termination. However the recursive aspect of PM-Basis is unhelpful to implement such an early termination. Also PM-Basis may require to know more coefficients of F than necessary. This can hinder the complexity when the cost of computing coefficients of the entry is dominant. This is the case for instance for the block Wiedemann algorithm which motivates this work.

Main results In this work we propose a relaxed variant of the PM-Basis algorithm. The property of relaxed algorithms is that they do not require more knowledge on the input than necessary while keeping a quasi-optimal complexity in the order σ.
We first propose an iterative variant Iterative-PM-Basis of PM-Basis which is more suited to the relaxed model and also to early termination. Then we show how to relax Iterative-PM-Basis *via* the use of a relaxed polynomial matrix multiplication algorithm. Thus we obtain our relaxed order basis computation within the complexity of PM-Basis with only an extra logarithmic factor in σ. Finally, we show the benefit of this algorithm to gain a constant factor on average on the block Wiedemann algorithm.

Order basis algorithms Let $\mathbb{K}$ be a field, $F = \sum_{i \geqslant 0} F_i x^i \in \mathbb{K}[[x]]^{m \times n}$ a matrix of power series, σ a positive integer and (F, σ) be the $\mathbb{K}[x]$-module defined by the set of $v \in \mathbb{K}[x]^{1 \times m}$ such that $vF \equiv 0 \bmod x^\sigma$. A polynomial matrix P is a (left) order basis of F of order σ and shift $\vec{s}$ if the rows of P form a basis of (F, σ) and P is $\vec{s}$-row reduced (see [10] for details). Without loss of generality we only consider in this poster the case $n = O(m)$ with a balanced shift $\vec{s}$ as in [3]. Indeed the techniques of [10] allow to reduce the general case to our particular case.

Two different algorithms presented in [3] compute an order basis P of F. The M-Basis algorithm works iteratively on the order σ to compute the order basis P. It is a lazy algorithm that costs $O(m^\omega \sigma^2)$ arithmetic operations in $\mathbb{K}$,

Algorithm 1: Iterative-PM-Basis

Input: $F \in \mathbb{K}[[x]]^{m \times n}$, $\sigma > 0$, $\vec{s} \in \mathbb{N}^m$
Output: $P \in \mathbb{K}[x]^{m \times n}$ such that P is a $\vec{s}$-row reduced order basis of (F, σ)

1: $P_0, \vec{u} :=$ M-Basis$(F \bmod x,\ 1,\ \vec{s}\ )$; $P := [P_0]$; $S := [0, \ldots, 0, F]$ with $\lceil \log_2(\sigma) \rceil$ zeros
2: **for** $k = 1$ **to** $\sigma - 1$ **do**
3: $\quad \ell := \nu_2(k)$; $\ell' := \begin{cases} \lceil \log_2(\sigma) \rceil & \text{if } k = 2^\ell \\ \nu_2(k - 2^\ell) & \text{otherwise} \end{cases}$
4: $\quad$ Update P by merging its first $\ell + 1$ elements by multiplication //Product tree of step 4)
5: $\quad S[\ell + 1] :=$ MiddleProduct$(P[1], S[\ell' + 1], 2^\ell)$ //Update of the series of step 2)
6: $\quad P_k, \vec{u} :=$ M-Basis$(S[\ell + 1] \bmod x, 1, \vec{u}\ )$ //Recursive calls on leafs of steps 1) and 3)
7: $\quad$ Insert P_k at the beginning of P
8: **return** $\prod_i P[i]$

i.e. it only requires the coefficients F_j of F for $0 \leqslant j \leqslant (i-1)$ for computing the intermediate order basis of order i. The PM-Basis algorithm uses a divide-and-conquer approach on the order σ to reduce the arithmetic complexity to $O(m^\omega \mathsf{M}(\sigma)\log(\sigma)) = O\tilde{}(m^\omega\sigma)$, where M denotes the arithmetic complexity of polynomial multiplication. Roughly speaking, the algorithm is made of four steps: 1) a recursive call to compute an order basis P_{low} of F of order $\sigma/2$, 2) an update of the problem *via* the middle product $F' := (x^{-\sigma/2}P_{\text{low}}F) \bmod x^{\sigma/2}$, 3) a recursive call to compute an order basis P_{high} of F' of order $\sigma/2$ and 4) return the order basis $P_{\text{high}}P_{\text{low}}$ of F of order σ. Step 2) implies that one may need at most twice as much coefficients of the input series than necessary to go from an intermediate order basis of order i to $i+1$.

Fast iterative order basis Let us give an iterative version of PM-Basis. Our algorithm performs exactly the same operations on matrices as PM-Basis when σ is a power of two. This iterative presentation of PM-Basis is original. Let us denote $\nu_2(k)$ the valuation in 2 of any integer k and index our lists from 1.

Relaxing order basis algorithm In algorithm PM-Basis, we have noticed that only the middle product of step 5 reads more entries of F than necessary at step k. Let us perform this step differently so that it reads at most the coefficients $F_0, \ldots, F_{k-1}$ of F at step k. This property is called a *relaxed* (or on-line) algorithm w.r.t. F.

A naive approach would be to compute a full $2n \times n$ product using a relaxed multiplication algorithm on polynomial of matrices ([2, 5, 6, 4, 8]) in time $\mathsf{R}(n) = O(\mathsf{M}(n)\log(n))$ [2]. We propose another relaxed algorithm that gains asymptotically a factor 2 compared to the full $2n \times n$ relaxed product. We decompose the relaxed middle product in a normal high product (in black) followed by a multiplication (in white and gray) relaxed w.r.t. only b using [4] in this example (see Figure 1).

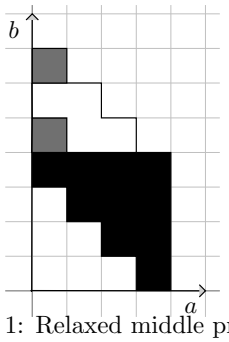


Figure 1: Relaxed middle product

Using this relaxed middle product algorithm within Iterative-PM-Basis we obtain an order basis algorithm Relaxed-PM-Basis relaxed w.r.t. F. This relaxed order basis algorithm costs $O(k^\omega \mathsf{R}(\sigma)\log(\sigma)) = O(k^\omega \mathsf{M}(\sigma)\log^2(\sigma))$.

Application to block Wiedemann algorithm Let $A \in \mathrm{GL}_N(\mathbb{K})$ with $O(N)$ non-zero elements. Block Wiedemann approach uses a minimal matrix generator of the matrix series $S = \sum_{i\in\mathbb{N}} UA^iVx^i$ for any random $U, V^T \in \mathbb{K}^{m\times N}$ in order to solve a linear system $Ax = b \in \mathbb{K}^N$. As described in [9], this matrix generator can be obtained from an order basis of $F = [S \mid I_m]^T \in \mathbb{K}[[x]]^{2m\times m}$. We can derive a bound on the maximal degree δ of this order basis using the stopping criteria of [7, Th. 4.19]. Since this bound may be loose, a constant factor in the complexity can often be saved using an early termination in the order basis algorithm.

We compare the complexity of Iterative-PM-Basis and Relaxed-PM-Basis in this setting. Computing S at precision σ costs $O(k^{\omega-1}N\sigma)$. In practice $k \ll N$ so that the cost of computing S always dominates the cost of (relaxed) order basis algorithm.

Assume that δ is uniformely distributed between 2^p+1 and 2^{p+1} for $p \in \mathbb{N}$. Iterative-PM-Basis requires the coefficients $F_0, \ldots, F_{2^{p+1}-1}$ whereas Relaxed-PM-Basis only asks for $F_0, \ldots, F_{\delta-1}$. Therefore our relaxed approach improves the dominant cost of computing F in block Wiedemann by a factor 2 at most and 4/3 on average.

References

[1] D. Coppersmith. Solving Homogeneous Linear Equation Over GF(2) via Block Wiedemann Algorithm. *Mathematics of Computation*, 62(205):333–350, 1994.

[2] M. J. Fischer and L. J. Stockmeyer. Fast on-line integer multiplication. *J. Comput. System Sci.*, 9:317–331, 1974.

[3] P. Giorgi, C.-P. Jeannerod, and G. Villard. On the complexity of polynomial matrix computations. In *ISSAC'03*, p. 135–142. ACM, 2003.

[4] J. v. d. Hoeven. Relaxed multiplication using the middle product. In *ISSAC'03*, p. 143–147, New York, 2003. ACM.

[5] J. v. d. Hoeven. New algorithms for relaxed multiplication. *J. Symbolic Comput.*, 42(8):792–802, 2007.

[6] J. v. d. Hoeven. Faster relaxed multiplication. Technical report, HAL-00687479, 2012.

[7] E. Kaltofen and G. Yuhasz. On the matrix berlekamp-massey algorithm. *ACM Trans. on Algorithms*, 2013. To appear.

[8] R. Lebreton and É. Schost. Relaxed power series multiplication using middle and short product. In preparation.

[9] W. J. Turner. *Black Box Linear Algebra with the LinBox Library.* PhD thesis, North Carolina State University, 2002.

[10] W. Zhou and G. Labahn. Efficient algorithms for order basis computation. *J. Symbolic Comput.*, 47(7):793 – 819, 2012.

[11] W. Zhou, G. Labahn, and A. Storjohann. Computing minimal nullspace bases. In *ISSAC '12*, p. 366–373. ACM, 2012.

Block Wiedemann algorithm on multicore architectures

Bastien Vialla
LIRMM, CNRS-UM2 France
bastien.vialla@lirmm.fr

Solving a linear system with large sparse matrices is a computational kernel used in a wide range of applications, *e.g.* cryptography, Gröbner basis ... Classical methods such as Gaussian elimination are not well suited because they tend to fill the matrix. In [7] Wiedemann proposed a blackbox algorithm which takes advantage of the sparsity to reduce the complexity. The main operations of this approach are sparse matrix vector products and the computation of the minimal generator of a scalar sequence. Despite a better complexity than classical methods, this algorithm is not efficient in the context of parallel computation as it needs a good repartition of the non-zero elements in the matrix. The block version of Wiedemann's algorithm proposed in [2] avoids this problem by using blocks instead of vectors. Therefore it offers parallelism outside the scope of the matrix.

Let $\mathbb{K}$ be a field, $A \in \mathcal{M}_{n\times n}(\mathbb{K})$, $U, V \in \mathcal{M}_{n\times k}(\mathbb{K})$ random matrices with $k \leq n$. We denote by γ the number of non-zero elements in A and we assume that $\gamma = O(n \log n)$. Block Wiedemann algorithm follows three steps:

1. Compute the first $O(\frac{2n}{k})$ elements of $S = [U^T A^i V]_{i\in\mathbb{N}}$ using $O(n\gamma + n^2 k^{\omega-2})$ operations in $\mathbb{K}$.
2. Find the minimal matrix polynomial generator of the sequence S using $\tilde{O}(k^{\omega-1}n)$ operations in $\mathbb{K}$.
3. Compute the solution using the polynomial found in step 2 using $O(n\gamma + n^2)$ operations in $\mathbb{K}$.

In practice the cost of the first step is dominant, therefore its parallelization is crucial. The capacity to parallelize the first step heavily relies on the dimension k of the blocks.

A classical approach is to take a block size equal to the number of cores. The parallel complexity of the first step becomes $O(\frac{n\gamma}{k} + n^2)$ operations in $\mathbb{K}$. We notice that the $O(n^2)$ part does not benefit from parallelism. In order to take advantage of parallelism everywhere in step 1, we must proceed otherwise.

Our approach We naturally extend the use of sparse blocks proposed by Eberly et al. in [3] to our context of block Wiedemann algorithm. Hence, instead of using random dense block for U, we use blocks of the form

$$U = \begin{bmatrix}\delta_1 I_k & \cdots & \delta_s I_k & \delta_{s+1} I'\end{bmatrix}^T \in \mathcal{M}_{n\times k}(\mathbb{K})$$

where $s = \lfloor n/k \rfloor$, $\delta_1, \cdots, \delta_{s+1} \in \mathbb{K}$ chosen at random, I_k the identity matrix of size k, and $I' = Diag(1, \cdots, 1) \in \mathcal{M}_{k\times r}(\mathbb{K})$ the matrix with only ones on the diagonal with $r = n \bmod k$. Using these new block projections, the sequential complexity of step 1 drops down to $O(n\gamma + n^2)$ operations in $\mathbb{K}$, eliminating the influence of the block size. In this work we study how these new block projections perform in practice and we show that they improve the performance of the first step of block Wiedemann algorithm.

Implementation and Benchmarks For the benchmarks, we have in mind matrices arising from NFS algorithm [6], which are very sparse. As γ is cheaper, the part of step 1 of complexity $O(n^2)$ has more importance. Therefore the block size has more influence on the sequence computation as γ is cheaper. In this case, we use a sparse matrix of size $10^5 \times 10^5$ over $\mathbb{F}_{65537}$ with ~ 15 non-zero elements per row uniformly dispatched.

The parallel complexity of step 1 using sparse blocks with k cores becomes $O(\frac{n\gamma+n^2}{k})$, hence offering perfect parallelism. So we want to see the influence of the block size on the computation of step 1.

First, we determine the most efficient block size depending on the number of cores. Let c be the number of cores, we benchmark the computation of the sequence starting with blocks of size c to $128c$ on 12 cores. As expected by the complexity analysis, a block size of c offers better performance for dense blocks. For sparse projections the theoretical study shows no influence of the blocks size. In practice we observe that a block size of $\simeq 32c$ is better,

which could be caused by memory management issues. However, this sparse block size is related to the number of non-zero elements of the matrix, so these values stand just for our test matrix. Despite their good complexity, sparse blocks have two flaws impacting the performance. The choice of matrix representation is important: first we choose to store blocks in column major representation to avoid concurrent writing, as suggested in [1]. Secondly, sparse blocks induce cache defaults as their size increase with the number of cores. To circumvent this problem, we permute block elements to obtain a cache friendly sparse blocks following ideas from [5]. For our tests we use an NUMA with four intel XEON E4620 with 8 cores at 2.2Ghz and 384GB of RAM. To obtain good performance on an NUMA, we design an hybrid MPI/tbb implementation that create one MPI process by node which use tbb to compute a part of the sequence. Each nodes own a copy of the sparse matrix and the block is split by column over the nodes, the results are gather at the end of the computation. All the libraries used are in the latest version from their svn directory. In table 1 we compare dense block which used LinBox's dense blocks implementation and our implementation using sparse blocks. For computing of dense block, LinBox relies on a BLAS library, in this case we use OpenBLAS wich is well optimized for intel XEON. The timings are in seconds and in parenthesis we indicate the block size used.

	Dense blocks (LinBox)		Sparse blocks	
	time in s	speed-up	time in s	speed-up
1 core	2205(1)	1	2165(32)	1
8 cores	540(8)	4	308(256)	7
16 cores	623(16)	3,5	154(512)	14
24 cores	798(24)	2,7	102(768)	21,2
32 cores	960(32)	2,2	77(1024)	28,1

Table 1: Times of sequence computation.

The time for dense blocks on one core is just for benchmark purposes. As predicted, sparse blocks perform better than dense blocks. However, the reasons that LinBox implementation does not perform well are that LinBox use an external library to compute dense block which as to create and destroy is own pool of threads for each computed element. Secondly, the LinBox implementation is not designed for a NUMA architecture as all the data is store in the first node memory.

This is a first step in an efficient implementation of block Wiedemann algorithm on multicore architectures. The next step will be an efficient implementation of $\sigma-$basis [4].

References

[1] Brice Boyer, Jean-Guillaume Dumas, and Pascal Giorgi. Exact Sparse Matrix-Vector Multiplication on GPUs and Multicore Architectures. *Proc. PASCO'10: Parallel Symbolic Computation*, 2010.

[2] Don Coppersmith. Solving Homogeneous Linear Equation Over GF(2) via Block Wiedemann Algorithm. *Mathematics of Computation*, 62(205):333–350, 1994.

[3] Wayne Eberly, Mark Giesbrecht, Pascal Giorgi, Arne Storjohann, and Gilles Villard. Faster inversion and other black box matrix computations using efficient block projections. *Proceedings of the 2007 international symposium on Symbolic and algebraic computation - ISSAC '07*, 3(1):143, 2007.

[4] Pascal Giorgi, Claude-Pierre Jeannerod, and Gilles Villard. On the complexity of polynomial matrix computations. *Proceedings of the 2003 international symposium on Symbolic and algebraic computation - ISSAC '03*, pages 135–142, 2003.

[5] Sardar A. Haque, Shahadat Hossain, and M. Moreno Maza. Cache friendly sparse matrix-vector multiplication. *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation (PASCO'10)*, pages 175–176, 2010.

[6] A. K. Lenstra and H. W. Lenstra. *The development of the number field sieve*. Springer-Verlag, 1993.

[7] D. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory*, 32(1):54–62, January 1986.

Recursive Sparse Interpolation

Andrew Arnold Mark Giesbrecht
Symbolic Computation Group
University of Waterloo, Canada
{a4arnold,mwg}@uwaterloo.ca

Dan Roche
United States Naval Academy, USA
roche@usna.edu

We consider the problem of interpolating a sparse univariate polynomial f over an arbitrary ring, given by a straight-line program. In this problem we are given a straight-line program that computes f, as well as bounds D and T on the degree and sparsity (i.e., the number of nonzero terms) of f respectively. We build on ideas developed in Garg and Schost (2009) and Giesbrecht and Roche (2011) towards algorithms for this specific problem. We present a Monte Carlo algorithm that improves on the best previously-known algorithm for this specific problem by a factor (softly) on the order of $T/\log D$. Thus this new algorithm is favourable for "moderate" values of T.

Our algorithm is recursive. At a recursive step of the algorithm we have a straight-line program for f, an approximation f^* of f, and respective bounds T and D on the sparsity and degree of the difference $g = f - f^*$. We initialize f^* to zero. We will construct an approximation f^{**} to g such that, with high probability, $g - f^{**}$ has at most $T/2$ terms. We then recurse with $f^* + f^{**}$ as our refined approximation for f.

The algorithms in Garg and Schost (2009) and Giesbrecht and Roche (2011), as well as the algorithm we will present, interpolate f by using its straight-line program to evaluate f at a symbolic k-th root of unity, for appropriate choices of k. This effectively gives the image $f \bmod (z^k - 1)$. We call such an evaluation a *probe* of degree k. The cost of a degree-k probe to a length-L straight-line program is quasi-linear in kL. We use the number of probes, multiplied by a bound on the probe degree, as a rough measure of the cost of an interpolation algorithm.

The image $f \bmod (z^k - 1)$ in practise gives a large amount of useful information about the polynomial f. Namely, a term cz^e of f will appear as $cz^{e \bmod k}$ in the image $f \bmod (z^k - 1)$, so the image should give us f's vector of exponents modulo k. However, there are potential obstacles. We may not be able to match images of the same term in multiple images of f. In addition, terms can *collide* modulo $z^k - 1$ if they have the same degree modulo k. Collisions are problematic because it is difficult to detect if a term in an image $f \bmod (z^k - 1)$ is in fact the image of a sum of colliding terms. Alternatively, colliding terms may sum to zero modulo $z^k - 1$, which also may be difficult to detect.

Previous Las Vegas interpolation algorithms require a "good" prime, a prime p for which the terms of f remain distinct modulo $z^p - 1$. If p is a good prime, $f \bmod (z^p - 1)$ has the same number of terms as f. Thus, once we have a good prime with high probability, we can detect the presence of collisions in other images of f. In order to guarantee one can find such a prime with high probability, one chooses primes at random on the order of $T^2 \log D$ as probe degrees.

In order to reduce this probe degree, we relax the condition that p separates all the terms of the difference g. We instead look for an "ok" prime: a prime which separates *most* of the terms of g. This allows instead to search over primes p of size $\mathcal{O}(T \log D)$.

Once we have an "ok" prime, we make probes of degree pq_i for a set of co-prime q_i, each of size $\mathcal{O}(\log D)$. Our probe degree thus becomes $\mathcal{O}(T \log^2 D)$. If a term of g does not collide with another term modulo $z^p - 1$ then it will not collide modulo $(z^{pq_i} - 1)$. These probes will allow us to construct a polynomial f^{**} containing the non-colliding terms of g, plus potentially a small proportion of deceptive terms: terms

constructed from garbage information due to collisions in the images $f \bmod (z^{pq_i} - 1)$. Fortunately, if p is an ok prime we can give an upper bound on the number of such deceptive terms that can appear in f^{**}.

After we construct f^{**} we then recursively interpolate the new difference $g - f^{**}$, with a new sparsity bound $T/2$. We continue in this fashion $\lfloor \log T \rfloor + 1$ times until the sparsity bound reaches 0. An advantage of the recursive nature of the algorithm is that, when we reach a threshold where $\log D$ begins to dominate T, we can plug in a better-suited algorithm to interpolate what remains.

References

Sanchit Garg and Éric Schost. Interpolation of polynomials given by straight-line programs. *Theor. Comput. Sci.*, 410(27-29):2659–2662, June 2009. ISSN 0304-3975. doi: 10.1016/j.tcs.2009.03.030. URL `http://dx.doi.org/10.1016/j.tcs.2009.03.030`.

M. Giesbrecht and D.S. Roche. Diversification improves interpolation. *ISSAC '11*, pages 123–130, 2011. doi: 10.1145/1993886.1993909. URL `http://doi.acm.org/10.1145/1993886.1993909`.

Towards Parallel General-Size Library Generation for Polynomial Multiplication

Lingchuan Meng and Jeremy Johnson
Department of Computer Science
Drexel University
Philadelphia, PA 19104
lm433@drexel.edu,jjohnson@drexel.edu

Fast Fourier Transforms (FFTs) are at the core of many operations in scientific computing. In computer algebra, FFTs are used for *fast polynomial and integer arithmetic and other modular methods*. FFT-based polynomial multiplication outperforms multiplication based on classical and Karatsuba-based algorithms. Computer algebra libraries, such as `modpn` [3], provide hand-optimized low-level routines implementing fast algorithms for multivariate polynomial computations over finite fields, in support of higher-level code. Such libraries do not fully utilize the underlying hardware, and in order to take advantage of platform-dependent optimizations, automated performance tuning that supports general input sizes should be incorporated.

Recently, we extended [2] the use of Spiral from fixed-size code generation to general input size library generation to produce a modular FFT [1] library. By incorporating and extending the new library generation mechanism in Spiral [4], the generated library provides similar speedup as the fixed-size code, which is an order of magnitude faster over the original implementations in `modpn`, and allows arbitrary input sizes. Additional parallelism exploiting multi-core architecture leading to further speedup also has been implemented. This addition required adding new rules and a new transform definition and parameterization in the library generation framework in order to generate *recursive function closure* in the resulting library. The backend was also extended to enable the generation of scalar and vectorized code for modular arithmetic.

Let the n-point modular DFT matrix be $\mathbf{ModDFT}_{n,p,\omega} = \left[\omega_n^{k\ell}\right]_{0\le k,\ell<n}$, where ω_n is a primitive nth root of unity in $\mathbb{Z}_p$. Let $n = rs$, then the divide and conquer step in the Cooley-Tukey algorithm can be represented as the parameterized matrix factorization:

$$\mathbf{ModDFT}_{n,p,\omega} = (\mathbf{ModDFT}_{r,p,\omega_r} \otimes \mathrm{I}_s)\, \mathrm{T}^n_s (\mathrm{I}_r \otimes \mathbf{ModDFT}_{s,p,\omega_s})\, \mathrm{L}^n_r,$$

where T^n_s is a diagonal matrix containing *twiddle factors*; the *stride permutation* matrix L^n_r permutes the input vector as $is + j \mapsto jr + i, 0 \le i < r, 0 \le j < s$; I_s is the $s \times s$ identity matrix; and the *tensor product* is defined as $A \otimes B = [a_{k,l}B]\,, A = [a_{k,l}]\,.$

The tensor product serves as the key construct in Spiral and its many fast algorithms, in that it captures loops, data independence, and parallelism concisely. For instance, Fig. 1 shows that it produces substructures that can be interpreted as vector and parallel operations. Furthermore, the formulae can be transformed to adapt to a given vector length and number of cores; and permutations can be manipulated to obtain desired data access patterns. *Rewriting systems* and *hardware tags* have been developed in Spiral to fully exploit two levels of parallelism: *vector parallelism* and *thread parallelism*.

We report experimental data comparing the performance of hand-optimized FFTs from the `modpn` library, fixed-size FFTs and general size parallel FFT library generated by Spiral. Performance is reported in Gops (giga-ops) or billions of operations per second (higher is better). As shown in Fig. 2, all Spiral generated codes are faster than the hand-optimized implementation in `modpn` by an order of magnitude. The

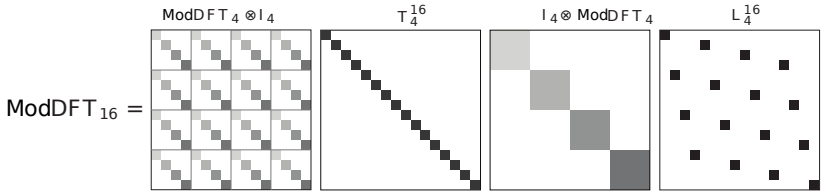


Figure 1: Representation of the matrix factorization based on the Cooley-Tukey algorithm. Shades of gray represent values that belong to the same tensor substructure

performance of general size library's scalar and vector codes are within 81% to 91% of that of corresponding fixed-size codes. For large sizes, the library code is up to 1.5 time faster than the fixed-size code, due to the use of thread level parallelism.

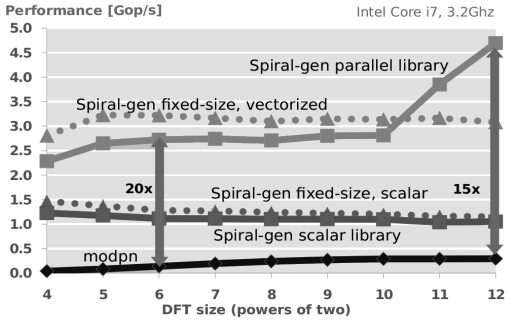


Figure 2: Performance comparison

To eventually generate an optimized parallel library for polynomial multiplication, we are developing additional algorithms for modular FFT, including Prime-factor algorithm and Rader's algorithm. We have also developed a Cooley-Tukey type algorithm for the Truncated Fourier Transform and its inverse (TFT/ITFT) for non-contiguous and non-power-of-two input/output. We have proved block symmetry of the ITFT matrices and derived direct generation formulae that can be used during library generation. Convolutions by definition and the Convolution Theorem have also been implemented in Spiral, whose performance relies on the auto-tuned underlying transforms, such as modular DFT and TFT, and the exploration of hybrid algorithms and automatic tuning of threshold parameters.

References

[1] L. Meng, J. Johnson. Automatic Parallel Library Generation for General-Size Modular FFT Algorithms. To appear in *Proc. of the CASC 2013*, 2013

[2] L. Meng, J. Johnson, F. Franchetti, Y. Voronenko,M. Moreno Maza and Y. Xie. Spiral-Generated Modular FFT Algorithms. In *Proc. of PASCO 2010*, p. 169–170, 2010.

[3] X. Li and M. Moreno Maza: Efficient implementation of polynomial arithmetic in a multiple-level programming environment. In *Proc. Intl. Congress of Mathematical Software*, p. 12–23, Springer, 2006.

[4] Y. Voronenko. Library Generation for Linear Transforms. PhD. thesis, Electrical and Computer Engineering, Carnegie Mellon University, 2008

A Parallel Algorithm to Compute the Greatest Common Divisor of Sparse Multivariate Polynomials

Jiaxiong Hu and Michael Monagan
Department of Mathematics, Simon Fraser University
Burnaby, Canada, V5A 1S6
jha107@sfu.ca and mmonagan@cecm.sfu.ca

Extended Abstract

Efficient algorithms for computing greatest common divisors (GCD) of multivariate polynomials have been developed over the last 40 years. Many of the general purpose computer algebra systems are using either Zippel's GCD Algorithm [5] or the EEZ-GCD [4] Algorithm or both. Both algorithms sequentially interpolate variables one at a time which limits parallel speedup. Since multi-core processors are now widely available, parallel algorithms are desirable. In this poster, we present a first multivariate GCD computation algorithm over $\mathbb{Z}$ which is based on the Ben-Or/Tiwari interpolation [1]. By using Ben-Or/Tiwari interpolation, we reduce the number of points needed to interpolate the GCD and improve parallelism.

Our algorithm considers multivariate GCD problems with at least three variables. The structure of the algorithm is similar to Zippel's GCD Algorithm except the way we determine the first modular image which determines all the monomials. Once this correct *form* is obtained with those monomials, we use Zippel's sparse interpolation with this form to compute more modular images and apply Chinese remaindering to reconstruct the true GCD over $\mathbb{Z}$.

Our algorithm determines the first modular image as follows: Suppose $a,\ b \in \mathbb{Z}[x_1, \ldots, x_n]$ are the input polynomials and let

$$g = \gcd(a, b) = \sum_{i=1}^{l} c_i M_i(x_1, x_2)$$

where l is the number of terms of $g(x_1, x_2)$ and M_i is the ith monomial of $g(x_1, x_2)$ and $c_i \in \mathbb{Z}[x_3, ..., x_n]$ is the ith coefficient of $g(x_1, x_2)$. The algorithm projects a and b down to bivariate polynomials by evaluating $\{x_3, \ldots, x_n\}$ at specific point $\{e_3^k, \ldots, e_n^k\}$ which satisfies the requirement of the Ben-Or/Tiwari interpolation. Then we compute bivariate

$$g_k = \gcd(a(x_1, x_2, e_3^k, \ldots, e_n^k), b(x_1, x_2, e_3^k, \ldots, e_n^k)) \in \mathbb{Z}_p[x_1, x_2],$$

where p is a carefully chosen prime. We redo this for $k = 0, 1, 2, \ldots, m$ until m is large enough. Now all bivariate GCDs should have the same monomials but different coefficients. For each monomial $M_i(x_1, x_2)$ in the g_k, we form an integer sequence by collecting M_i's coefficient in g_k $(0 \le k \le m)$. Then the Ben-Or/Tiwari algorithm is applied to this sequence to interpolate the coefficient $c_i \in \mathbb{Z}_p[x_3, \ldots, x_n]$. For this to work we require $m \ge 2t$ where $t = \max_{i=1}^{l}(\#\text{ terms } c_i)$. Obviously all polynomial coefficients $c_i(x_3, ..., x_n)$ can be recovered in parallel. Moreover, the bivariate GCDs can be computed in parallel as well. In general, this approach is easy to parallelize.

Compared with Zippel's algorithm, our algorithm uses fewer evaluation points – $O(t)$ instead of $O((n-2)dt)$ and fewer trial divisions – $O(1)$ instead of $O(n)$. One disadvantage of our algorithm is that we do not know t. We must try $t = 2, 4, 8, 16, \ldots$ stopping when we have redundancy.

A problem with the original Ben-Or/Tiwari algorithm is the intermediate expression swell that occurs using $e_3^k, e_3^k, e_4^k, ... = 2^k, 3^k, 5^k, ...$ and computing over $\mathbb{Q}$. A modular version of the algorithm was first developed by Kaltofen, Lakshman and Wiley in [3]. Their algorithm uses a small prime q with a lifting technique to determine the monomials in the c_i. One lifts until $q^k > p_{n-2}^d$ where p_n denotes the n'th prime and $d \geq \max(\deg c_i)$. Instead, we adapt Giesbrecht, Labahn and Lee's method in [2]. We construct a smooth prime p so that we can efficiently compute discrete logarithms in $\mathbb{Z}_p$. The prime p is slightly larger than $\prod_{i=3}^n d_i$ where $d_i = \deg_{x_i} g$ thus of size $O(n \log d)$. To determine the d_i accurately we compute one univariate image of g in each variable (in parallel).

A further problem is that all underlying bivariate GCDs are monic over $\mathbb{Z}_p$. The leading coefficient of the true GCD is required to scale all bivariate GCDs consistently. We use Wang's leading coefficient algorithm [4] to solve this problem. We compute and factor the gcd h of the leading coefficients of $a, b \in \mathbb{Z}[x_3, \ldots, x_n][x_1, x_2]$. This creates another sequential step in our algorithm. This is the main reason why we reduce to bivariate GCDs instead of univariate – we likely reduce the size of h. We also likely reduce t and hence the number of bivariate GCDs needed. If $a(x_1, x_2)$ and $b(x_1, x_2)$ are dense (which they often are in practice) we lose nothing by doing this.

We have implemented our algorithm in Maple. For most large problems, it outperforms Maple's default multivariate GCD procedure, which is a Zippel based algorithm and almost entirely coded in C. For example, for input polynomials having 5 variables and 500 terms, our algorithm is almost 2 times faster than Maple's default procedure; with input polynomials having 40 variables and 4000 terms, our algorithm is almost 20 times faster. We have not yet attempted a parallel implmentation but plan to do so using Cilk. We expect that such an implementation will be much faster.

References

[1] M. BEN-Or, P. Tiwari: A deterministic algorithm for sparse multivariate polynomial interpolate. *Proc. 20th annual ACM Symp Theory Comp*, 1988, 301–309.

[2] M. Giesbrecht, G. Labahn, W-S. Lee: Symbolic-numeric sparse interpolation of multivariate polynomials. *ISSAC'06*, 2006.

[3] E. Kaltofen, Y.N. Lakshman, J-M. Wiley: Modular rational sparse multivariate polynomial interpolation. *Watanabe and Nagata*, 1990, 135–139.

[4] P. Wang: The EEZ-GCD Algorithm. *SIGSAM Bulletin*, 14, 1980, 50–60.

[5] R. E. Zippel: Probabilistic algorithms for sparse polynomials. *EUROSAM '79*, Springer-Verlag LNCS, 2, 1979, 216–226.

Calculating Approximate GCD of Multiple Univariate Polynomials using Approximate Syzygies

Akira Terui
Faculty of Pure and Applied Sciences, University of Tsukuba
Tsukuba, 305-8571, Japan
terui@math.tsukuba.ac.jp, http://researchmap.jp/aterui/

For given n univariate polynomials with $n \geq 3$, we present a Symbolic-Numeric method for calculating approximate greatest common divisor (GCD) of them by calculating approximate Syzygies. This kind of GCD calculation can be used in application such as blind image deconvolution [1]. In such a case, it is especially effective when we try to restore the original image from several number of blurred images.

In our previous research, we have developed a method for calculating approximate GCD, called *GPGCD* [4]. Furthermore, we have extended the original method for n polynomial inputs [3] based on Rupprecht's first algorithm [2, Sect. 4]. However, this method is inefficient for large number and/or degree of input polynomials because, in such cases, the dimension of a generalized Sylvester matrix becomes large and sparse. While Rupprecht's second algorithm [2, Sect. 5] seems more efficient by using Syzygies with another generalization of Sylvester matrix whose dimension is much smaller than those used in the first algorithm, as for our GPGCD method, we have difficulty applying the method directly (we will explain the reason in detail below). We present a method to overcome the difficulty.

For $i = 1, \ldots, n$, let $P_i(x)$ be real univariate polynomial of degree $d_1 \leq \cdots \leq d_n$, respectively, with $d_1 > 0$, given as $P_i(x) = p_{d_i}^{(i)} x^{d_i} + \cdots p_1^{(i)} x + p_0^{(i)}$. At first we assume that $P_1, \ldots, P_n$ have a GCD. Let $H = \gcd(P_1, \ldots, P_n)$ and $d = \deg(H)$ with $d \leq d_1$.

For a real univariate polynomial $P(x)$ represented as $P(x) = p_n x^n + \cdots + p_0 x^0$, let $C_k(P)$ be a real $(n+k, k+1)$ matrix (called "convolution matrix") defined as $C_k(P) = \left({}^t(p_n, \ldots, p_0, 0, \ldots, 0), {}^t(0, p_n, \ldots, p_0, 0, \ldots, 0), \ldots, {}^t(0, \ldots, 0, p_n, \ldots, p_0)\right)$ and let $\boldsymbol{p}$ be the coefficient vector of $P(x)$ defined as $\boldsymbol{p} = (p_n, \ldots, p_0)$, and vice versa.

As a generalized Sylvester matrix, we use the second definition by Rupprecht [2, Sect. 5]. For $k > d_1$, define the k-th Sylvester matrix of $P_1, \ldots, P_n$ as $N_k(P_1, \ldots, P_n) = \begin{pmatrix} C_{k-d_1}(P_1) & C_{k-d_2}(P_2) & \cdots & C_{k-d_n}(P_n) \end{pmatrix}$, where $C_{k-d_i}(P_i)$ has empty element for $k < d_i$.

If a vector $\boldsymbol{v} = {}^t\begin{pmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \cdots & \boldsymbol{r}_n \end{pmatrix}$ with $\dim(\boldsymbol{r}_i) = k - d_i + 1$ satisfies $N_k \boldsymbol{v} = \boldsymbol{0}$, then we see that the polynomials $R_1, \ldots, R_n$ whose coefficient vectors are $\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_n$, respectively, satisfy $R_1 P_1 + \cdots + R_n P_n = 0$. In such a case, we call a tuple of polynomials $(R_1, \ldots, R_n)$ a Syzygy of $P_1, \ldots, P_n$ of degree k.

In Rupprecht's second method, we first calculate Syzygies of $P_1, \ldots, P_n$, then calculate cofactors of $P_1, \ldots, P_n$ by using calculated Syzygies, as follows.

1. Calculate $n-1$ "independent" (as elements in a module over polynomial ring R[x]) Syzygies which satisfy the following condition on the degrees. For $j = 1, \ldots, n-1$, let $R_j = (U_1^{(j)}, U_2^{(j)}, \ldots, U_n^{(j)})$ be a Syzygy of $P_1, \ldots, P_n$ of degree r_j. Then, we have

$$d = d_1 + \cdots + d_n - (r_1 + \cdots + r_{n-1}) \tag{1}$$

   [2, Lemma 5.3]. With numerical computation on coefficients, we calculate a Syzygy by the Singular Value Decomposition (SVD) on Sylvester matrix N_k by increasing the degree k by 1 from the initial value d_1, until we obtain $n-1$ Syzygies satisfying condition (1).

2. For calculated Syzygies $R_j = (U_1^{(j)}, U_2^{(j)}, \ldots, U_n^{(j)})$, $j = 1, \ldots, n-1$, define a matrix $U = (u_{ij})$ as $u_{ij} = U_j^{(i)}$, and let Δ_i be the minor of U by deleting the i-th column (note that we must define U satisfying that $\Delta_i \neq 0$ for all i). Then, Δ_i is the cofactor of P_i satisfying $P_i = H \cdot \Delta_i$ [2, Lemma 5.2]. Thus, by calculating U and Δ_i, we obtain desired GCD H.

In our GPGCD method, we accept polynomials $P_1, \ldots, P_n$ that are pairwise relatively prime in general, then find "perturbation terms" ΔP_i, $i = 1, \ldots, n$, satisfying that "perturbed polynomials" $\tilde{P}_i = P_i + \Delta P_i$ have a nontrivial GCD H. With the original method by Rupprecht, we may encounter the following issue in Step 1. In our GPGCD method, we set up constrained optimization problem with constraints on the coefficients in input polynomials and their Syzygies that need coefficients in *all* input polynomials at once. On the other hand, Step 1 in the above may not involve *all* input polynomials from the beginning step(s), thus it is not clear if we can calculate appropriate perturbed terms incrementally to make all the perturbed polynomials satisfy the Syzygy relations in the final phase. Therefore, we modify the method so that we use Syzygy relations that involve *all* input polynomials from the beginning step, as follows.

1. Let l be greater than or equal to d_n satisfying (1). For given polynomials $P_1, \ldots, P_n$, calculate perturbed polynomials $\tilde{P}_1, \ldots, \tilde{P}_n$ along with Syzygies $R_j = (U_1^{(j)}, U_2^{(j)}, \ldots, U_n^{(j)})$ of degree l satisfying $U_1^{(j)}\tilde{P}_1 + \cdots + U_n^{(j)}\tilde{P}_n = 0$, as follows. Let $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m$ be the right singular vectors of $N_l(P_1, \ldots, P_n)$ calculated with the SVD. By optimization method (in our case we use so-called the modified Newton method; see our literature [4] for reference), we obtain $\tilde{P}_1, \ldots, \tilde{P}_n$ by perturbing coefficients in $P_1, \ldots, P_n$, and $\tilde{\boldsymbol{v}}_1, \ldots, \tilde{\boldsymbol{v}}_m$ by perturbing $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m$, respectively, satisfying $N_l(\tilde{P}_1, \ldots, \tilde{P}_n)\tilde{\boldsymbol{v}}_j = \boldsymbol{0}$. From vector $\tilde{\boldsymbol{v}}_j = \begin{pmatrix} \boldsymbol{r}_1^{(j)} & \boldsymbol{r}_2^{(j)} & \cdots & \boldsymbol{r}_n^{(j)} \end{pmatrix}$, we extract coefficients of a Syzygy $R_j = (U_1^{(j)}, U_2^{(j)}, \ldots, U_n^{(j)})$.

2. Using Syzygies R_j calculated in the above step, select and/or calculate Syzygies of appropriate degree satisfying (1) to make up matrix U with the following strategies.

   (a) If we need to calculate Syzygies of degree k smaller than l, make appropriate linear combination of the right singular vectors $\tilde{\boldsymbol{v}}_1, \ldots, \tilde{\boldsymbol{v}}_m$ to eliminate coefficients of degrees greater than k in the corresponding Syzygy, as follows. Let M be a submatrix of $\begin{pmatrix} \tilde{\boldsymbol{v}}_1 & \cdots & \tilde{\boldsymbol{v}}_m \end{pmatrix}$ consisting of the rows corresponding the coefficients of $U_i^{(j)}$ of degree greater than k. Then, calculate the SVD on M to find basis of the null space of M. Repeat this step until we find appropriate Syzygies satisfying (1) along with $\Delta_i \neq 0$ for all i.

   (b) If we could not find all of $n-1$ independent Syzygies satisfying (1) with the above procedure, then, for degree $k \neq l$ satisfying (1), calculate new Syzygies from $N_k(\tilde{P}_1, \ldots, \tilde{P}_n)$ until we find all of $n-1$ independent Syzygies satisfying (1) along with $\Delta_i \neq 0$.

References

[1] Z. Li, Z. Yang, and L. Zhi. Blind image deconvolution via fast approximate GCD. In *Proceedings of ISSAC '10*, pages 155–162, 2010.

[2] D. Rupprecht. An algorithm for computing certified approximate GCD of n univariate polynomials. *J. Pure Appl. Algebra*, 139:255–284, 1999.

[3] A. Terui. GPGCD, an iterative method for calculating approximate GCD, for multiple univariate polynomials. In *Lecture Notes in Computer Science* 6244, pages 238–249. Springer, 2010.

[4] A. Terui. GPGCD: An iterative method for calculating approximate GCD of univariate polynomials. *Theoretical Computer Science*, 479:127–149, 2013. Symbolic-Numerical Algorithms.

Incremental PSLQ with Application to Algebraic Number Reconstruction *

Yong Feng, Jingwei Chen, Wenyuan Wu
Automated Reasoning and Cognition Key Laboratory of Chongqing, CIGIT, CAS

{yongfeng, chenjingwei, wuwenyuan}@cigit.ac.cn

1. Introduction. A vector $\boldsymbol{m} = (m_1, \ldots, m_n) \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$ is called an *integer relation* for $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ if $\sum_{i=1}^n m_i x_i = 0$. In literature, the HJLS algorithm [3, Sec. 3] and the PSLQ algorithm [2] solve the problem of finding integer relation polynomially. Although it has been theoretically proved that PSLQ is to some extent equivalent to HJLS, under the exact real arithmetic computational model (see, e.g., [7, 1]), the PSLQ algorithm seems more practical.

The problem of finding the *minimal polynomial* from an approximation $\overline{\alpha}$ of a d_0 degree *algebraic number* α, equivalent to finding an integer relation for the vector $(1, \alpha, \ldots, \alpha^{d_0})$, was first solved in [5] by using the celebrated LLL algorithm [6]. This routine has been recently improved in [4]. Naturally, the PSLQ algorithm applies to the algebraic number reconstruction problem as well [8].

Given an approximation to an unknown algebraic number α, a degree bound d and an upper bound M on its height, if the exact degree d_0 of α is also unknown, then no matter whether one uses PSLQ or LLL, one has to search an integer relation for the vector $(1, \alpha, \ldots, \alpha^i)$ from $i = 2, 3, \ldots$ until d_0 ($\leq d$). Hence, if the complexity of a polynomial algorithm for finding an integer relation is $\mathcal{O}(P(n, M))$ for an n-dimensional vector, then the complexity of the minimal polynomial algorithm, based on the integer relation finding algorithm, is $\mathcal{O}(d_0 \cdot P(d_0, M))$. Our main contribution in the present work is to give an *incremental* version of PSLQ, which leads to an efficient algebraic number reconstruction algorithm with complexity only $\mathcal{O}(P(d_0, M))$, even though the exact degree of the algebraic number is unknown.

Algorithm 1 (IPSLQ).

Input: A vector $\boldsymbol{x} = (x_1, \cdots, x_n) \in \mathbb{R}^n$ with $x_i \neq 0$ for $i = 1, \ldots, n$ and a positive number M.
Output: Either return an integer relation for $\boldsymbol{x}$, or return "no relation with length smaller than M".
1. Construct $H_x \in \mathbb{R}^{n \times (n-1)}$. Set $H := H_x$, $A := I_n$ and $B := I_n$. Size-reduce H and update A and B.
2. For k from $n-1$ to 1 do
   (a) While $h_{n-1,n-1} \neq 0$ do
      i. Choose r such that $\gamma^r |h_{r,r}| = \max_{j \in \{k, \cdots, n-1\}} \{\gamma^j |h_{j,j}|\}$.
      ii. Swap the r-th and the $(r+1)$-th rows of H and update A and B.
      iii. If $r < n-1$ then update H to L-factor of H.
      iv. Size-reduce H and update A and B.
      v. If $\max_{j \in \{k, \cdots, n-1\}} |h_{j,j}| < 1/M$ then do the following: If $k > 1$ then go to Step 2; Else return "no relation with length smaller than M".
   (b) Return the last column of B.

2. The Incremental PSLQ Algorithm. The main difference between IPSLQ (Algorithm 1) and PSLQ is that PSLQ considers $x_1, \ldots, x_n$ directly, while IPSLQ considers $x_i, \ldots, x_n$ gradually, i.e., if the vector $(x_i, \ldots, x_n)$ has no relation with 2-norm less than M then add x_{i-1} to the left; see Step 2(a)v.

The application of IPSLQ to efficient reconstructing minimal polynomial depends on the following two key points: (1) We use $(x_1, \ldots, x_n) = (\alpha^{n-1}, \ldots, \alpha, 1)$, which is the reverse order of the traditional version, to construct the matrix H_x (see [2, Def. 2] for the construction). (2) The important observation is that

*This work was partially supported by NKBRPC (2011CB302400) and NSFC (11001040, 11171053, 91118001).

the matrix H_x for $(x_i, \ldots, x_n)$ is exactly the right-bottom most submatrix of H_x for $(x_{i-1}, x_i, \ldots, x_n)$. Thus, the results produced by the previous iterations are still valid for the new matrix H. However, the traditional methods can not reuse those previous information. Therefore, the complexity of IPSLQ for minimal polynomial without knowing degree is only $\mathcal{O}(P(d_0, M))$, which is the same as PSLQ for minimal polynomial with knowing degree.

3. Experiments. The following experiments are preliminary and to compare the performance between traditional PSLQ and IPSLQ for minimal polynomial reconstruction. Consider approximations of $\alpha = 3^{1/s} + 2^{1/t}$ with 500 decimal digits. Running these experiments in Maple 15 with `Digits :=500` gives a preliminary experimental results in Table 1. Note that here `Digits :=500` may not be necessary (see [8] for the a detailed error control). In Table 1, the input degree bound and height bound in these tests are d and $M+1$; the exact degree and height of α are $d-1$ and M, respectively. All these experimental results are obtained by using a Windows 7 (32 bits mode) PC with AMD Athlon II X4 645 processor (3.10 GHz) and 4 GB memory. Note that there exists a built-in function `IntegerRelations:-PSLQ` in Maple 15, but for the comparison in Table 1, we implement the PSLQ algorithm by ourselves. The reasons we do not use the built-in function is that there does not exist a height parameter in the built-in function. This may cause that the built-in function will go on the iterations even if the height has been greater than M. In our implementations of PSLQ and IPSLQ, the same function uses the same technique for fairness. According to Table 1, the IPSLQ algorithm is faster than the PSLQ algorithm. Meanwhile the ratio between T_{PSLQ} and T_{IPSLQ} seems to get larger and larger with increasing d, but always smaller than d.

No.	s	t	d	M	T_{IPSLQ}	T_{PSLQ}	$\frac{T_{PSLQ}}{T_{IPSLQ}}$
1	2	2	5	10	0.08	0.16	2.00
2	2	3	7	36	0.16	0.64	4.00
3	3	3	10	125	0.89	5.34	6.00
4	3	4	13	540	3.14	21.34	6.79
5	2	7	15	5103	6.91	45.91	6.64
6	3	6	19	10278	23.37	144.11	6.17
7	4	5	21	11160	32.73	249.54	7.62
8	5	5	26	57500	78.95	838.99	10.63
9	5	6	31	538380	186.28	2089.87	11.22
10	6	6	37	4281690	421.94	4313.99	10.22

Table 1: IPSLQ VS PSLQ for minimal polynomial

References

[1] J. Chen, D. Stehlé, and G. Villard. A new view on HJLS and PSLQ: Sums and projections of lattices. In *Proc. ISSAC '13*, Boston, USA, 2013. To appear.

[2] H. R. P. Ferguson, D. H. Bailey, and S. Arno. Analysis of PSLQ, an integer relation finding algorithm. *Math. Comput.*, 68(225):351–369, 1999.

[3] J. Håstad, B. Just, J. Lagarias, and C. Schnorr. Polynomial time algorithms for finding integer relations among real numbers. *SIAM J. Comput.*, 18(5):859–881, 1989.

[4] M. van Hoeij and A. Novocin. Gradual sub-lattice reduction and a new complexity for factoring polynomials. *Algorithmica*, 63(3):616–633, 2012.

[5] R. Kannan, A. Lenstra, and L. Lovász. Polynomial factorization and nonrandomness of bits of algebraic and some transcendental numbers. *Math. Comput.*, 50(181):235–250, 1988.

[6] A. Lenstra, H. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261(4):515–534, 1982.

[7] A. Meichsner. Integer Relation Algorithms and the Recognition of Numerical Constants. Master's thesis, Simon Fraser University, 2001.

[8] X. Qin, Y. Feng, J. Chen, and J. Zhang. A complete algorithm to find exact minimal polynomial by approximations. *Int. J. Comput. Math.*, 89(17):2333–2344, 2012.

Hypergeometric generating functions and series for $1/\pi$

James G. Wan
Singapore University of Technology and Design
james_wan@sutd.edu.sg

Introduction

Some truly innovative series for $1/\pi$, first discovered by Ramanujan and elucidated in [1], take the form

$$\sum_{n=0}^{\infty} \frac{(s)_n(\frac{1}{2})_n(1-s)_n}{n!^3}(a+bn)z_0^n = \frac{1}{\pi}. \tag{1}$$

In other words, the constant $1/\pi$ can be written as a suitable linear combination of a hypergeometric function (in this case a ${}_3F_2$) and its derivative at some z_0. Such series have both theoretical and practical applications. In a recent preprint [3], some double sums are conjectured to also evaluate to $1/\pi$; we aim to prove them using the theory behind (1). Examples of these sums include

$$\sum_{n=0}^{\infty}\sum_{k=0}^{n} \binom{n}{k}\binom{2n-2k}{n-k}\binom{2k}{k}\binom{2n}{n}\frac{140n+19}{2^{6k}}\Big(\frac{2}{17}\Big)^{2n} = \frac{289}{3\pi}, \tag{2}$$

$$\sum_{n=0}^{\infty}\sum_{k=0}^{n} \binom{2n-2k}{n-k}\binom{2k}{k}^2\binom{2n}{n}\frac{12n+1}{6^{2k}}\Big(\frac{3}{20}\Big)^{2n} = \frac{75}{8\pi}. \tag{3}$$

Method

Instead of ${}_3F_2$'s, these conjectural series in [3] all contain functions of the form

$$G(x,z) = \sum_{n=0}^{\infty}\sum_{k=0}^{n} F(n,k)x^n z^k,$$

where F is a product of four or more binomial coefficients. It is routine to find a differential equation in x satisfied by G; however such ODEs have degrees ≥ 4 and current CAS struggle to find or rule out hypergeometric solutions implicitly required in (1). Our approach is to guess, based on numerical evidence, that x and z are connected by a simple algebraic relation r. For instance, we may guess that

$$G(x, r_{a,b}(x)) = \sum_{n=0}^{\infty}\sum_{k=0}^{n} F(n,k)\frac{x^{k+n}}{(a+bx)^{2n+1}} \text{ or } \sum_{n=0}^{\infty}\sum_{k=0}^{n} F(n,k)\frac{(-1)^n x^{k+n}}{(a+bx)^{n+1/2}}, \tag{4}$$

for some a and b. We compute sufficiently many coefficients in the x-expansion of (4), and attempt to find a, b such that they satisfy a *three-term* recurrence (with polynomials of bounded degrees as coefficients). Such a recurrence corresponds to a degree 3 ODE satisfied by G. The key step then comes down to an easy problem in linear algebra of checking if a certain determinant is zero for some a and b.

Once suitable a and b are found, we need to solve the 3rd order ODE satisfied by $G(x, r_{a,b}(x))$; for this we have a more complete theory. E. g. in the case corresponding to (2), *Maple 13* is able to give the solution which can be rearranged into a ${}_3F_2$. In the case of (3), the ODE is of Heun type, and can be solved using [2, eqn. (3.5b)] followed by a transform due to E. Goursat; we obtain

$$\sum_{n=0}^{\infty}\sum_{k=0}^{n}\binom{2n-2k}{n-k}\binom{2k}{k}^2\binom{2n}{n}\frac{x^{k+n}}{(1+4x)^{2n+1}} = \sum_{n=0}^{\infty}\frac{(\frac{1}{3})_n(\frac{1}{2})_n(\frac{2}{3})_n}{n!^3}\big(108x^2(1-4x)\big)^n. \tag{5}$$

In either case the ${}_3F_2$ is of the type in (1), and the extensive theory for producing formulas of this type can be used to prove equations (2) and (3).

Some details

When we take the x-derivative of (5) (as is required in (1)), linear dependence on k appears on the left hand side, which is not found in (3). To cancel this k term, a vanishing, k-dependent identify (known as a 'satellite identity', coined in [4]) is required. For (5), the satellite identity is

$$\sum_{n=0}^{\infty}\sum_{k=0}^{n}\binom{2n-2k}{n-k}\binom{2k}{k}^2\binom{2n}{n}\frac{x^{k+n}}{(1+4x)^{2n}}\big(4x+2k(4x+1)+n(4x-1)\big) = 0. \tag{6}$$

Identity (6) was *guessed* as follows: pick a small, irrational x and compute $a_0 = \sum_{n,k} A(n,k,x)$, $a_1 = \sum_{n,k} A(n,k,x)k$, and $a_2 = \sum_{n,k} A(n,k,x)n$ (A being the summand), then use PSLQ to find a null integer linear combination among the elements of $\{a_0, a_1, a_2, a_0x, a_1x, a_2x, a_0x^2, a_1x^2, a_2x^2, \ldots\}$. Once found, the satellite identity can be proven by the multiple WZ algorithm. Similarly, (5) itself can be rigorously proven (as the 3rd order recursion was only a guess): write the coefficients of x on the LHS as a double sum, apply the multiple WZ algorithm to obtain a recursion, convert it to an ODE for the LHS, and finally check that the ODE annihilates the RHS. Many conjectures from [3] have been settled using our method, via the discovery of generating functions like (5).

Future work

Some conjectures in [3] do not fall into the type (4); perhaps more elaborate algebraic relations are needed – this could also anticipate more exotic generating functions. It would be illuminating to be able to find suitable a and b in (4) analytically (without extensive computer searches), and also to prove the existence of satellite identities whenever F is a hypergeometric term.

References

[1] J. M. Borwein and P. B. Borwein, *Pi and the AGM: A study in analytic number theory and computational complexity* (Wiley, New York, 1987).

[2] R. Maier, Transforming the Heun equation to the hypergeometric equation: I. Polynomial transformations, *preprint* (2002).

[3] Z.-W. Sun, List of conjectural series for powers of π and other constants, *preprint* `arXiv: 1102.5649` (Jan 2012).

[4] W. Zudilin, A generating function of the squares of Legendre polynomials, *Bull. Austral. Math. Soc. to appear* (2013).

Annihilating monomials with the integro-differential Weyl algebra

Johannes Middeke
Department of Applied Mathematics
University of Western Ontario
London (ON), Canada, N6A 5B7
jmiddeke@uwo.ca

Abstract

The integro-differential Weyl algebra provides an algebraic model for differential and integral operators with polynomial coefficients. It has a natural action on the ring of polynomials. We are interested in computing the annihilator of a given polynomial with respect to this action. This contribution contains a first step towards that goal—namely we give a description of the annihilator of a monomial.

1 The integro-differential Weyl algebra

The (univariate) integro-differential Weyl was introduced for the first time in [1]. It arose from the algebra of integro-differential operators first discussed in [2] and refined in [3, 4] with the goal of solving boundary value problems using purely algebraic methods. While the aforementioned papers construct integro-differential operators with arbitrary coefficients using a Gröbner basis approach, in [1] the authors chose to model operators with polynomial coefficients using Ore polynomials.

A way to construct the integro-differential Weyl algebra as a generalised Weyl algebra has been discussed in [5].

In this abstract we briefly recall the basic properties of the integro-differential Weyl algebra and refer to [1] for details. Let $\mathbb{K}$ be a field of characteristic 0. The *integro-differential Weyl algebra*—denoted $A_1(\partial, \int)$—is the $\mathbb{K}$-algebra generated by the symbols x, ∂ and $\int$ and defined by the equations

$$\partial x = x\partial + 1, \qquad \textstyle\int\int = x\int - \int x, \qquad \text{and} \qquad \partial\int = 1. \tag{1}$$

One can prove that $A_1(D, \int)$ is neither simple nor left or right Noetherian; it even contains zero divisors. Also, the integro-differential operators from [2, 3, 4] are isomorphic to $A_1(\partial, \int)/(\mathrm{E}x - c\mathrm{E})$ where $\mathrm{E} = 1 - \int\partial$ and c is a constant depending on the integral operator—cf again [1].

Interpreting ∂ as derivation and $\int$ as an integral, the integro-differential Weyl algebra has a natural action on the polynomial ring $\mathbb{K}[x]$. More precisely, the action $*\colon A_1(\partial, \int) \times \mathbb{K}[x] \to \mathbb{K}[x]$ is defined by

$$x * x^n = x^{n+1}, \qquad \partial * x^n = \frac{dx^n}{dx} = nx^{n-1}, \qquad \text{and} \qquad \textstyle\int * x^n = \int_0^x x^n dx = \frac{1}{n+1}x^{n+1}$$

where $n \geq 0$. With this action, the relations in (1) model the Leibniz rule, partial integration and the fundamental theorem of calculus, respectively. Moreover, E corresponds to the evaluation at 0.

2 Annihilators

We want to compute annihilators of polynomials, i. e., the set of all operators in $A_1(\partial, \int)$ whose action sends a given polynomial to zero. It is well-known that annihilators are left ideals in $A_1(\partial, \int)$. As a first step towards computing them, we give the following theorem:

Theorem 1 *For any $n \geq 1$, the annihilator of x^n within the integro-differential Weyl algebra $A_1(\partial, \int)$ is generated as a left ideal by*

$$\partial^{n+1}, \quad (x - \textstyle\int)\partial^n, \quad \mathrm{E}\partial^{n-1}, \quad \ldots, \quad \mathrm{E}\partial, \quad \mathrm{E}$$

for $n \geq 0$ and the annihilator of 1 is generated by ∂ and $x - \int$. In particular, all these annihilators are finitely generated.

As a next step, we intend to generalise this result to annihilate arbitrary polynomials. Also, we shall add more evaluation operators to our algebra in order to model boundary conditions instead of only initial value problems. Moreover, a theorem in [6] states that all finitely generated left ideals in $A_1(\partial, \int)$ can be generated by only two elements. Therefore, another path of research is to attempt to compute these two generators for the annihilators.

References

[1] Regensburger, G., Rosenkranz, M. & Middeke, J. A skew polynomial approach to integro-differential operators. In May, J. P. (ed.) *Proceedings of ISSAC 2009* (Association for Computing Machinery, 2009).

[2] Rosenkranz, M. A new symbolic method for solving linear two-point boundary value problems on the level of operators. *Journal of Symbolic Computation* **39**, 171–199 (2005).

[3] Rosenkranz, M. & Regensburger, G. Solving and facroring boundary problems for linear ordinary differential equations in differential algebras. *Journal of Symbolic Computation* **43**, 515–544 (2008).

[4] Regensburger, G. & Rosenkranz, M. An algebraic foundation for factoring linear boundary problems. *Annali di Matematica Pura ed Applicata* **188**, 123–151 (2009).

[5] Bavula, V. V. The algebra of integro-differential operators on a polynomial algebra. *Journal of the London Mathematical Society* **83**, 517–543 (2011).

[6] Bavula, V. V. The algebra of integro-differential operators on an affine line and its modules. *Journal of Symbolic Computation* 495–529 (2013).

Probabilistic Analysis of Wiedemann's Algorithm for Minimal Polynomial Computation

Gavin Harrison, Jeremy Johnson, B. David Saunders
Department of Computer Science
Drexel University, University of Delaware
gmh33@drexel.edu,jjohnson@cs.drexel.edu,saunders@udel.edu

Blackbox algorithms for linear algebra problems start with one sided (Lanczos) or two sided (Wiedemann) projection of the sequence of powers of a matrix to a sequence of scalars or a sequence of smaller matrices. Such algorithms usually require that the minimal polynomial of the resulting sequence should be that of the given matrix. Exact formulas are given for the probability that this occurs based on the Jordan structure of a matrix, and from these formulas sharp bounds follow. The bounds are valid for all finite field sizes and show that a small blocking factor can give high probability of success for all cardinalities and matrix dimensions.

Let K be a finite field with cardinality q. Given $A \in K^{n\times n}$, and $\bar{A}$ be the linearly generated sequence $\{I, A, A^2, \ldots\}$. Given $U, V \in K^{n\times b}$ whose elements are selected uniformly randomly from K, $U^T\bar{A}V$ is a linearly generated sequence of smaller matrices, and with high probability, the minimal generating polynomial of $U^T\bar{A}V = \{U^TV, U^TAV, U^TA^2V, \ldots\}$ is the minimal polynomial of the matrix A. All square matrices are similar to a generalized Jordan form matrix, $A = PJP^{-1}$, where $J, P \in K^{n\times n}$. If U and V are selected uniformly randomly, then $X = P^TU$ and $Y = P^{-1}V$ are also uniformly random. $U\bar{A}V = X^T\bar{J}Y = \{X^TY, X^TJY, X^TJ^2Y, \ldots\}$, and $X\bar{J}Y$ has the same probability as $U\bar{A}V$ of having its minimal generating polynomial match the minimal polynomial of A and J. We call this probability $Prob_{q,b}(A)$.

Let $C_f \in K^{d\times d}$ represent the companion matrix for the polynomial $f(x) = f_0 + f_1x + \ldots + f_{d-1}x^{d-1} + x^d$ with coefficients in K. Let J_{f^e} be the generalized Jordan block of an irreducible f occurring with multiplicity e. Since $Prob_{q,b}(J_f \oplus J_g) = Prob_{q,b}(J_f)Prob_{q,b}(J_g)$ when $\gcd(f, g) = 1$, unique irreducibles can be treated separately, and for each irreducible only its highest multiplicity affects the probability that the projection preserves the minimal polynomial. Furthermore we show $Prob_{q,b}(J_f) = Prob_{q,b}(J_{f^e})$ for any e. Therefore, letting $T = \{(f_1, e_1, t_1), (f_2, e_2, t_2), \ldots\}$, where the polynomials f_i are the irreducibles occuring in the invariant factors of A, e_i is the highest multiplicity of f_i, and t_i is the number of occurrences of $f_i^{e_i}$, it follows that

$$Prob_{q,b}(J) = \prod_{k=1}^{|T|} Prob_{q,b}\left(\bigoplus_{t_k} J_{f_k}\right).$$

For an irreducible polynomial f of degree d, the probability that $U^T\bar{C}_fV$ has minimal polynomial f is easy to determine. The minimal polynomial of the projection is always a factor of f, which for irreducible f is 1 or f. It is 1 only if the sequence is a sequence of zero matrices, which is to say that one of U, V is zero. Thus

$$Prob_{q,b}(C_f) = (1 - q^{-db})^2.$$

If $J = \bigoplus_t C_f$, then for $U, V \in K^{dt\times b}$, with blocking conformal to the diagonal blocks of J, we have $U^T\bar{J}V = \sum_{k=1}^{t} U_k^T\bar{C}_fV_k$. We show $Prob_{q,b}(C_f) \leq Prob_{q,b}\left(\bigoplus_t J\right)$.

	field cardinality			
block size	2	3	10007	$2^{31}-1$
1	0.000467	0.00112	0.0499	0.911
2	0.25	0.444	$1-2\times10^{-4}$	$1-4.3\times10^{-11}$
4	0.766	0.927	$1-2\times10^{-12}$	$1-9.4\times10^{-30}$
8	0.984	$1-9.1\times10^{-4}$	$1-2\times10^{-28}$	$1-4.4\times10^{-67}$
16	$1-6\times10^{-5}$	$1-1.4\times10^{-7}$	$1-2\times10^{-60}$	$1-9.8\times10^{-142}$
32	$1-9.3\times10^{-10}$	$1-3.2\times10^{-15}$	$1-2\times10^{-124}$	$1-4.8\times10^{-291}$

Table 1: Bounds for worst case probability of success to preserve minimum polynomial, matrix size $10^8 \times 10^8$

It is evident that the probability of success increases with d as well as with b. The worst case is a matrix whose minimal polynomial is a distinct product of the smallest possible irreducibles. This yields an exact lower bound formula for the probability that a projection $U^T\bar{A}V$ of A has the same minimal polynomial. Let $L_q(d,n)$ be the number of degree d irreducible factors over the finite field of cardinality q that fit in a matrix of dimension n after all smaller degree irreducibles have been inserted. Then, for an $n \times n$ matrix A,

$$Prob_{q,b}(A) \geq \prod_{d=1}^{\infty} \left(1 - \frac{2q^{db}-1}{q^{2db}}\right)^{L_q(d,n)}$$

We compare this bound to previously given lower bounds in the case when field cardinality and matrix dimension are of similar size. For small primes, Wiedemann (proposition 3) treats the case $b = 1$ and he fixes the projection on one side because he is interested in linear system solving and thus in the sequence $\bar{A}b$ [2] . For small q, his formula, $1/(6\log_q(N))$, computed with some approximation, is nonetheless quite close to our exact formula. However as q approaches N the discrepancy with our exact formula increases. At the large/small crossover, $q = N$, Kaltofen/Pan's lower bound is 0, Wiedemann's is 1/6, and ours is $1/e$. The Kaltofen/Pan probability bound improves as q grows larger from N [1] . The Wiedemann bound becomes more accurate as q goes down from N. But the area $q \approx N$ is of some practical importance. In integer matrix algorithms where the finite field used is a choice of the algorithm, sometimes practical considerations of efficient field arithmetic encourages the use of primes in the vicinity of N. For instance, exact arithmetic in double precision and using BLAS works well with $q \in 10^6..10^7$. Sparse matrices of order N in that range are tractable. Our bound may help justify the use of such primes.

But the primary value we see in our analysis here is the understanding it gives of the value of blocking, $b > 1$. Table 1 shows the bounds for the worst case probability that a random projection will preserve the minimal polynomial of a matrix $A \in K^{10^8 \times 10^8}$ for various fields and projection block sizes. It shows that the probability of finding the minimal polynomial correctly under projection converges rapidly to 1 as the projected block size increases. Even over $GF(2)$, with block size $b = 16$ the probability is very good.

References

[1] Erich Kaltofen and B. David Saunders. On wiedemann's method of solving sparse linear systems. In *Proceedings of the 9th International Symposium, on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, AAECC-9, pages 29–38, London, UK, UK, 1991. Springer-Verlag.

[2] D. Wiedemann. Solving sparse linear equations over finite fields. *Information Theory, IEEE Transactions on*, 32(1):54–62, 1986.

Implementation of a Solution to the Conjugacy Problem in Thompson's Group F

James Belk, Nabil Hossain, Francesco Matucci, and Robert McGrail
Bard College, Annandale-on-Hudson, NY, USA, 12504
Université Paris-Sud 11, Bâtiment 425, Bureau 21, F-91405 Orsay Cedex, France
belk@bard.edu, nh1682@bard.edu, francesco.matucci@math.u-psud.fr, mcgrail@bard.edu

Abstract

We present an efficient implementation of the solution to the conjugacy problem in Thompson's group F. This algorithm checks for conjugacy by constructing and comparing directed graphs called strand diagrams. We provide a description of our solution algorithm, including the data structure that represents strand diagrams and supports simplifications.

1 Thompson's Group F and Strand Diagrams

The elements of Thompson's Group F [3] are piecewise, linear homeomorphisms of the interval $[0, 1]$ such that each piece has slope that is a power of 2 and, furthermore, the breakpoints between pieces take place at dyadic rational coordinates. The group operation is simply function composition. In a group, the **conjugacy problem** is the problem of determining whether any two elements are conjugate. The conjugacy problem is not solvable in general [5], but is solvable in certain cases.

A **strand diagram** [2] is a finite acyclic digraph embedded on the unit square. The digraph has a **source** along the top edge of the square and a **sink** along the bottom edge. Any internal vertex is either a **merge** or a **split** (Figure 1). Elements of Thompson's Group F can be translated to strand diagrams. Each element in a generating set corresponds to a particular strand diagram. A composition of such elements is represented by a concatenation of the associated strand diagrams.

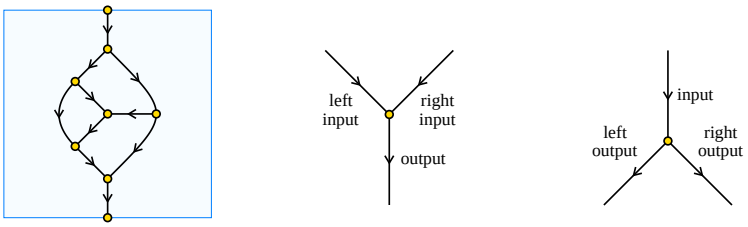


Figure 1: A strand diagram, a merge, and a split (image taken from [2]).

2 Algorithm for the Conjugacy Problem in F

The algorithm to determine whether two strand diagrams inhabit the same conjugacy class proceeds as follows. First, we convert the strand diagrams to **annular strand diagrams**. This is achieved by a process called **closing**, in which sources are identified with sinks. Next, the annular strand diagrams are

reduced using a graphical rewriting system that is both confluent, terminating, and respects conjugacy [1]. Furthermore, any two connected and reduced annular strand diagrams s_1 and s_2 can be encoded into two planar graphs g_1 and g_2 respectively such that s_1 and s_2 represent conjugate elements if and only if g_1 and g_2 are isomorphic. Hence the problem reduces to checking whether two simplified planar graphs are isomorphic. Moreover, this enterprise can be carried out in linear time given a linear time planar-graph-isomorphism checker [4].

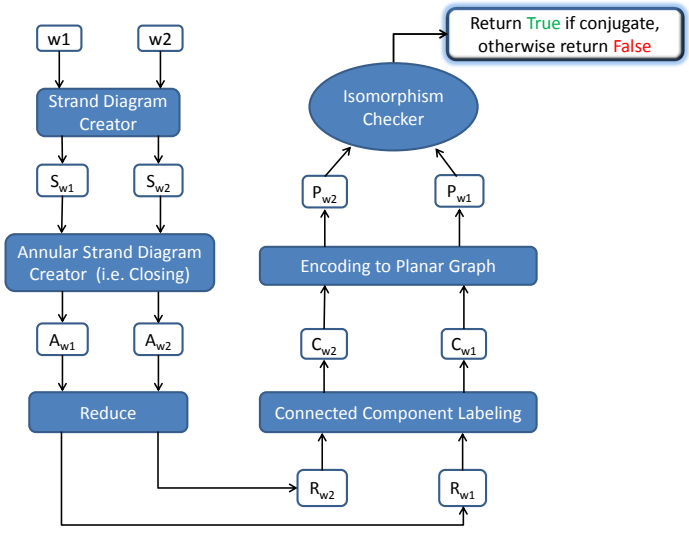


Figure 2: Algorithm Flowchart

References

[1] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1999.

[2] J. Belk and F. Matucci. Conjugacy and dynamics in Thompson's groups. Preprint, 2013.

[3] J. W. Canon, W. J. Floyd, and W. R. Parry. Introductory notes on Richard Thompson's groups. *Enseignement Mathématique*, 42: 215–256, 1996.

[4] J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). *Proceedings of the Sixth Annual ACM symposium on Theory of Computing*, 172–184, 1974.

[5] P. S. Novikov. Unsolvability of the conjugacy problem in the theory of groups. *Izv. Akad. Nauk SSSR. Ser. Mat*, 18: 485–524,1954.

A Linear Sparse Systems Solver (LSSS) applied to the Classification of Integrable non-abelian Laurent ODEs

Thomas Wolf
Brock University
Ontario, Canada
twolf@brocku.ca

Eberhard Schrüfer
Fraunhofer Institut
Bonn, Germany
eschruefer@ca-musings.de

Kenneth Webster
University of
Waterloo, Canada

Sparse linear algebraic systems can be of different nature. A well known class are systems, we call them "numerical", that result in the discretization of partial differential equations (PDEs). A very different class of systems, we call them "selective", arises from integrability investigations of differential equations. Both types of systems behave very differently during the solution process and give results of different nature and therefore are also solved most efficiently with different methods.

The Linear Selective Systems Solver LSSS, described on the poster, was developed to solve "selective" systems that result when the aim is to find discrete mathematical objects of symbolic nature like Lie symmetries or first integrals if they exist. Table 1 compares both types of problems.

type	"numerical" systems	"selective" systems
examples	systems resulting from a discretization of PDEs	systems resulting from a symmetry investigation of PDEs
value of free parameters when applying the solution of the linear system	any floating point numbers (boundary values of PDE)	0 or 1 (to isolate the individual symmetries)
number of zero-valued variables in solution	essentially none	most variables
initial sparsity	yes	yes
sparsity throughout exact solution	yes	yes
overdetermination	no	yes
usability of iteration schemes for large problems of that type	useful	not useful

Table 1: Characterization of two different types of sparse linear systems

The special nature of selective systems allows a dedicated computer program LSSS (Linear Sparse Systems Solver) running in the computer algebra system REDUCE to be much more efficient than conventional computer programs for solving these systems [1]. Reasons are:

- Because of the existence of many variables that take the value of zero in the solution the systems involve 1-term equations which are utilized first to simplify the remaining system and generate more 1-term equations.
- The simplification of a system due to the vanishing of variables can be accomplished much faster than the simplification due to other substitutions.
- From the mathematical problem it is clear whether the type of a system is numerical or selective and thus to apply the most suitable technique from the start.
- Selective linear systems are typically formulated by separation of larger expressions. The complete separation and up-front formulation of the whole linear system can be avoided through a repeatedly selective splitting and thus formulation and solution of 1-term equations.
- Increasing the complexity of the mathematical problem (e.g. by a increased degree of the ansatz for symmetries or first integrals) the overdetermination and sparseness increases compensating partially the exploding size of the initial linear system if 1-term equations are used rigorously.

The package LSSS was developed in the course of investigating the integrability of the Kontsevich system ([3])

$$u_t = uv - uv^{-1} - v^{-1},\ v_t = -vu + vu^{-1} + u^{-1} \tag{1}$$

where u, v are non-commutative variables (in particular, square matrices of arbitrary size). This is the first non-abelian system with non-polynomial right hand sides for which integrability could be shown, in this case by computing a Lax pair with spectral parameter [2]. Essential ingredients were the computation of Lie-symmetries, first integrals, a pre-Hamiltonian operator and a recursion operator of (1), all of them requiring the solution of selective linear systems. With the program LSSS it was possible to compute Lie-symmtries of degree up to 16. The complete linear system that had been solved includes over 10^9 equations for 172 Mio variables. Despite of the majority of them being zero, the general solution is not trivial as it has 32 free parameters and its formulation requires already several mega byte.

Current applications of LSSS include the integrability investigation of generalizations of the form

$$u_t = uv + P(u, v, u^{-1}, v^{-1}),\ v_t = -vu + Q(u, v, u^{-1}, v^{-1}). \tag{2}$$

References

[1] Wolf, T., Schrüfer, E., Webster, K. Solving large linear algebraic systems in the context of integrable non-abelian Laurent ODEs, *Programming and Computer Software*, (2012) DOI:10.1134/S0361768812020065, also arXiv:1109.2785 (nlin.SI).

[2] Wolf, T., Efimovskaya, O. On integrability of the Kontsevich non-abelian ODE system, *Lett. in Math. Phys.*, vol 100, no 2 (2012), p 161-170 DOI:10.1007/s11005-011-0527-4, also arXiv:1108.4208v1 (nlin.SI).

[3] Kontsevich, M., private communication.

Abstracts of Recent Doctoral Dissertations in Computer Algebra

Each month we are pleased to present abstracts of recent doctoral dissertations in Computer Algebra and Symbolic Computation. We encourage all recent Ph.D. graduates (and their supervisors), who have defended in the past two years, to submit their abstracts for publication in CCA.
Please send abstracts to the CCA editors <editors_SIGSAM@acm.org> *for consideration.*

Author: Brice Boyer
Title: Efficient matrix multiplication and design for the exact linear algebra library LinBox
Institution: Laboratoire Jean Kuntzman, Université de Grenoble.
Thesis Advisor: Jean-Guillaume Dumas
Defended: June 2012
Keywords: exact linear algebra, sparse matrix, SpMV, dense matrix, fast matrix multiplication, pebble game, schedulings, design patterns, generic mathematics library, LinBox.

Matrix multiplication is a major cornerstone in exact linear algebra: its study can concern algorithmic, complexity, design, reduction, *etc.* problems. We are interested in the few following aspects.

We first expose, in this thesis, efficient exact matrix multiplication techniques, developed for both multiplication ($A = B \times C$) and product with accumulation ($A = A + B \times C$). We set up new schedules that allow us to minimize the extra memory requirements during a Strassen–style matrix multiplication, while keeping the complexity competitive with Winograd's multiplication algorithm. In order to obtain them, we develop external tools (pebble games), tight complexity computations and new hybrid algorithms.

We then use parallel technologies (multicore CPU and GPU) in order to efficiently accelerate the sparse matrix–dense vector multiplication (SpMV) or sparse–matrix dense matrix multiplication (SpMM), crucial to *blackbox* (block) algorithms. We also set up new hybrid, environment dependant, sparse matrix formats that help yield large speed-ups. We exemplify these results by speeding up the block Winograd rank algorithm in the LinBox library.

Finally, we establish generic design methods focusing on efficiency, especially via *building block* conceptions or self-optimization. We also propose tools for improving and standardizing code quality in order to make it more sustainable and more robust. This is applied in particular to the LinBox computer algebra library.

Author: Ibrahim Adamou
Title: Curve and Surface Bisectors, and Voronoi Diagram of a family of parallel half-lines in $\mathbb{R}^3$
Institution: Universidad de Cantabria

Thesis Advisor: Laureano Gonzalez-Vega and Mario Fioravanti
Committee Members: Tomas Recio, Marie-Françoise Roy, Bert Jüttler
Defended: September 10th, 2013 *Keywords:* Bisectors, Rational Curves and Surfaces, Voronoi Diagram, Spatial Subdivision, Meshing.

This thesis has three main parts: computation of the bisectors of two curves or a point and a curve in the plane, of the bisector of two surfaces in $\mathbb{R}^3$, and of the Voronoi diagram of a finite family of parallel half lines in $\mathbb{R}^3$, with the same orientation. These subjects are closely related, and have applications in CAD/CAGD and Computational Geometry. In each of the three parts, we present algorithmic methods for computing certain representations of the geometric object of interest: the bisector curve, the bisector surface, or the Voronoi diagram.

We present a new approach to determine an algebraic parametrization (rational or non rational) of the bisector curve of two given planar rational curves. The method uses Cramer's rule and algebraic elimination steps. The method is applied, in particular, to obtain parametrizations of the bisector of two rational plane curves, when one of them is a circle or a straight line. Then, this approach is generalized to determine an algebraic parametrization of the bisector surface of two low degree rational surfaces. We show how to easily obtain parametrizations of the bisector of the following pairs of surfaces: plane-quadric, plane-torus, circular cylinder-non developable quadric, circular cylinder-torus, cylinder-cylinder, cylinder-cone and cone-cone. These parametrizations are rational in most cases. In the remaining cases, the parametrization involves one square root which is well-suited to determine a good approximation of the bisector.

In addition, we present a different approach for the bisector curve problem. This new method uses dynamic color in GeoGebra (a dynamical geometry software) for the geometric and numerical characterizations of the bisector of two curves, or a curve and a point, in the plane. Even if it does not provide an algebraic representation, the method could lead to the computation of an approximate representation of the bisector curve.

The Voronoï diagram (VD) is a fundamental data structure in computational geometry with various applications in theoretical and practical areas. We consider the VD of a set of parallel half-lines, with the same orientation, constrained to a compact domain $\mathcal{D}_0 \subset \mathbb{R}^3$, with respect to the Euclidean distance. This new kind of VD can be used to provide an efficient solution to some problems in the drilling industry. We present an efficient algorithm for computing an approximate VD, using a box subdivision process, which produces a mesh representing the topology of the VD in $\mathcal{D}_0$. The concept of minimization diagram plays an important role in the method.

Recent and Upcoming Events

October 23–25, 2013
2nd International Seminar on Program Verification, Automated Debugging and Symbolic Computation
Beijing, China

Organizers: Tudor Jebelean, Wei Li, Dongming Wang
Dates: Submission deadline: September 10, 2013
Website: `http://pas2013.cc4cm.org/`

December 11–13, 2013
5th International Conference on Mathematical Aspects of Computer and Information Sciences
Nanning, China

Organizers: Dongming Wang, Jinzhao Wu
Dates: Submission deadline: October 12, 2013
Website: `http://www.mpi-inf.mpg.de/conference/macis2013/`

January 29–February 1, 2014
9th International Conference on Applied Informatics (ICAI 2014)
Eger, Hungary

Organizers: Attila Pethő, Franz Winkler, Roland Kunkli, Gabor Kusper
Dates: Submission deadline: January 6, 2014
Website: `http://icai.ektf.hu/`

March 31–April 4, 2014
Latin American Theoretical INformatics (LATIN 2014)
Montevideo, Uruguay

Organizers: A. Viola (PC chair), A. Pardo (Local chair)
Dates: Abstract submission: September 17, 2014; Full submission: September 22, 2014
Website: `http://www.fing.edu.uy/eventos/latin2014/`

June 16–20, 2014
25th International Conference on Probabilistic, Combinatorial, and Asymptotic Methods for the Analysis of Algorithms (AofA'14)
Paris, France

Organizers: M. Bousquet-Melou, M. Soria
Dates: Submission deadline: January 27, 2014
Website: `http://www.aofa14.upmc.fr/`

June 29–July 3, 2014
26th International Conference on Formal Power Series and Algebraic Combinatorics (FPSAC 2014)
Chicago, USA

Organizers: L. Billera, I. Novik (PC Chairs), B. Tenner (Local Chair)
Dates: Submission deadline: November 18
Website: `https://sites.google.com/site/fpsac2014/`

July 9–12, 2014
20th Conference on Applications of Computer Algebra
New York, USA

Organizers: R.H. Lewis (general chair), T. Shaska, I. Kotsireas (PC Chairs)
Dates: session proposal: February 28, 2014; talk submissions: May 15, 2014
Website: `http://faculty.fordham.edu/rlewis/aca2014/`

July 9–24, 2014
Vienna Summer of Logic
Vienna, Austria

Organizers: M. Baaz, A. Ciabattoni, Th. Eiter, A. Leitsch, G. Gottlob, T. Henzinger, V. Sabljakovic-Fritz, S. Szeider, H. Veith, S. Woltran and others
Website: `http://vsl2014.at/`

Message from the SIGSAM Chair

Ilias S. Kotsireas
Wilfrid Laurier University
Waterloo, ON, Canada

Dear SIGSAM Members,

I would like to take this opportunity to update you on the two ACM meetings that I recently attended on behalf of SIGSAM in New York City:

- September 30, 2013, SIG Leader Orientation meeting
- October 1, 2013 SGB[1] meeting.

1 SIG Leader Orientation meeting

During the SIG Leader Orientation meeting, the participants had the chance to hear introductory remarks from Erik Altman (SGB Chair) and John White, (ACM CEO). Subsequently, Erik Altman spoke on the topic of the Volunteer Structure of the ACM. He emphasized the SIG Video Repositories initiative and focused on the example of SIGSIM that has created a 1000+ simulation-related videos on their webpage `http://www.acm-sigsim-mskr.org/Videos/videos.htm` . Among the benefits of such a repository are that it represents a good value for members, could be used to entice more people to join a SIG. In addition, the repository could use automated video mining capability. I thought this is a very interesting initiative and would like to solicit opinions from SIGSAM Members on whether we could organize a similar initiative within SIGSAM.

During the second part of the SIG Leader Orientation meeting I attended four workshops: (1) Finance, (2) Marketing/Membership, (3) Publications, (4) Information Systems. Here is a summary of some interesting aspects of these workshops:

1. ACM Tech Pack initiative `http://techpack.acm.org/` . Production of annotated bibliographies on specific areas compiled by experts. Examples include the Cloud Computing, the Parallel Computing and the Security Tech Packs.
2. ACM Learning Webinars `http://learning.acm.org/webinar/` . Started in 2012, these video presentations on topics of interest to a wide range of computing professionals are non-vendor-specific and open to everyone.
3. ACM Distinguished Speakers Program `http://dsp.acm.org/` . Potential speakers can be nominated and ACM covers the cost of transportation for the speaker to travel an event.
4. ACM Books `http://books.acm.org/` . A new ACM book series has been created, in collaboration with Morgan & Claypool Publishers based in San Francisco.

[1] SIG Governing Board

2 SIG Governing Board meeting - ACM-W

During the SGB meeting there were several interesting presentations, but i would like to focus on the presentation on **ACM-W**, `http://women.acm.org/` the ACM organization whose mission statement reads:

> ACM-W supports, celebrates, and advocates internationally for the full engagement of women in all aspects of the computing field, providing a wide range of programs and services to ACM members and working in the larger community to advance the contributions of technical women.

On of the several interesting initiatives sponsored by ACM-W is the ACM-W Athena Lectures award. Athena Lectures celebrate outstanding women researchers who have made fundamental contributions to computer science. Each year ACM will honor a preeminent woman computer scientist as the Athena Lecturer. Speakers are nominated by SIG officers. The Athena Lecturer will give a one-hour invited talk at an ACM conference determined by the speaker and the SIG which nominated her. A video of the talk will appear on the ACM website. The award includes travel expenses to the meeting and a $ 10000 honorarium. Financial support for the 2008-2009 through 2014-2015 Athena Lecturers, is being provided by Google.

I believe that SIGSAM should try to nominate one of its members for the ACM-W Athena Lectures award and would like at this point to solicit suggestions for potential qualified candidates.

The minutes of the above two meetings as well as overheads of workshop presenters can be found on-line at `http://www.acm.org/sigs/sgb/minutes` and I would encourage you to browse these materials for more information on ACM initiatives and more details on how these are potentially relevant to SIGSAM.

Ilias S. Kotsireas
SIGSAM Chair `chair_SIGSAM@acm.org`
Wilfrid Laurier University
Waterloo, ON, Canada

ACM Communications in Computer Algebra, Vol. 47, No. 4, Issue 186, December 2013

A computer algebra user interface manifesto

David R. Stoutemyer*

Abstract

Many computer algebra systems have more than 1000 built-in functions, making expertise difficult. Using mock dialog boxes, this article describes a proposed interactive general-purpose wizard for organizing optional transformations and allowing easy fine grain control over the form of the result – even by amateurs. This wizard integrates ideas including:

• flexible subexpression selection;

• complete control over the ordering of variables and commutative operands, with well-chosen defaults;

• interleaving the choice of successively less main variables with applicable function choices to provide detailed control without incurring a combinatorial number of applicable alternatives at any one level;

• quick applicability tests to reduce the listing of inapplicable transformations;

• using an organizing principle to order the alternatives in a helpful manner;

• labeling quickly-computed alternatives in dialog boxes with a preview of their results, using ellipsis elisions if necessary or helpful;

• allowing the user to retreat from a sequence of choices to explore other branches of the tree of alternatives – or to return quickly to branches already visited;

• allowing the user to accumulate more than one of the alternative forms;

• integrating direct manipulation into the wizard; and

• supporting not only the usual input-result pair mode, but also the useful alternative derivational and *in situ* replacement modes in a unified window.

1 Introduction

"*Before the Lisp machine interface to Macsyma, computer algebra was like doing mathematics encumbered by boxing gloves.*"
– Bill Gosper

I am sorry Bill, but that user interface from 1988 [18] disappeared with the Lisp machine, and its best features regrettably have not yet been implemented in any of the current most powerful computer algebra systems. Even the much earlier 1972 article [6] discusses desirable features that are still missing from modern systems.

Many computer algebra systems have more than 1000 built-in functions. Besides standard *mathematical functions* such as cos(...) and classic higher transcendental functions, built-in functions often include numerous *optional transformation functions* such as expand(...), factor(...),

*dstout at hawaii dot edu

trigExpand(...), convert(...), and simplify(...) that supplement default simplification with various transformations. Besides the expression being transformed, these transformational functions often accept optional extra arguments such as a list of variables, and/or various keyword arguments that control details such as the amount of factoring. Moreover, most computer algebra systems have numerous global *control variables* whose values help control transformations done by these functions and/or by default.

Unlike cos(...), the names and semantic details of these transformation functions and control variables are not part of any standard mathematics curriculum. Therefore it requires a long time to fully exploit such systems well, and most users never do. Moreover, these names and behaviors vary greatly between systems, making it challenging to become skilled with more than one system in order to exploit their differing capabilities. Consequently many users are frustrated because they don't know how to make any system transform an expression to a desired form.

Worse yet, often there is *no* composition of functions and/or or combination of control-variable settings capable of producing a desired form. For example, users often want expansion with respect to a certain proper subset of the variables, with polynomial coefficients that are factored with respect to the other variables – or want partial fractions with respect to a certain proper subset of some variables, with factored numerators and denominators.

This article address the usefulness of computer algebra systems as productivity tools to help amateur users accomplish their tasks without necessarily being aware of the underlying algorithms, transformation functions and their nomenclature. This article presents some ideas for enhancing the kind of wizard implemented on the Lisp Machine by:

1. more flexible subexpression selection;
2. complete control over the ordering of variables and commutative operands, with well-chosen defaults;
3. interleaving the choice of successively less main variables with applicable function choices to provide detailed control without incurring a combinatorial number of applicable alternatives at any one level;
4. quick applicability tests to reduce the listing of inapplicable transformations;
5. using an organizing principle to order the alternatives in a helpful manner;
6. labeling quickly-computed alternatives in dialog boxes with a preview of the result, using ellipsis elisions if necessary or helpful;
7. allowing the user to retreat from a sequence of choices to explore other branches of the tree of alternatives – or to return quickly to branches already visited;
8. allowing the user to accumulate more than one of the alternative forms;
9. integrating direct manipulation into the wizard; and
10. supporting not only the usual input-result mode, but also the useful alternative derivational and *in situ* replacement modes in a unified window.

Section 2 describes important features to combine in a user interface. Section 3 presents some mock examples of using the proposed wizard. Section 4 addresses design issues and their resolution, with

a summary in Section 5. The Appendix summarizes some important rational-expression transformations that should be included in addition to those discussed in subsection 4.3. The wizard must also be aware of all of the many transformations specific to irrational expressions that are built in or should be, but that is an open-ended topic too large for discussion here. However, the wizard should be implemented in an extensible way that allows users to add components easily at run time for new transformations that they implement.

This article often uses "Float" as an abbreviation for "floating-point number".

2 Important features to combine in a user interface

Many of the ideas in this section have been implemented to some extent in various computer algebra systems, but integrating them into a uniformly designed user interface could greatly enhance the user experience over that of any one current system.

Computer algebra often generates large expressions, and with current RAM sizes measured in gigabytes, screen area is now the most precious resource for most user's tasks.[1] Many of the ideas discussed here are concerned with attempting to make the best use of that limited resource to help amateur and expert users arrive quickly at the most comprehensible alternative output forms.

2.1 Enhancing the Lisp Machine Macsyma precedent

Among the most helpful features of Lisp Machine Macsyma was that as you moved the mouse over an expression, the minimal rectangle containing a syntactically complete subexpression surrounding the mouse pointer would automatically be framed, which is perhaps the entire expression. A right click would open a drop-down menu of common transformations such as factor and expand, or you could enter a function name of your own. The selected transformation is applied to the framed subexpression.

This feature could be regarded as a *wizard* that helped users quickly locate appropriate subexpressions for desired transformations, then apply them to those subexpressions. This is important because without subexpression selection and shortcuts for applying a desired transformations to selected subexpressions, it is painful for even expert users to force large expressions into anything near the form they would prefer – death by a thousand cuts and pastes.

Part of the pain is carefully reassembling a final result from independently transformed subexpressions, then carefully deleting the distracting debris of all the intermediate steps.

Even merely ordering commutative operands as desired is difficult or impractical in most computer algebra systems. For example, it is a constant irritant to be unable to transform a result such as $E = c^2m$ to $E = mc^2$ or, better yet, to prearrange that it will automatically be ordered as desired.

2.2 Including direct manipulation

Direct manipulation provides a complementary way that major computer algebra systems could make their user interfaces more helpful: With Milo [2] or Theorist [7] you could use the mouse to select a term or a factor, then drag and drop it to

- reorder terms and factors,

[1] The maximum number of legible characters simultaneously legible on multiple high-resolution screens or sheets of paper is unimprovable.

- distribute a term over a factor,
- factor out a common factor from a sum of terms,
- transpose factors or terms from one side of an equation to the other.

The selected subexpression could also be dropped into a variable in another expression to substitute the subexpression for every instance of the variable in that other expression. Also, expressions could optionally be compressed by automatic or mouse-driven temporary replacement of subexpressions with ellipses.

Here is a temporal sequence of Milo snapshots for dragging x successively further right in an equation [16]:

$$\frac{\boxed{\mathrm{x}}\mathrm{y}}{2}+\mathrm{x}=1 \quad\rightarrow\quad \frac{\mathrm{y}\boxed{\mathrm{x}}}{2}+\mathrm{x}=1 \quad\rightarrow\quad \frac{\mathrm{y}}{2}\boxed{\mathrm{x}}+\mathrm{x}=1 \quad\rightarrow\quad \left(\frac{\mathrm{y}}{2}+1\right)\boxed{\mathrm{x}}=1 \quad\rightarrow\quad \frac{\mathrm{y}}{2}+1=\frac{1}{\mathrm{x}}$$

Milo evolved to The Plotting Calculator, which is still available and supported [3]. The most recent version of Theorist is named LiveMath$^{\mathrm{tm}}$, which is also still available and supported [19]. Both are oriented toward mathematics through calculus, but direct manipulation should also be implemented in other computer algebra systems, integrated with the transformation wizard proposed here: After selecting subexpressions, one of the transformation options, if applicable, should be "drag and drop".

2.3 Collecting multiple alternatives

The wizard generates and displays the results of alternative transformations as the user explores a tree of successive applicable transformations. The user can accumulate any number of these alternative results into a list that is returned as the result if the user wants more than one. For example, as users interactively view alternative factored and partial fraction forms for a rational expression, they can indicate which ones they want included in a returned list of alternatives. This is inspired by Wolfram|Alpha [38], which automatically returns multiple alternative results. The difference here is that the user can participate in a more thorough exploration of the possible alternatives and select only those of interest.

2.4 Input-result pairs *versus* derivation steps *versus* replacement

Most industrial-strength computer algebra systems use only the *input-result* mode. This mode is particularly appropriate when the goal is to obtain good final results in as few steps as is practical.

Some mathematics education programs such as Mathpert$^{\mathrm{tm}}$ and the SMG application for some TI computer algebra products use the *derivation* mode wherein the input is transformed to a result by selecting successive subexpressions and choosing transformations from menus, with the annotated result of each step displayed beginning on a separate line. This mode is also often used in theorem proving software [29, 37]. This mode is also good for expository use by professionals when they want to explain in a presentation or publication how a result is derived. For example, the multi-step derivational style is used several times in this article.

Some programs such as The Graphing Calculator offer the *replacement* mode wherein selected transformations replace the selected subexpressions *in situ*. This mode has the advantage of conserving screen space by minimizing the amount of debris – at the expense of not being able to view the input and result simultaneously.

An industrial-strength computer-algebra system should offer all three modes. The following observations can justify allowing mixtures of all three modes in a single session window and name-space context:

- An input-result pair can be regarded as a one-step instance of the derivation mode.
- A one step *in situ* replacement could be labeled with a two-button setter bar such as 

that toggles between the two.
- A multi-step *in situ* replacement could have a slider bar between these two endpoints, and perhaps also a Play button that does a slide show or an animation.
- A right click could offer the option of changing previous computations between these three modes, such as
  - collapsing a derivation sequence to an input-result pair or to an *in situ* replacement,
  - expanding an input-result pair to a refinable derivation sequence that was automatically used to create it.

The model-view-controller paradigm is a good way to achieve this multi-view software design [36].

2.5 {Undo^m, redo^n}

Anyone who has used software with a well designed essentially unlimited undo-redo capability knows how aggravating it is to return to software that offers only one step of undo – perhaps with no redo. With current RAM capacities measured in gigabytes there is no excuse for this. Internet browsing has familiarized users with using the "←" and "→" buttons together with a drop-down browse history list to revisit easily throughout the tree of past web page visitations. The wizard can use the same techniques and temporarily save all recent closed dialog boxes for quick regeneration.

2.6 Dynamically created dialog boxes specialized for the example

The variety of mathematics examples is so great that general-purpose dialog boxes created when a computer algebra system is built would be unpleasantly cumbersome to use:

- They would have numerous distracting grayed-out controls.
- They would entail numerous subsidiary dialog boxes to accommodate all of the inapplicable entries without making each dialog box unreasonably large.
- They would contain lengthy or awkward wording such as "variable or variables" or "variable(s)" to correctly accommodate both singular and plural cases without distracting grammatical errors.

Thus custom dialog boxes specialized to the framed subexpression must be created at run time.

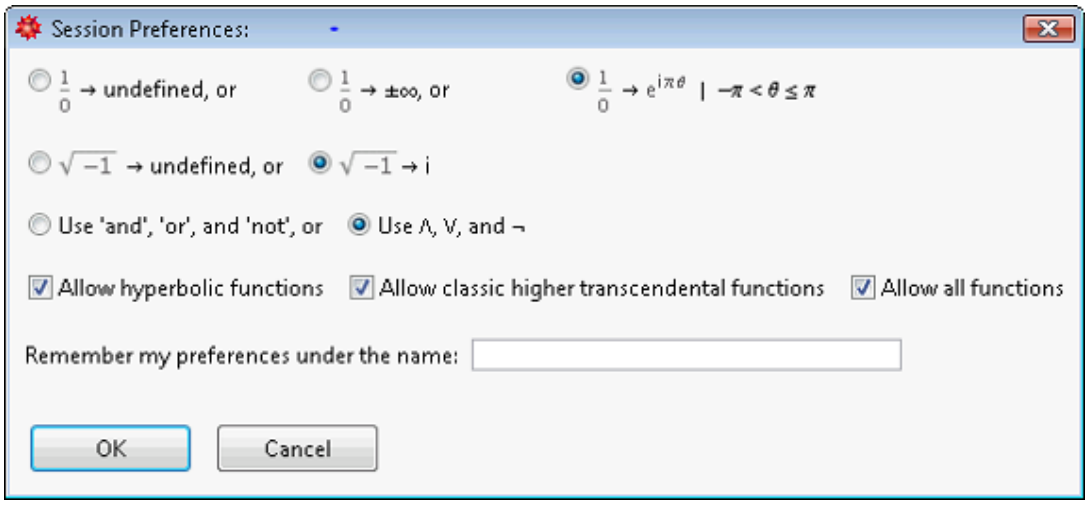
2.7 Adapt to the user's level and goals

Computer algebra is being used by students from beginning secondary school algebra through graduate-level mathematics – and by professional mathematicians, scientists, engineers, economists etc. at many different levels of sophistication. The number of potential users declines rapidly with increasing mathematics level. However, most computer algebra systems are designed for the higher levels of this spectrum. Consequently the most powerful general-purpose systems are quite daunting to most potential users. For example, in many courses from secondary school algebra through university real-variable calculus:

1. Many students know nothing about hyperbolic functions, higher transcendental functions and hypergeometric functions.
2. Most students have not encountered many standard mathematics symbols and notations such as $\exists$, $\forall$, $\neg$, $\vee$, $\wedge$, $\Re$, $\Im$, $\mathbb{Z}$ and $\mathbb{Q}$.
3. Most students know nothing about terminology such as algebraic groups, rings, fields, ideals, varieties, and square-free factorization.
4. The expression $1/0$ is usually or always regarded as undefined rather than as $\pm\infty$ or a circle of infinite radius in the complex plane.
5. The expression $\sqrt{-1}$ is usually or always regarded as undefined rather than i.
6. The expression $(-1)^{1/3}$ is usually taken to mean -1 rather than $1/2 + i\sqrt{3}/2$.

The mathematically weakest students and professionals who could most benefit from computer algebra are most intimidated by the appearance of such unknown function names, symbols and nomenclature in their dialog boxes and results. Often this intimidation and the consequent loss of self esteem terminates receptivity to learning effective use of the computer algebra system.

In computer aided instruction there are efforts to automatically infer the level and overall goals of users, then adapt the interface accordingly. Those techniques are not explored in this article, and the termination of receptivity might occur before enough input occurs to make an accurate inference. However, one easy way to accomplish many of the benefits of such customization is for the first dialog box of a session to have a button labeled "Session preferences" that opens a dialog such as the following if pressed:



The defaults should be those of the computer algebra system, but the more elementary alternatives should appear first in each row of alternatives to minimize alarming elementary users.

Another complementary alternative to matching student mathematical level is to enable the easy creation of named shell programs and their icons that launch the computer algebra system then immediately set appropriate preferences automatically. An instructor can then create such shells named, for example, MapleForAlgebra1 or MathematicaForCalculus1.

3 Examples of using the wizard

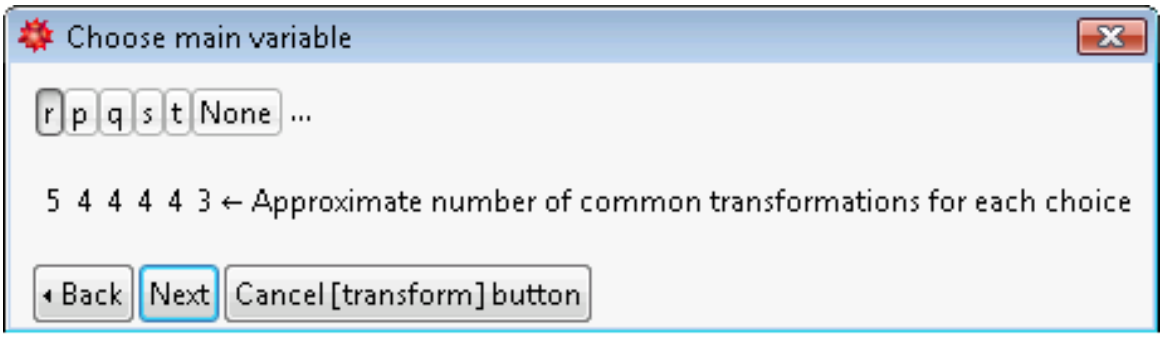
This section contains examples of using the proposed wizard.

3.1 Simplifying an ugly expression

The *Mathematica* CreateDialog[...] function is a convenient way to create new dialog boxes at run time. Thus I used CreateDialog[...] to create dialog boxes that are appropriate for specific examples, without bothering to attach these boxes to each other or to any *Mathematica* transformation functions. The mock intermediate and final results are not that of any particular current computer algebra system, but rather what I wish they would produce – especially with regard to ordering of factors and terms. For example, suppose that *either an input or a result* of previous steps is

$$\begin{gathered}(p^2qr^3+p^2qr^2s+p^2qr^2t+p^2qrst+p^2r^4+p^2r^3s+p^2r^3t+p^2r^2st+pqr^4+pqr^3s+pqr^3t+pqr^2st+\\ 2pqrs+2pqrt+4pqr+4pqst+2pqs+2pqt+pr^5+pr^4s+pr^4t+pr^3st+pr^2s+2pqrs+\\ 2pqrt+4pqr+4pqst+2pqs+2pqt+pr^5+pr^4s+pr^4t+pr^3st+pr^2s+pr^2t+2pr^2+\\ 2prst+2prs+2prt+2pr+2pst+ps+pt+qr^2s+qr^2t+2qr^2+2qrst+2qrs+2qrt+\\ 2qr+2qst+qs+qt+2r^2s+2r^2t+4r^2+4rst+2rs+2rt)/\\ (pqr^2+pqrs+pqrt+pqst+pr^3+pr^2s+pr^2t+prst+qr^3+qr^2s+qr^2t+qrst+r^4+r^3s+r^3t+r^2st).\end{gathered} \tag{1}$$

I have no *particular* goal form in mind, but I would like a result that is more concise and comprehensible – and more efficient for substitution of numbers. I position the mouse pointer between the left margin and the expression, thus framing the entire expression, then right click and choose "transform".[2] This opens the following dialog box courteously positioned just above the subexpression if the subexpression is low on the screen, or just below the subexpression otherwise[3]:

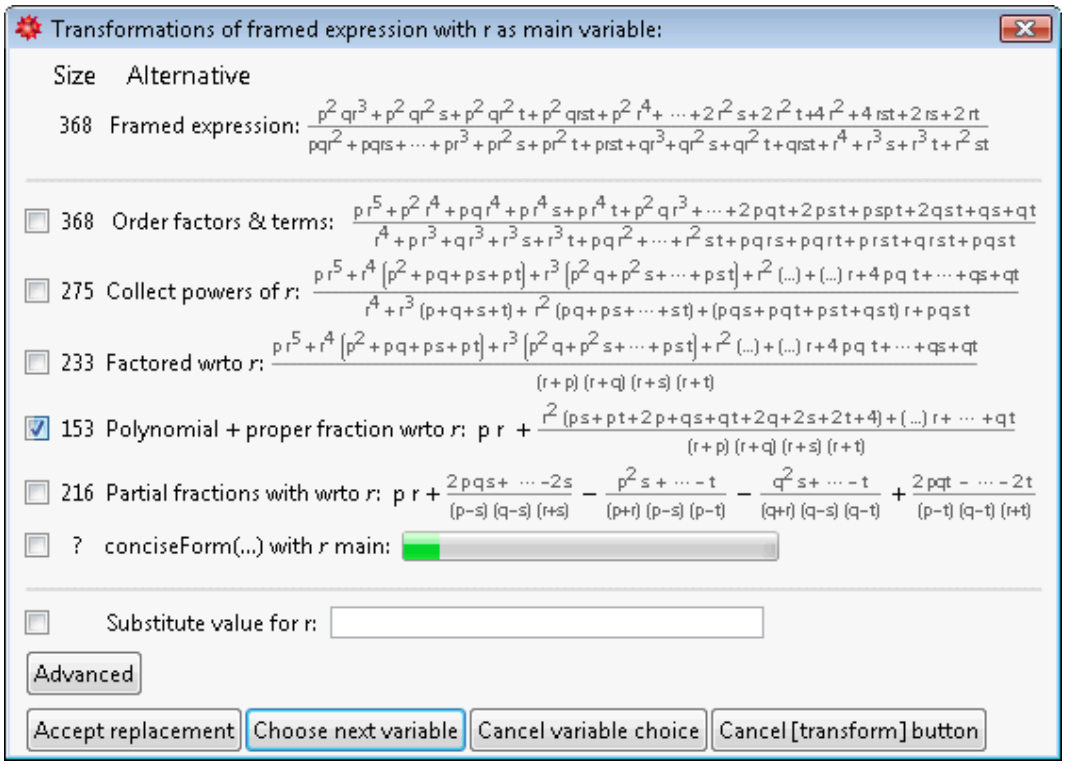


[2] Alternatively, I can choose "Transform ..." from the main menu bar or click Transform on the main toolbar.

[3] I hate it when a dialog box initially covers the information I need to respond to it!

Some transformations such as expansion of an improper ratio to a polynomial plus a proper ratio require a designated variable. The None button considers only transformations that do not require such a variable, such as factoring or *polynomial* expansion with respect to *all* variables. With that choice, the variables would be ordered according to the analysis in [20].

The variables are listed in non-increasing order of an estimated number of applicable common transformations because the choice of main variable tends to have the greatest influence on the overall form of the result. Choosing a main variable for which there are few alternative forms tends to narrow the choices more than otherwise. For example, if we chose p, q, s or t as the main variable, then there would probably be fewer than 5 common alternatives for r thereafter. Thus button r was initialized to *pressed* to encourage lazy users such as me to accept it, which I do. This opens the following dialog box:



"*A lot of times, people don't know what they want until you show it to them.*"
– Steve Jobs

The Advanced button lists alternatives that would interest most users only occasionally for this example – alternatives such as continued fractions, Hornerization, expression in terms of Chebyshev polynomials, or series approximations.

The displayed *sizes* are some easily computed measure that correlates approximately with the relative area that would be required to display the entire alternative results. The initially checked boxes are those having the smallest size.

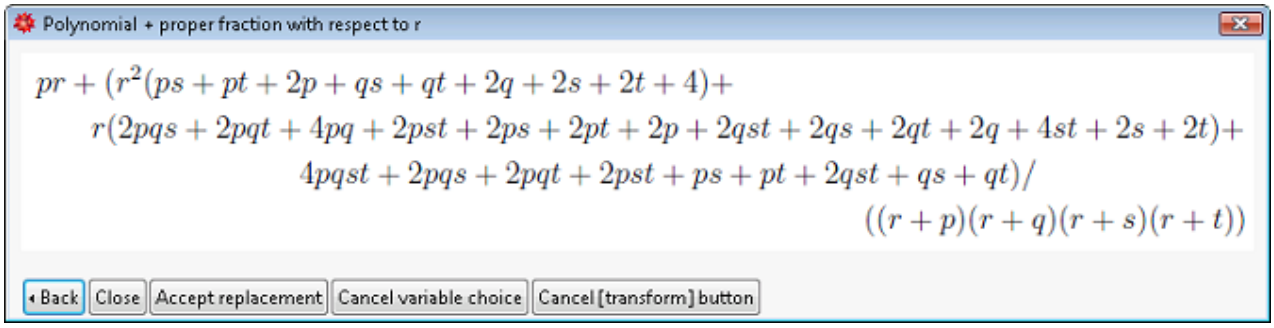
The dialog box shows alternative results for all the alternatives that can be computed in a *total* of at most 0.1 seconds – with elisions if necessary to avoid scroll bars or using the entire screen. User interface designers [23] feel that maximum acceptable response times are:

- about 0.1 seconds for responses to a mouse click, key press, or anything involving hand-eye coordination;
- about 1 second for opening a progress indicator, closing a dialog box or reformatting a table;
- about 10 seconds for everything else, including displaying a graph or completing an understandably time-consuming task. This "mind begins to wander" threshold is not always achievable with computer algebra.

The *Mathematica* Collect[. . . , r], Factor[. . .], PolynomialQuotient[. . . , . . . , r], PolynomialRemainder[. . . , . . . , r] and Apart[. . . , r] functions require a total of only 0.04 seconds on a dual-core 1.6 gigahertz computer to compute *Mathematica*-ordered versions of the five initially-displayed results in this dialog box. Therefore this dialog box could be created and displayed in an acceptable amount of time.

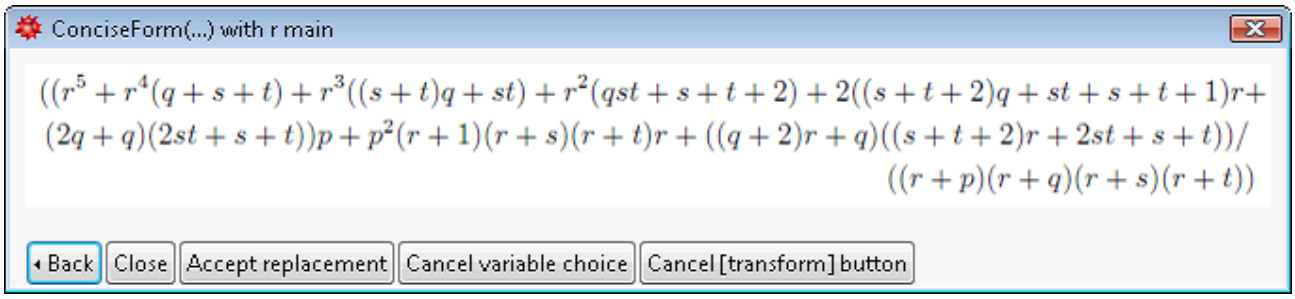
The progress bar for the conciseForm(. . .) alternative appears within one second, indicating that it is still being computed after the initial display of the dialog box. If that function doesn't post progress messages, then a slug cyclically moves from left to right to let the user know that the computer is working rather than merely awaiting user input.[4] The conciseForm(. . .) alternative represents the system's most powerful general-purpose simplification function such as the *Mathematica* FullSimplify[. . .] function, which requires 2.4 seconds for this example. This is way less than the time it requires for me to compare the five alternatives above conciseForm(. . .) with the Framed expression. As such continuing computations complete, their results are displayed in the dialog box – elided if necessary. They are listed below initially presented results to reduce displacement distraction when they complete. The conciseForm(. . .) alternative completes with a size of 125, which is the smallest, so that check box *also* becomes checked if I haven't already pushed a button.

Every time a check box is checked that has not been checked before, if the corresponding result is elided, then another dialog box opens that merely displays the un-elided alternative result, with scroll bars if necessary. For example, the initial "Polynomial + proper fraction wrt r" choice opens the dialog box



and when conciseForm(. . .) completes, it opens the dialog box

[4]It is not yet customary to have computer algebra functions post progress messages, but there are obvious candidate events for some algorithms. For example, many algorithms for degree n or for n variables, terms, factors, equations or columns process them one at a time, permitting messages of the form "$1/n$ % done", "$2/n$ % done", ... even if the time spent for each such step is likely to be rather uneven. People are comforted by progress bars even when they are inaccurate.



Both alternatives are significant improvements over the original framed expression (1), but both numerators are still lengthy with no easily discerned pattern. The reasons for the factored denominator in "Polynomial + proper fraction" are:

- The factored denominator was already computed for the alternative "Factored wrt r".
- The factored denominator is much more compact and informative than the fully expanded original denominator.
- There is nothing in the phrase "Polynomial + partial fraction" that promises displaying the denominator expanded with respect to r that was used to compute the polynomial part and the numerator.
- It is easy to frame the factored denominator then expand it if desired.

Although the ConciseForm result is slightly more compact, perhaps I could improve the numerator of the proper fraction result because I requested no more than a polynomial plus a proper fraction, and the quickest path to that goal was to expend no extra effort on the numerator beyond the collection with respect to the main variable r that was already done. Therefore in either the dialog box that contains the elided or the complete version of the proper fraction, I frame the entire numerator, right click, then choose "transform". This recursively opens up a new dialog box to choose a main variable, for which I again choose r for consistency. The resulting displayed alternatives include the factored numerator

$$(r(p+q+2)+2pq+p+q)\,(r(s+t+2)+2st+s+t)\,.$$

This is much more compact, with insightful symmetries $p \leftrightarrow q$, $s \leftrightarrow t$ and $[p,q] \leftrightarrow [s,t]$ that are also true of the denominator. Therefore I accept this replacement in this *sub-problem* that is an alteration of the "Polynomial + proper fraction" alternative, thus transforming the expression in that dialog box to

$$pr + \frac{(r(p+q+2)+2pq+p+q)\,(r(s+t+2)+2st+s+t)}{(r+p)(r+q)(r+s)(r+t)}. \tag{2}$$

This is the nicest overall result so far, but before accepting it, I notice that although every factor contains r, the first numerator factor and the first two denominator factors are free of s and t, whereas the second numerator factor and the last two denominator factors are free of p and q. From experience I know that for two ratios having disjoint variable sets or nearly so, common denominators almost always increase bulk because there can be very little cancellation in the resulting numerator.

Thus conversely, partitioning the ratio in (2) into a ratio containing $\{r, p, q\}$ and a ratio containing $\{r, s, t\}$ then transforming each ratio to partial fractions *might* reduce bulk. Consequently, I drag the first numerator factor left of the ratio giving

$$pr + (r(p+q+2) + 2pq + p + q)\,\frac{(r(s+t+2) + 2st + s + t)}{(r+p)(r+q)(r+s)(r+t)}.$$

Then I drag the first two denominator factors under the former numerator factor giving

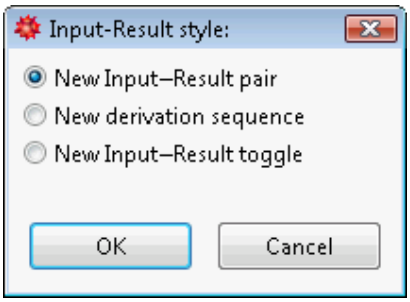
$$pr + \left(\frac{r(p+q+2) + 2pq + p + q}{(r+p)(r+q)}\right)\left(\frac{r(s+t+2) + 2st + s + t}{(r+s)(r+t)}\right). \tag{3}$$

(With the ability to select several non-adjacent subexpressions, I could instead select in (2) the first factor of the numerator and the first two factors of the denominator, then right click, then choose an action named something such as "collect", "group" or "isolate" – or perhaps directly choose partial fractions.)

Next I highlight the left ratio in (3), choose r as the main variable, then accept partial fraction expansion with respect to r, then do similarly for the right factor, giving

$$pr + \left(\frac{p+1}{r+p} + \frac{q+1}{r+q}\right)\left(\frac{s+1}{r+s} + \frac{t+1}{r+t}\right). \tag{4}$$

This is a very gratifying result compared to the equivalent input (1), and I am happy with the ordering of the terms and factors. Therefore I press the Accept result button, which opens the dialog



If I choose "New Input-Result pair", then the main computer algebra session window would be updated with a numbered input such as $Input_4$: Transform($Result_3, \ldots$) followed by expression (4) preceded by a label such as $Result_4$. The purpose of the Input line is to capture a programmatic way to transform $Result_3$ to $Result_4$ for purposes such as scripting. Most users will not want to view the ugly details, but if I click on the ellipsis in Transform($Result_3, \ldots$), then it expands to a procedure that generates $Result_4$, such as

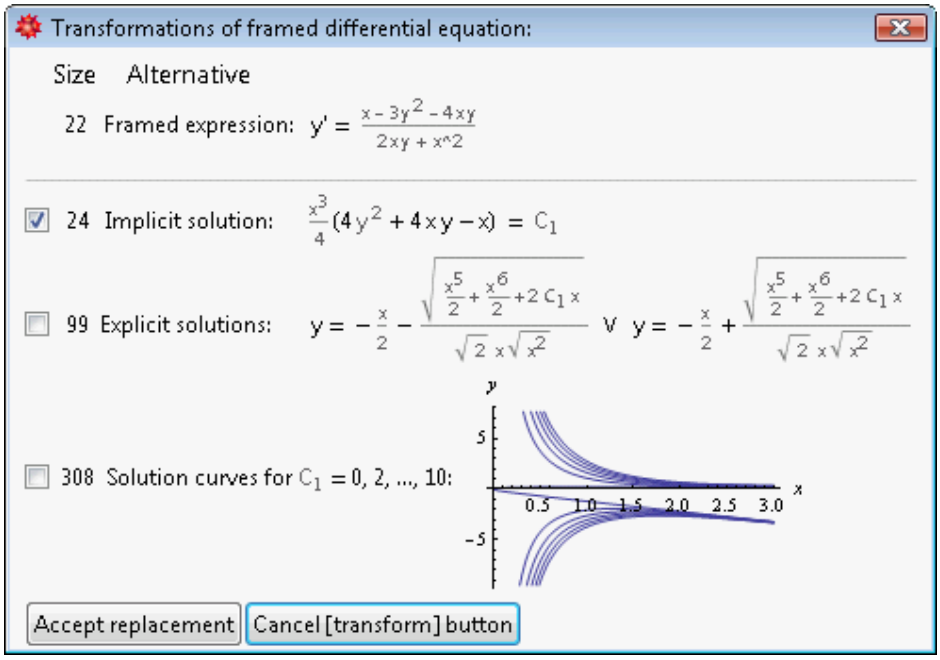
```
Block(Local(temp1,temp2,temp3,temp4),
  temp1 := PolynomialPlusProperFraction(Result3, r),
  temp2 := Factor(Numerator(Second(temp1)), r),
  temp3 := Factor(Denominator(Second(temp1)), r),
  First(temp1) +
    PartialFractions(First(temp2)/(First(temp3)*Second(temp3)), r)*
    PartialFractions(Second(temp2)/(Third(temp3)*Fourth(temp3)), r));
```

The "New derivation sequence" choice is similar, except it generates a collapsible sequence of such pairs using labels such as $\text{Intermediate}_{4,1}$, $\text{Intermediate}_{4,2}$, ...

The "New Input-Result toggle" is similar to the "New derivation sequence", except listing a single expression labeled with a two-button setter bar.

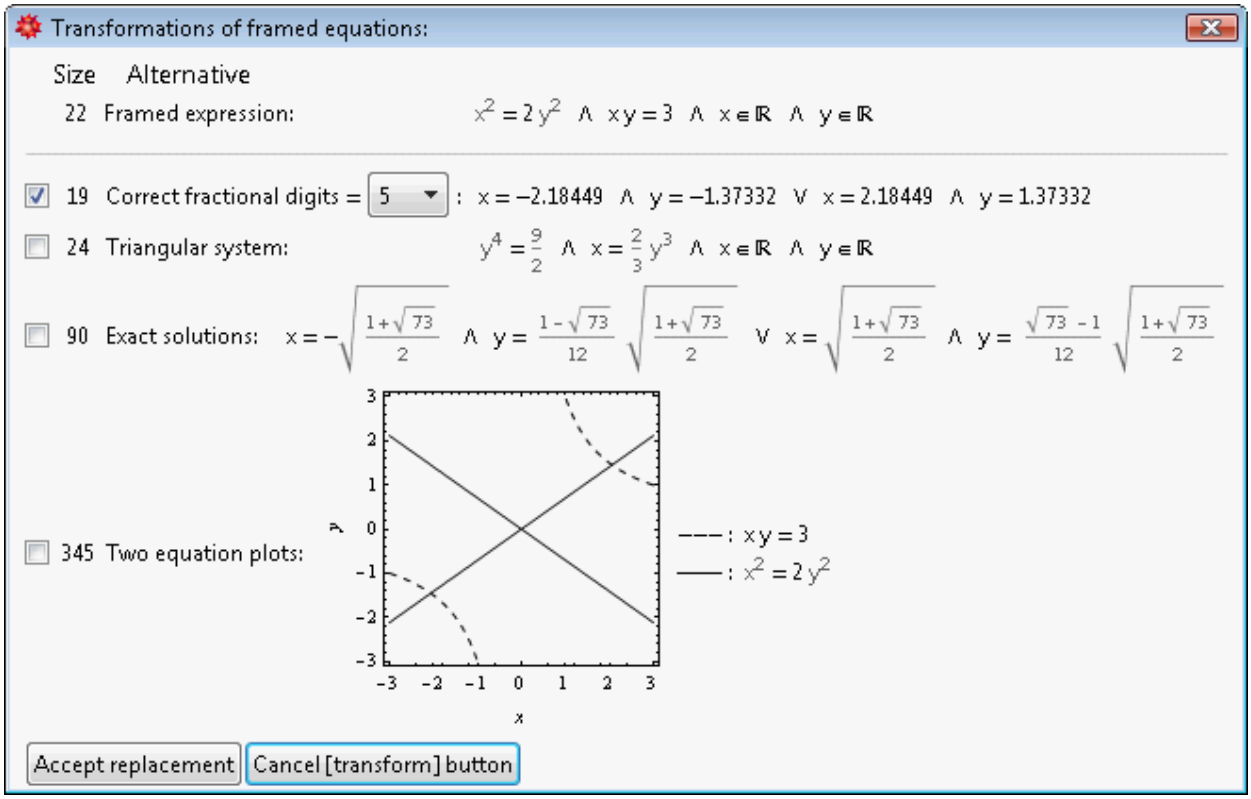
3.2 Transforming equations, inequalities and Boolean expressions

The primary activity that users want to do with equations, inequalities and systems thereof is to transform them into explicit solutions, so why force users to learn numerous different function names with different parameter semantics for solving different kinds of equations? If the user clicks the Transform button when an entire equation, inequality or system thereof is framed, then the wizard tries to return solutions. Here is an example for a differential equation:



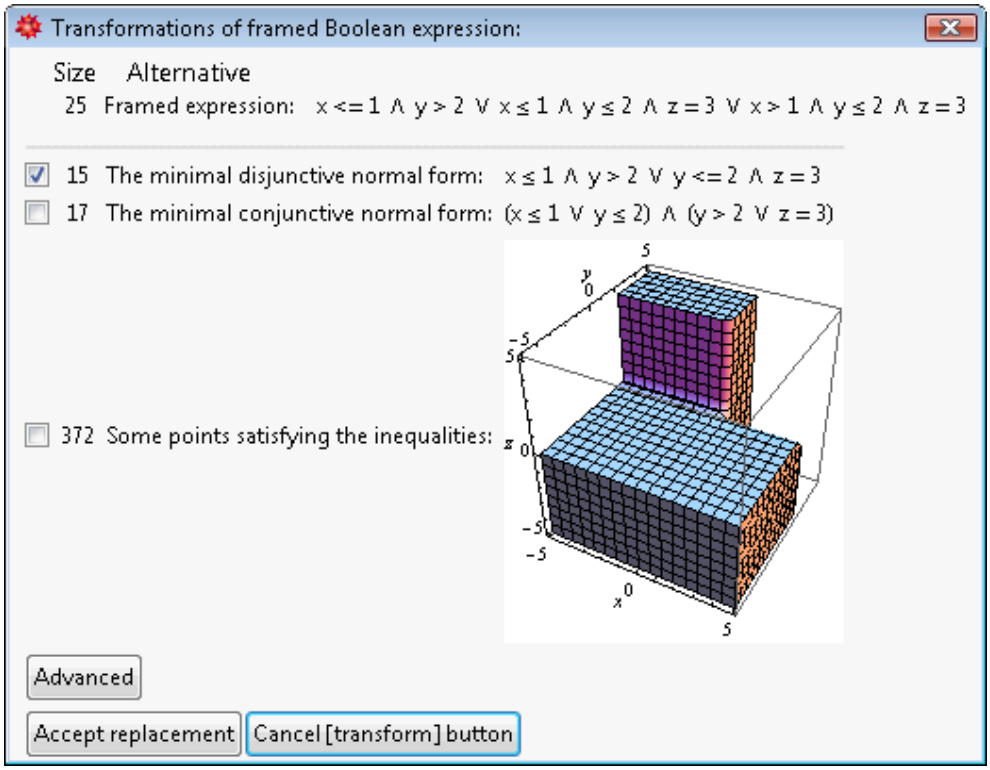
Notice the synergy of multiple views of the solutions. When the mouse pointer is over a curve, it is attached to a call-out displaying the corresponding explicit solution containing the associated numeric value for C_1. The plot range for x and the set of values for C_1 were chosen to insure that y is real. Checking this alternative opens a dialog with a magnified plot that gives the user control over the plot ranges for x and y together with the set of numeric values for C_1. The listed size of 308 is correlated with the area of the plot in the session window if accepted.

Here is an example for a system of two nonlinear algebraic equations:



- The approximate solution was computed with *adaptive precision interval arithmetic* to deliver guaranteed requested accuracy initially set to correspond to six significant digits for nonzero components. If interval arithmetic is inapplicable or is taking too long for the specified accuracy, then the wizard tries *adaptive significance arithmetic*. If that is also taking too long, the wizard switches the popup digits setting to "IEEE" double and uses that. The corresponding displayed phrases are "Estimated correct fractional digits" or "Approximate solutions of unknown accuracy".
- The triangularized system is a reduced lexicographic Gröbner basis, which might be the preferred alternative for parametrized systems having exact explicit solutions that are messy or require unendurable time to complete. If the system included inequalities, then there would be a cylindrical algebraic decomposition instead.
- The x and y ranges in the plot were automatically set to include a margin around the convex hull of all the isolated finite solutions – a margin small enough to resolve detail near the solutions but large enough to provide useful context.

Here is an example of transforming a Boolean expression:



The Advanced button lists alternatives expressed in terms of nands, nors, etc.

3.3 The wizard is helpful even for mere numbers

Common useful internal representations for exact numbers are rational numbers and irrational constant expressions such as $\sqrt{\pi} + \ln 2$. Common representations for approximate numbers are:

1. software variable-precision Floats – preferably with adaptive significance tracking,
2. IEEE double Floats to take advantage of fast hardware instructions;
3. intervals whose endpoints are each independently an exact rational number or a Float – preferably adaptive precision for floating-point endpoints, and allowing a disjoint union of such intervals with either open or closed endpoints.

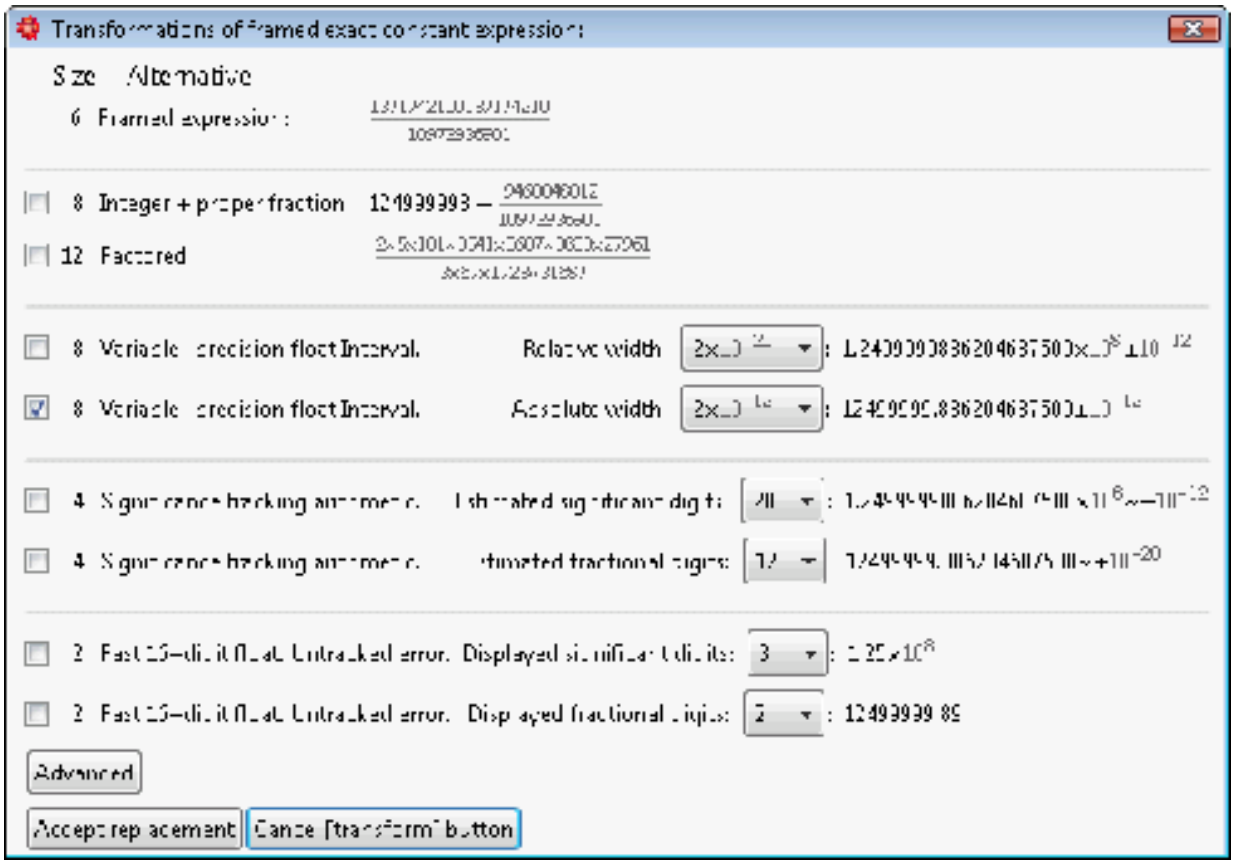
Before commencing a computation a user might want to transform from an exact to an approximate representation to make the computation faster – or from an approximate to an exact representation to avoid rounding errors. *After* a computation a user might want to convert from an approximate to an exact representation to attempt recovering an exact result – or from an exact to an approximate representation to make lengthy exact numbers more comprehensible or faster for purposes such as plotting. Also, at the end of a computation a user might want to simply alter the *display* of a number, such as displaying an exact rational number factored or as an integer plus a proper fraction or as a decimal fraction or in scientific form with a particular number of fractional or significant digits. The wizard makes it easy to do these transformations.

3.3.1 Alternate forms for rational numbers

If the framed subexpression is

$$\boxed{\frac{1371742100137174210}{10973936901}},$$

then the wizard could offer the dialog box



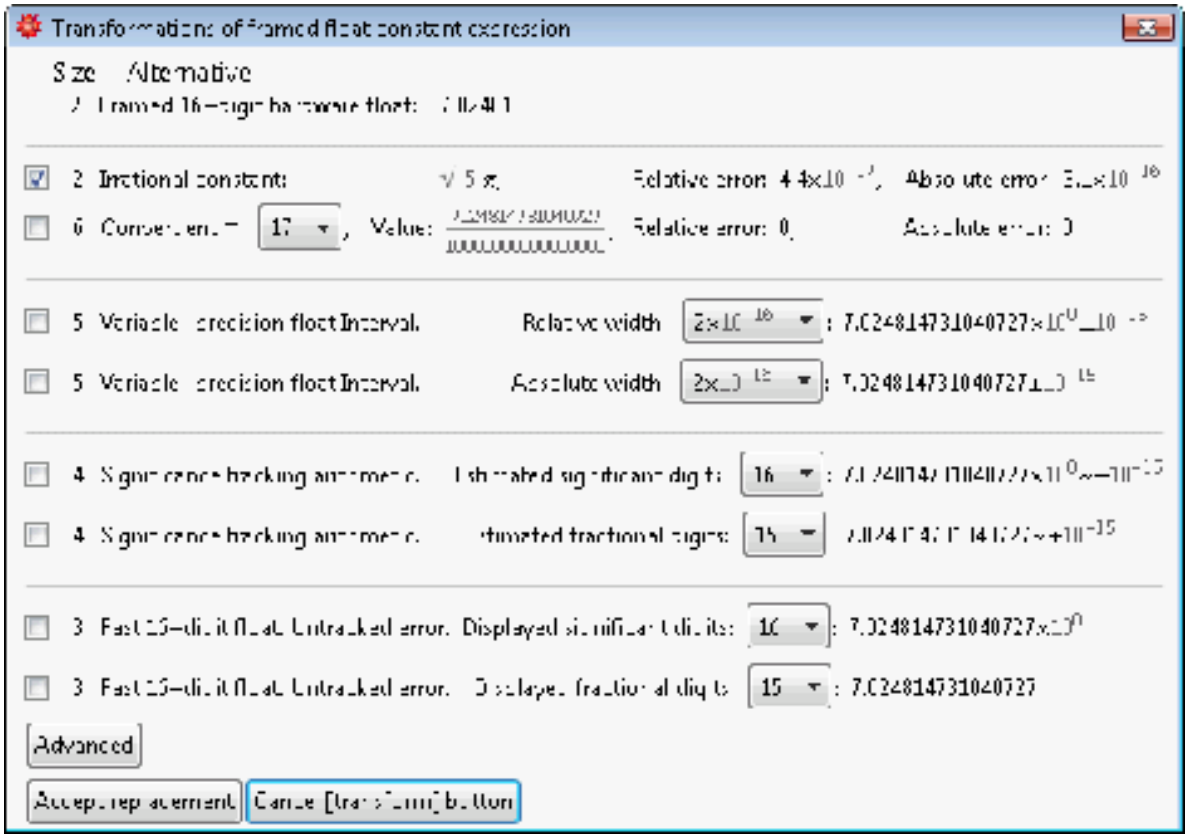
In accordance with the recommendations of [27], the approximate alternatives are ordered from intervals through bare IEEE Floats to encourage more use of arithmetic that is closer in spirit to exact arithmetic. The Advanced button could offer p-adic, continued fraction, and different radix representations.[5]

Notice that this dialog provides useful supplementary information about the framed number even if the user never intended to replace the framed number: The user now has a good estimate for its magnitude, can see that it is well approximated by 1.25×10^8, and that both the numerator and denominator are composite but square free.

[5]In most systems, default simplification would immediately transform a factored or continued fraction or integer plus proper fraction form of a rational number back to a reduced ratio. Thus special passivity is required to make such volatile forms appear in results, and such non-idempotent forms are quite likely to disappear when such results are used in subsequent inputs. For this reason, many systems return a factored rational number as a list of pairs of bases and exponents, etc. The wizard must correct for such impediments to direct substitution of transformed subexpressions into the expression from which it came, suppressing default simplification where necessary to preserve the displayed overall result in standard mathematical form.

3.3.2 Alternate forms for intervals and Floats

If the framed subexpression is an interval, an IEEE double or a variable-precision Float, then the wizard could offer alternatives including approximating the number with an exact rational or irrational constant. For example, if the framed number was the IEEE double 7.024814731040727 or a reasonably close approximation to it, albeit perhaps *displayed* with fewer significant digits, then the wizard could offer



Details matter. For example:

- The alternate expressions are aligned, if practical, to make it easier to compare them.
- If the framed Float displayed few digits, the initial displayed digits for alternatives displays all or many digits – and *vice versa*.

The delightful alternative $\sqrt{5}\,\pi$ was computed quickly by the Maple identify(...) function, for which there is a more powerful free stand-alone version on the internet [5].

3.3.3 Alternate forms for non-real numbers

For non-real numbers, which of *rectangular*, *unit polar* and *exponential polar* form is most attractive depends on the particular number. For example, compare

$$\begin{array}{ccccc} \textit{Rectangular} & & \textit{Unit polar} & & \textit{Exponential polar} \\ \boldsymbol{7+5i} & = & (-1)^{\arctan(5/7)/\pi} & = & \sqrt{74}\,e^{\arctan(5/7)\,i}, \\ -2\sin\left(\frac{3\pi}{14}\right)+2\cos\left(\frac{3\pi}{14}\right)i & = & \boldsymbol{2(-1)^{5/7}} & = & 2e^{5\pi i/7}, \\ 2\cos(1)+2\sin(1)\,i & = & 2\,(-1)^{1/\pi} & = & \boldsymbol{2e^{i}}. \end{array}$$

Default simplification would ideally *display* the form that is most concise for each non-real number in a result even if a different form is used internally. However, optional transformations can conveniently offer all three alternatives.

4 Design issues and their resolutions

Challenging design issues include deciding:

1. What set of transformations should the wizard consider?
2. How can the wizard quickly estimate the number of applicable transformations without knowing the next variable choice?
3. When the next variable choice is known or None, how can the wizard quickly determine what subset of transformations are applicable and schedule them so that a worthwhile number are completed quickly?

This section addresses these issues and a few others. However, there are so many transformations that the wizard should know about that this section concentrates on those that help resolve issues 2 and 3. For more completeness, the Appendix discusses additional transformations that are relevant to the rational aspects of expressions. Transformations of the *irrational* aspects of expressions is too large a topic for treatment in this article.

First, three simple definitions:

Definition. *Default simplification* is the result of pressing the Enter key or else perhaps Shift Enter with the factory-default mode settings and no transformational or simplification functions anywhere in the input expression.

Definition. A *functional form* is an expression of the form

$$f\,(\textit{expression}_1,\ \textit{expression}_2,\ \ldots)$$

where f is any function name.

Definition. *Generalized variables* are the smallest subtrees of an expression tree that are not a sum, difference, product, ratio, rational number, Float, or reasonably regardable as an integer power.

For example, z, π, i, $\cos(x+f(2))$, $z^{1/5}$, and $3^{1/5}$ are generalized variables. In contrast, $3/4$, x/y, and $x-3$ are not. Also, $z^{2/5}$ and $3^{2/5}$ are not generalized variables, because they can be regarded as $(z^{1/5})^2$ and $(3^{1/5})^2$.

The importance of generalized variables is that transformations that are applicable with respect to variables in an expression can also be applicable with respect to generalized variables in an expression. For example, a user might want ordering, expansion or factoring with respect to to π and or $\cos(x+f(2))$. Additional transformations might be applicable with respect to generalized variables that are not merely indeterminates, such as $\cos(x+f(2)) \to \sin(\pi/2-x-f(2))$, $\cos(x+f(2))^2 \to 1-\sin(x+f(2))^2$, or $\pi \to 3.14159$.

As a prerequisite to discussing the wizard, it is helpful to organize the most important optional transformations offered by most computer algebra systems into categories of related transformations. For simplicity, the discussion addresses only constant ground domains that are common scalar numeric domains of characteristic 0. However, much of the discussion is relevant to other ground domains such as $\mathbb{Z}_m$ or {true, false}.

4.1 Different transformations for different generalized variables

"*To each his own.*"
– Cicero

"*I got different strokes for different folks.*"
– Muhammad Ali

Often users want certain transformations such as expansion or factoring only with respect to certain variables. For example:

- To compute the integral of $\left(x^5+(c+1)^{999}x+1\right)^2$ with respect to x, it is helpful to expand with respect to x, but foolish to expand with respect to c.
- To solve
$$\left(c^{999}-1\right)\left(z^2-1\right)=0 \quad | \quad c^{999}\neq 1$$
it is helpful to factor with respect to z, but foolish to factor with respect to c.

In these cases we would prefer either concise or mere default simplification with respect to c.

When the user requests successive transformation for successive variables, we do not want to destroy transformations done for prior variables. Consequently, requested transformations are automatically mapped into the largest subexpressions that do not contain variables that have already been treated. For example:

1. If the alternative for *expanding* a framed expression with respect to the chosen main variable x is
$$\left(y^2+2y+1\right)x^2-y^2+2y-1,$$
then *factoring* this alternative with respect to y gives $(y+1)^2x^2-(y-1)^2$ rather than
$$((y+1)x+y-1)\left((y+1)x-y+1\right).$$

2. If the alternative for *factoring* a framed expression with respect to the chosen main variable x is
$$(y-1)(y+1)\left((y-2)(y+2)x+(z+1)^2\right)\left(x+(2y+1)(2y-1)\right),$$
then *expanding* this alternative with respect to y gives
$$(y^2-1)\left((y^2-4)x+(z+1)^2\right)\left(x+4y^2-1\right).$$

3. If the alternative for *factoring* a framed expression with respect to the chosen main variable x is
$$\left(y^2-1\right)\left((y^2-4)x+(z+1)^2\right)\left(x+4y^2-1\right),$$
then *factoring* this alternative with respect to y gives
$$(y-1)(y+1)\left((y-2)(y+2)x+(z+1)^2\right)\left(x+(2y-1)(2y+1)\right).$$
(Notice that the wizard factored not only the *coefficients* of powers of x with respect to y, including the coefficient of the zeroth power of x, but also the top-level *content* y^2-1, because none of these contain x.)

4. If the alternative for *expanding* a framed expression with respect to the chosen main variable x is
$$\left((z+1)^9y^2+y+3\right)x^2+(z+1)^9x+(y+1)(y-1),$$
then *expanding* this alternative with respect to y gives two distinct alternatives: *distributed* form
$$(z+1)^9x^2y^2+yx^2+3x^2+(z+1)^9x+y^2-1,$$
and the often more concise *recursive* form
$$\left((z+1)^9y^2+y+3\right)x^2+(z+1)^9x+y^2-1,$$
both of which are offered to the user. Expansion of $(z+1)^9$ will be offered if the user proceeds to that last remaining variable rather than balking or accepting an alternative already displayed.

Now consider the input $\sin(x)\left(\cos(2y)+1\right)\cos(x)$. If the user is allowed to choose trigonometric expansion of multiple angles for the generalized variable $\cos(2y)$ but choose the opposite transformation for $\sin(x)$ and $\cos(x)$, then this product can transform to the particularly concise equivalent $\sin(2x)\cos(y)^2$ because $\cos(2y)\equiv 2(\cos(y)^2-1)$ and $\sin(x)\cos(x)\rightarrow\sin(2x)/2$.

Thus for maximum flexibility:

1. The user should be able to choose the order of generalized variables.
2. The user should be able to choose separate transformations for each generalized variable.
3. The choices for each variable should include conciseForm(...) and mere reordering with any associated default simplification when such results differ from the framed subexpression.
4. Where there is expansion with respect to two or more successive variables, both distributed and recursive forms should be offered if they are not identical.

4.2 Control over the order of generalized variables

"*Order is the shape upon which beauty depends.*"
– Pearl S. Buck

Subsection 4.1 discussed how collection of similar powers of a generalized variable can be recursively applied to the resulting collected coefficients to perform transformations for successive generalized variables in any order. However, current computer algebra systems would nonetheless impose their built-in ordering rules to the resulting factors and terms. Therefore the displayed ordering of factors in terms and of terms in multinomials might not correspond to the recursive most main to least main order in which the user has treated successive variables. For example after requesting expansion with respect to y with coefficients that are factored with respect to all other variables a user might obtain a result such as

$$x^3 + y^3(z+1)^2 + y^2 z$$

rather than the more appropriate result

$$(z+1)^2 y^3 + z y^2 + x^3.$$

Also, regardless of the requested order, Newton's definition of force might be displayed as

$$f = am \tag{5}$$

which is visually quite disturbing despite its compliance with the usual alphabetical ordering convention for variables in a monomial. Consequently:

Optional transformations should include control over the displayed ordering of factors and terms.

The few systems that give such control tend to do so indirectly and incompletely via control over the ordering of generalized variables. For example:

- In the Maxima computer algebra system the desired traditional order displayed in result equation (5) can be accomplished by a declaration such as ordergreat(a, m) or orderless(m, a). If the user has issued an ordergreat(...) and a non-conflicting orderless(...) declaration, then all other variables order between the least of the great and the greatest of the least, alphabetically. The effect is global from the time of a declaration until all declared orders are deleted with an unorder() declaration, which *must* be used between any two ordergreat(...) declarations or between any two orderless(...) declarations. Maxima also provides another mechanism for overriding default alphabetical ordering:

  declare (*variable*$_1$, *property*$_1$, *variable*$_2$, *property*$_2$, ...)

  gives each variable the corresponding property. Possible properties include *constant*, *scalar*, and *mainvar*. The command remprop(*variable*, *property*) can be used to remove such a declaration. Using $\prec$ to represent "less main",

  $$\textit{constants} \prec \textit{scalars} \prec \textit{undeclared} \prec \textit{mainvars}.$$

  By default, alphabetical order is used within each of these categories.

- The Reduce computer algebra system has an order declaration that is similar to ordergreat(...), except that more than one cumulative order declaration is allowed before a declaration

  order nil;

  which clears all such ordering declarations. The Reduce order declaration also accepts functional forms and built-in literal constants such as π, which is important.

- These and some other systems provide some control over the ordering of special distributed polynomials for Gröbner bases, but that is not very helpful for controlling the ordering of factors and terms in *general* expressions.

4.3 Common alternate forms for the *rational* aspect of expressions

Many computer algebra systems have separate functions for common denominators, various factorization levels, and various levels of polynomial or partial fractions expansions. This subsection describes how, for any particular ordering of generalized variables, these traditionally disparate concepts can be organized into a single topologically sorted list of partially-ordered alternatives varying from the most complete commonly-named factorization through the most complete commonly-named expansion offered by many computer algebra systems. This *organizing principle* greatly simplifies the transformation wizard by preventing selection of a set of contradictory transformations and by making the trade-off consequences in this list more obvious.

This subsection concerns only addition, subtraction, multiplication, division and integer powers, but most of the ideas also apply recursively to rational compositions of generalized variables and to fractional powers. Moreover, this subsection discusses only factoring, common denominators and expansion because they are most relevant to estimating quickly how many transformations are applicable for each variable and for quickly determining exactly which common transformations are applicable for a particular variable. The Appendix discusses additional rational transformations.

4.3.1 Reasons for common denominators, factoring and expansion

Definition. A **candid expression** is one that is not equivalent to an expression that visibly manifests a simpler expression class [24] .

As counterexamples:

- The expression $x(y+1) - xy$ is *not* candid because it contains the superfluous variable y.
- The expression $(x+1)^2 - x^2$ is *not* candid because it appears to be quadratic but is actually linear.
- The expression $(x+1)/(x^2+2x+1)$ is *not* candid because it is equivalent to $1/(x+1)$, which has lower numerator and denominator degrees.

For most computer algebra systems, any amount of factoring *includes* reduction over a common denominator. Reduction over a common denominator yields a candid form for rational expressions, because the resulting form has no superfluous variables and has maximum possible cancellation of poles with coincident zeros.

4.3.2 A *univariate* partially-ordered set of factorization and expansion levels

1. For univariate factoring there are names for certain amounts of exact or approximate factoring based on multiplicities and the desired numeric coefficient domain of the factors:
   (a) *term primitive*,[6]
   (b) *square free*,
   (c) *over the integers* $\mathbb{Z}$,
   (d) *over the Gaussian integers* $\mathbb{Z}[i]$,
   (e) *over particular algebraic extensions*[7]
   (f) *exact reasonably absolute*,[8]
   (g) *exact absolute*,[9]
   (h) *approximate absolute over the floating-point real numbers* $\tilde{\mathbb{R}}$,
   (i) *approximate absolute over the floating-point complex numbers* $\tilde{\mathbb{C}}$.[10]

2. For systems that support variable precision Floats, users can choose the precision level for alternatives (h) and (i). For systems that offer significance and/or interval arithmetic, those alternatives immediately before (h) for real numbers and before (i) for non-real numbers.

[6]The *term content* of a univariate polynomial is the gcd of the numeric coefficients times the smallest occurring power of the variable. Factorization into the term content times the *term primitive part* forces a common denominator if any coefficient has a denominator, because with polynomials A, B, C and D,

$$\frac{A}{B}+\frac{C}{D} \equiv AB^{-1}+CD^{-1} \quad \rightarrow \quad (AD+BC)B^{-1}D^{-1} \equiv \frac{AD+BC}{BD} \quad \rightarrow \quad \frac{(AD+BC)/G}{(BD)/G}$$

where $G \leftarrow \gcd(AD+BC,\, BD)$.

[7]As a convenience in *Mathematica*, Factor[*expression*, Extension → Automatic] automatically uses extensions implied by the complex unit i and/or any radicals *present* in *expression*. For example,

$$\text{Factor}[x^2+x-2+\sqrt{2},\ \text{Extension} \rightarrow \text{Automatic}] \quad \rightarrow \quad -\left(-x-1+\sqrt{2}\right)\cdot\left(x+\sqrt{2}\right)$$

However, $\text{Factor}[x^2+2\cdot\sqrt{2}\cdot x-1,\ \text{Extension} \rightarrow \text{Automatic}] \not\rightarrow \left(x+\sqrt{2}-\sqrt{3}\right)\cdot\left(x+\sqrt{2}+\sqrt{3}\right)$ because $\sqrt{3}$ is not in the given polynomial. We must instead use $\text{Factor}[x^2+2\cdot\sqrt{2}\cdot x-1,\ \text{Extension} \rightarrow \{\sqrt{2},\ \sqrt{3}\}]$ to obtain this factorization, but how many users would know to include $\sqrt{3}$?

[8]This is what is usually expected of algebra through calculus students for purposes such as solving equations or integrating rational functions: Algebraic extensions implied by radicals in the input together with use of the quadratic formula and n^{th} roots to factor binomials. *Derive* offers this option but also includes cubic and quartic formulas, which tends to generate unreasonably messy factorizations.

[9]This means whatever algebraic extension is necessary to factor the polynomial as much as possible, without the extension being provided by the user. Reference [9] discusses some algorithms for this. Some systems appear to use absolute factorization in their functions that solve systems of polynomial equations and integrate, but unfortunately appear not to offer it as a built-in factorization option. Therefore many computer algebra systems cannot directly factor x^2+2x-1 into $(x+1+\sqrt{2})(x+1-\sqrt{2})$ without assistance, which any beginning algebra student can do!.

Attempted exact absolute factorization might consume an intolerable amount of computing time, or resulting factors might entail intolerably messy nested radicals or intolerably messy subexpressions containing functional forms named something such as Root. This is why I list the *exact reasonably absolute* level of factorization.

[10]Alternatives (h) and (i) are *approximations* rather than equivalence transformations. Some methods for exact absolute factorization begin from an approximate absolute factorization that is often preferable to the resulting messy exact factorization!

3. With $F_1 \succeq F_2$ denoting the fact that for a given example, a factorization at level F_1 is either identical to a factorization at level F_2 or is a further splitting of the factorization at level F_2, we have

$$\mathbb{Z}[i] \succeq \mathbb{Z} \succeq \text{square free} \succeq \text{term primitive},$$
$$\text{exact absolute} \succeq \text{specific algebraic extensions} \succeq \mathbb{Z},$$
$$\text{exact absolute} \succeq \text{reasonably exact absolute} \succeq \mathbb{Z},$$
$$\tilde{\mathbb{C}} \succeq \tilde{\mathbb{R}} \succeq \mathbb{Z},$$
$$\tilde{\mathbb{C}} \succeq \mathbb{Z}[i].$$

   Thus these factorization levels form a *directed acyclic graph.* that we can topologically sort into one of several alternative lists, such as order 1(a) through 1(i) above.[11]

4. If we fully expand the product of the numerator factors and the product of the denominator factors of a reduced ratio, then we have the reduced ratio of two fully expanded polynomials. Despite the common denominator, the result is an expanded polynomial when this reduced denominator is 1 or when both the numerator and denominator are numeric. Therefore this form is on the borderline between factored and expanded.

5. The computer algebra built-into Texas Instruments hand held, Windows and Macintosh products has a function propFrac(*expression*, *variable*) that expands *expresssion* into a expanded polynomial with respect to *variable* plus a reduced ratio of two polynomials that is *proper* with respect to *variable*. The propFrac(...) function can easily be implemented using a polynomial quotient and remainder function, and the resulting form is an appropriate next node in our partial ordering from most factored to most expanded. This form is often more concise than either a common denominator or a partial fraction expansion. For example, this form was a key intermediate step in the example of subsection 3.1. For canonicality:

   (a) The coefficients of the resulting expanded univariate polynomial part that are not complex Floats can be normalized to Gaussian rationals $\mathbb{Q}[i]$ or rationalized algebraic numbers.

   (b) The denominator of the proper ratio can be made unit normal as described in [25].

   (c) The numeric coefficients in the resulting proper ratio that are not complex Floats can be normalized to Gaussian integers or algebraic integers.

   (d) If all of the denominator numeric coefficients are complex Floats, then we can normalize their magnitudes – such as making the largest of the real and imaginary magnitudes in the denominator coefficients be 1.0.

6. Polynomial expansion can be regarded as a special case of propFrac(...) for when the denominator of the given reduced ratio is numeric – perhaps 1.

7. If the reduced ratio of two polynomials has a non-numeric denominator, then the relevant adjective phrases 1(a) through 1(i) above can be used to label successive nodes corresponding to the amount of denominator factorization for corresponding partial fraction expansions.

[11] Actually, different specific algebraic extensions generally form a directed acyclic subgraph because, for example, we could have any one, two or all three of the extensions $\sqrt{2}$, $\sqrt{3}$ and $\sqrt{5}$, giving more than one path to $\{\sqrt{2}, \sqrt{3}, \sqrt{5}\}$. These directed acyclic subgraphs are the field extension lattices of Galois theory.

8. However, the square-free aspect of partial fraction expansion has two variants in the partial ordering. In non-decreasing order of the amount of expansion, adjective phrases applicable to the square free aspect are:
   (a) *incomplete*, meaning multiples of all powers of the same square-free denominator factor are combined over a common denominator, and
   (b) *complete*, meaning instead that for each resulting *very proper* ratio $N(x)/D(x)^m$ with expansion variable x, $\deg_x(N(x)) < \deg_x(D(x))$.[12]

Whenever a resulting numerator has more than one term, we can distribute an associated denominator over the numerator terms. It is generally unwise to distribute a multinomial denominator over the numerator terms for purposes such as integration, and it almost always increases bulk. However, it is helpful to do such a distribution for purposes such as fragmenting a ratio into the greatest number of simplest possible pieces for angle sum expansion.[13] The wizard can display both alternatives when they are not identical.

Table 1 shows the named alternative forms for a univariate example of the reduced ratio of two expanded polynomials.

1. The first two rows and last two rows are approximations to all of the other rows, which are equivalent to each other.
2. The double line separating "ratio of expanded polynomials" and "polynomial + proper ratio" separates factored from expanded forms.
3. Factors that differ from those of the preceding row are boldface.
4. Wherever there is a sum in a numerator, the corresponding denominator can optionally be distributed over the terms of the numerator.
5. For each of these named levels an example can be constructed where it is more concise than all of the other named levels. Therefore all of the levels are important.[14]
6. If a system doesn't offer built-in support for all of these named levels and a wizard implementer is not inclined to add such support, then:
   (a) Missing intermediate factorization levels can be provided by over-factoring then expanding appropriate subsets of factors.
   (b) Missing expansion levels can be provided by over expanding then combining appropriate subsets of summands.
7. The input could be any of these expressions or any rational expression that is equivalent to one of these expressions. If an input contains Floats, then float-free alternatives can be obtained by using a function such as the Maple identify(...) function to determine close rational or irrational constants [5].

[12] In contrast, for the *incomplete* square-free partial fraction expansion we can guarantee only that $\deg_x(N(x)) < \deg_x(D(x)^m)$. Many systems offer only complete expansions, but incomplete expansions are adequate for most purposes and are often more concise!

[13] If you must distribute multinomial denominators over numerator terms, it is most efficient to wait until after the expansion is complete in other respects.

[14] There are also unnamed intermediate levels such as splitting some but not all of the square-free factors over $\mathbb{Z}$.

Table 1: A univariate expression partially ordered from most factored to expanded

Amount of factor or expand	Boldface parts are different from the alternative above them
$\tilde{\mathbb{C}}$	$\frac{1.5(z-1.37)(z+5.7)(z-1.07+0.76i)(z-1.07+0.76i)\cdots(z+1.09+0.18i)(z+1.09-0.18i)}{z(z-1)^2(z+i)(z-i)(z+1.414)(z-1.414)(z+1.13)(z-1.04+0.82i)(z-1.04-0.82i)\cdots}$
$\tilde{\mathbb{R}}$	$\frac{1.5\cdots\mathbf{(z^2-2.13z+1.71)(z^2-1.13z+0.9)(z^2-0.05z+0.24)\cdots(z^2+2.19z+1.23)}}{z(z-1)^2\mathbf{(z^2+1)}(z+1.414)(z-1.414)(z+1.13)\mathbf{(z^2-2.08z+1.76)(z^2+0.95z+1.5)}}$
reasonably absolute ($\mathbb{Z}\left[i,\sqrt{2}\right]$)	$\frac{\mathbf{3(z^{12}+4z^{11}-9z^{10}+4z^9+z^8+13z^6-29z^5+7z^4+13z^3-25z^2+6z-6)}}{\mathbf{2}z(z-1)^2\mathbf{(z+i)(z-i)(z+\sqrt{2})(z-\sqrt{2})(z^5+z+3)}}$
$\mathbb{Z}[i]$	$\frac{3(z^{12}+4z^{11}-9z^{10}+4z^9+z^8+13z^6-29z^5+7z^4+13z^3-25z^2+6z-6)}{2z(z-1)^2(z+i)(z-i)\mathbf{(z^2-2)}(z^5+z+3)}$
$\mathbb{Z}$	$\frac{3(z^{12}+4z^{11}-9z^{10}+4z^9+z^8+13z^6-29z^5+7z^4+13z^3-25z^2+6z-6)}{2z(z-1)^2\mathbf{(z^2+1)}(z^2-2)(z^5+z+3)}$
square free	$\frac{3(z^{12}+4z^{11}-9z^{10}+4z^9+z^8+13z^6-29z^5+7z^4+13z^3-25z^2+6z-6)}{2z(z-1)^2\mathbf{(z^9-z^7-z^5+3z^4-z^3-3z^2-2z-6)}}$
term primitive	$\frac{3(z^{12}+4z^{11}-9z^{10}+4z^9+z^8+13z^6-29z^5+7z^4+13z^3-25z^2+6z-6)}{2z\mathbf{(z^{11}-2z^{10}+2z^8-2z^7+5z^6-8z^5+2z^4+3z^3-5z^2+10z-6)}}$
ratio of expanded polynomials	$\frac{\mathbf{3z^{12}+12z^{11}-27z^{10}+12z^9+3z^8+39z^6-87z^5+21z^4+39z^3-75z^2+18z-18}}{\mathbf{2z^{12}-4z^{11}+4z^9-4z^8+10z^7-16z^6+4z^5+6z^4-10z^3+20z^2-12z}}$
polynomial + proper ratio	$\frac{3}{2}+\frac{\mathbf{18z^{11}-27z^{10}+6z^9+9z^8-15z^7+63z^6-93z^5+12z^4+54z^3-105z^2+36z-18}}{2z^{12}-4z^{11}+4z^9-4z^8+10z^7-16z^6+4z^5+6z^4-10z^3+20z^2-12z}$
term primitive partial fraction	$\frac{3}{2}+\frac{\mathbf{3}}{\mathbf{2z}}+\frac{\mathbf{15z^{10}-21z^9+6z^8+3z^7-9z^6+48z^5-69z^4+6z^3+45z^2-90z+6}}{\mathbf{2z^{11}-4z^{10}+4z^8-4z^7+10z^6-16z^5+44^5+6z^3-10z^2+20z-12}}$
incomplete square free part frac	$\frac{3}{2}+\frac{3}{2z}+\frac{\mathbf{3z+3}}{\mathbf{(z-1)^2}}+\frac{\mathbf{12z^8-3z^6-3z^4+36z^3-18z+24}}{\mathbf{2z^9-2z^7-2z^5+6z^4-2z^3-6z^2-4z-12}}$
complete square free part frac	$\frac{3}{2}+\frac{3}{2z}+\frac{\mathbf{3}}{\mathbf{(z-1)^2}}+\frac{\mathbf{3}}{\mathbf{2z-2}}+\frac{12z^8-3z^6-3z^4+36z^3-18z+24}{2z^9-2z^7-2z^5+6z^4-2z^3-6z^2-4z-12}$
partial fractions over $\mathbb{Z}$	$\frac{3}{2}+\frac{3}{2z}+\frac{3}{(z-1)^2}+\frac{3}{2z-2}+\frac{\mathbf{3z}}{\mathbf{z^2-2}}+\frac{\mathbf{3z}}{\mathbf{z^2+1}}+\frac{\mathbf{3z^2-12}}{\mathbf{2z^5+2z+6}}$
partial fractions over $\mathbb{Z}[i]$	$\frac{3}{2}+\frac{3}{2z}+\frac{3}{(z-1)^2}+\frac{3}{2z-2}+\frac{3z}{z^2-2}+\frac{\mathbf{3z}}{\mathbf{2z+2i}}+\frac{\mathbf{3}}{\mathbf{2z-2i}}+\frac{3z^2-12}{2z^5+2z+6}$
absolute partial fractions	$\frac{3}{2}+\frac{3}{2z}+\frac{3}{(z-1)^2}+\frac{3}{2z-2}+\frac{\mathbf{3}}{\mathbf{2z+2\sqrt{2}}}+\frac{\mathbf{3}}{\mathbf{2z-2\sqrt{2}}}+\frac{3z}{2z+2i}+\frac{3}{2z-2i}+\frac{3z^2-12}{2z^5+2z+6}$
partial fraction over $\tilde{\mathbb{R}}$	$\cdots+\frac{\mathbf{1.5}}{\mathbf{z+1.41}}+\frac{\mathbf{1.5}}{\mathbf{z-1.41}}+\frac{\mathbf{3.0}z}{\mathbf{z^2+1}}+\frac{\mathbf{0.162}}{\mathbf{z+1.1}}-\frac{\mathbf{0.18z-0.27}}{\mathbf{z^2-2.1z+1.8}}+\frac{\mathbf{0.018z+0.31}}{\mathbf{z^2+0.95z+1.5}}$
partial fraction over $\tilde{\mathbb{C}}$	$1.5+\cdots-\frac{\mathbf{0.089+0.049}i}{\mathbf{z-1.04+0.82}i}-\frac{\mathbf{0.089-0.049}i}{\mathbf{z-1.04-0.82}i}+\frac{\mathbf{0.0088+0.13}i}{\mathbf{z+0.48+1.13}i}+\frac{\mathbf{0.0088-0.13}i}{\mathbf{z+0.48-1.13}i}$

4.3.3 *Multivariate* partially-ordered sets of factorization and expansion levels

Reference [26] describes how to do multivariate partial fraction expansion with respect to two or more successive generalized variables. As a degenerate case, the expansion is *polynomial* expansion with respect to variables that don't occur in the reduced common denominator of the given expression.

As shown that article, the number of terms in a multivariate partial fraction expansion can depend on the ordering of the expansion variables. Table 2 shows eight alternatives for factoring and or expanding a bivariate example over $\mathbb{Z}$. Notice how the partial fraction expansion with respect

to y in the last two rows introduces poles at $x = \pm 1$ into some individual ratios, forcing the use of a piecewise result to avoid contracting the domain of definition. Making a ratio proper can also cause this. (Unfortunately, most current computer algebra systems quietly do such domain reductions.) Common denominators remove these additively removable singularities.

Table 2: Some alternative recursive form factorizations and expansions over $\mathbb{Z}$.
f = **f**actored. e = **e**xpanded to incomplete partial fractions.

1st	2nd	Result
f_x	f_y	$\frac{(y-1)(y+1)(y^2+3)x^3-y(y+1)(y^4-y^3+y^2-3y-4)x-2y^2(y-1)(y^2+2)}{(x-y)(x+y)(y-1)(y+1)(y^2+2)}$
f_x	e_y	$\frac{(y^4+2y^2-3)x^3-(y^6-2y^3-7y^2-4y)x-(2y^5-2y^4+4y^3-4y^2)}{(x-y)(x+y)(y^4+y^2-2)}$
f_y	f_x	$\frac{xy^6+2y^5-(x^3+2)y^4-2(x-2)y^3-(2x^3+7x+4)y^2-4xy+3x^2}{(y-x)(y+x)(y-1)(y+1)(y^2+2)}$
f_y	e_x	$\frac{xy^6+2y^5-(x^3+2)y^4-(2x-4)y^3-(2x^3+7x+4)y^2-4xy+3x^2}{(y-x)(y+x)(y-1)(y+1)(y^2+2)}$
e_x	f_y	$\frac{y^3+3}{y^2+2}x + \frac{2y^2}{(x+y)(y-1)(y+1)} + \frac{2y}{(x-y)(y-1)(y+1)}$
e_x	e_y	$x + \frac{x}{y^2+2} + \frac{2}{x+y} + \frac{1}{xy-x+y^2-y} + \frac{1}{xy+x+y^2+y} + \frac{1}{xy-x-y^2+y} + \frac{1}{xy+x-y^2-y}$
e_y	f_x	$\begin{cases} 1 - \frac{y+8}{6(y-1)^2} - \frac{11y+8}{6(y+1)^2} + \frac{1}{3(y^2+2)}, & \text{if } x = -1, \\ -1 + \frac{1}{(y-1)^2} - \frac{2y+1}{(y+1)^2} - \frac{1}{y^2+2}, & \text{if } x = 1, \\ x + \frac{x}{y^2+2} + \frac{2x}{(x-1)(x+1)(x-y)} - \frac{2x^2}{(x-1)(x+1)(y+x)} + \frac{2x}{(x-1)(x+1)(y-1)} - \frac{2}{(x-1)(x+1)(y+1)}, & \text{otherwise} \end{cases}$
e_y	e_x	$\begin{cases} 1 - \frac{y+8}{6(y-1)^2} - \frac{11y+8}{6(y+1)^2} + \frac{1}{3(y^2+2)}, & \text{if } x = -1, \\ -1 + \frac{1}{(y-1)^2} - \frac{2y+1}{(y+1)^2} - \frac{1}{y^2+2}, & \text{if } x = 1, \\ \cdots + \frac{2}{y+x} - \frac{1}{xy+y+x^2+x} + \frac{1}{xy-y+x^2-x} - \frac{1}{xy+y-x^2-x} - \frac{1}{xy-y-x^2+x} + \cdots - \frac{1}{xy-y+x-1} + \frac{1}{xy-y-x+1}, & \text{otherwise} \end{cases}$

4.4 Series and other approximations

The discussion so far has been about transformations to alternatives that are *equivalent* to the input wherever the input is defined – except perhaps approximating exact numbers in the input with approximate Floats or approximating Floats in the input with nearby rational numbers. This subsection instead addresses the equally important alternatives of transformations that *approximate* the input with *simpler expressions*.

Closed-form exact results aren't always obtainable. Even when they are, the results might be too bulky to convey needed insight or to permit fast enough well-conditioned evaluation for numerous floating-point values of the variables therein. Therefore, various kinds of approximation are useful transformations. Also it is important to realize that the ultimate destiny of many exact expressions is to substitute Floats into them, in which case the resulting rounding errors might exceed those caused by an approximate expression. Here are some examples of appropriate optional approximate transformations for a wizard to offer:

- Quadrature can often be used to determine a single approximate number for a definite integral.

- Approximate equation solving is often preferable even when compact explicit exact solutions are obtainable.
- Generalized infinite or truncated Laurent-Puiseux series (allowing, for example, logarithmic factors) can concisely approximate lengthy expressions. The wizard can initialize expansion points to ones that are most likely of interest, such as 0, infinities, and poles. If selected, the user can adjust the expansion points and the requested order.
- Padé approximations often have a larger region of convergence and greater computational efficiency than power series.
- Truncated Fourier or wavelet series are often more appropriate than expansions about a point.

A well-chosen approximation can be simpler than any obtainable exact result and yet retain all of the important characteristics of an exact result.

4.5 Generating the list of generalized variables

If a variable v in the framed subexpression has an assigned value, then it would be misleading to list that irrelevant v. However, we do want to consider listing some or all of the generalized variables in the *assigned value*, if any. We can use default simplification for this purpose, because it replaces all assigned variables with their values.

On many systems *default* simplification can easily produce results containing generalized variables that candid simplification would eliminate. For example, the default simplification of most systems merely reorders the factors and/or terms in the input $((x+1)x - x^2 - x + 2)\,/\,(y^2-1)$, but all of the factoring and expansion transformations described in sub-subsections 4.3.2 and 4.3.3 transform the expression to $2/(y^2-1)^2$ or $2/\left((y-1)(y+1)\right.$ or $1/(y-1) + 1/(y+1)$, which all depend on y alone.

It is helpful for the wizard to recognize such *superfluous* generalized variables. For polynomial expressions, expansion to recursive form always eliminates superfluous variables. For other rational expressions, reduction over a common denominator always does so.[15] Thus the first thing that the wizard can do is compute such a form to identify generalized variables that thereby disappear. One of these transformations can be initially checked to help encourage the user to eliminate superfluous generalized variables. However, the user might prefer to eliminate them in a way that alters cherished structure less drastically. To do this, the wizard could also offer the alternative, for example, "merely eliminate superfluous x and $\ln(y)$". This can be accomplished by substituting simple exact constants such as 0, 1 or -1 for those generalized variables, then applying default simplification. If the substitution value is at a removable singularity and thereby causes division by 0, then the wizard can backtrack and try another simple constant.

A generalized variable wouldn't be offered if doesn't affect ordering and no optional transformation is applicable to it. For example, it is pointless to list x if it only occurs as the argument in $\sin(x)$. However, it might be worthwhile to list the generalized variable $\sin(x)$, depending on how it occurs in the default-simplified result. As another example, *none* of the transformations or ordering choices being discussed here are applicable to the subexpression $3x^2$.

There will usually be an initial "Choose main variable" dialog if there is more than one applicable generalized variable. If so, and after choosing the transformation with respect to the selected main

[15] If this reduction produces $0/0$, then the expression is undefined for all values of its variables, so the one and only offered alternative can be the result of "0/0".

variable there is still more than one applicable generalized variable for some maximal subexpression that doesn't contain the main variable, then there will be another dialog labeled instead "Choose next most main variable", and so on until the user aborts the investigation or accepts replacements.

4.6 Estimating the number of common transformations for each generalized variable choice

Quick syntactic checks can identify some opportunities for transforming a framed subexpression with respect to a generalized variable. For example,

- Expansion is applicable to any variable that occurs in a multinomial raised to an integer power or a multinomial multiplied by any subexpression.
- A common denominator is applicable to any variable that occurs to a negative power in a sum.
- Angle sum expansion is applicable if the argument of a sinusoid is a sum.
- The transformations $\sin(u)^2 \to 1 - \cos(u)^2$ or $\sin(u)^n \to (\sin(nu) + \cdots)/2^n$ are applicable if $\sin(u)$ is raised to any integer power exceeding 1.

Quick syntactic checks can also *preclude* some opportunities for transforming a framed subexpression with respect to a generalized variable. For example, expansion is precluded if the framed subexpression is monomial or linear in the variable.

Another ordering heuristic is that the number of applicable transformations is likely to increase with the degree of a variable in a numerator or denominator, and more transformations are associated with high degree denominators than high degree numerators, because denominator factorizations can also enable partial fraction expansions.

If time remains in the 0.1 second acceptable delay for generating and displaying the first dialog box, then we can compute more accurate estimates for the number of common applicable transformations for each variable: For the rational aspects of a framed subexpression, minimal-effort reduction over a common denominator reveals the numerator and denominator degrees of every top-level generalized variable, which are all of those that can be affected by top-level factoring or expansion. Minimal effort here means using partially factored form – preferably recursive – as described in [24]. If this reduced partially factored form differs from the framed subexpression, then it is already one applicable transformation. Moreover, expansion to a polynomial plus a proper fraction is applicable to every such generalized variable whose degree in the resulting numerator is at least as large as the degree in the denominator; and it is easy to compute these degrees for partially factored forms.

If time still remains in the 0.1 second acceptable delay, then we can compute more accurate estimates by factoring the common denominator with respect to all of its generalized variables. From this it is easy to determine which successively lesser factorizations would combine two or more factors containing that main variable and thus determine the number of distinct named partial fraction expansions with respect to that variable and a lower bound on the number of distinct named factorizations over a common denominator.

If time still remains, then the wizard can factor the numerator to better estimate the number of named factorizations over a common denominator.

To reduce the chance of exceeding 0.1 second before *any* factorization is achieved, it could be done in levels, such as term content with respect to every variable, then square free, etc. through, say, exact reasonably absolute factorization followed by absolute factoring over the complex Floats. If the allotted 0.1 seconds runs out before completing this agenda, then we simply use the best estimates that we have at that time. Moreover, we can continue to refine and update the estimates after the initial display, without changing the initial order of the generalized variable buttons.

4.7 Recognizing applicable transformations

The exact and approximate factorizations with respect to all variables is a useful point of departure for computing alternative results regardless of what variable the user chooses. Therefore it is helpful to proceed computing those factorizations in the background while users ponder their choice of variable.

After a user chooses a variable, the wizard can complete the distinct factorization levels by appropriately expanding pairs of factors, which is fast. To convey the strongest possible information, when labeling the displayed alternatives or elided versions thereof, they are labeled with the most thorough applicable factoring level. For example, if the only distinct factorizations are factorizations over the Gaussian integers and over the integers, then the later would not be labeled "square-free" even though it is that as well. If there is only one factored alternative, then it could be labeled merely "factored" for brevity, but with a more detailed phrase displayed upon mouse-over. This provides a learning opportunity for mathematically unsophisticated users.

If there is a denominator and it contains the chosen variable, the wizard can then proceed to compute the polynomial part and proper fraction, followed by any distinct partial fraction levels with respect to that variable.

5 If I want this interface, why haven't I implemented it?

Good question. The reasons are:

- I am not an interface programmer.
- It is best done by a team.
- Some aspects will require access to proprietary internals that are inaccessible to outsiders for systems that aren't open source.
- It will require testing feedback from numerous users. Surely some of the ideas presented here won't work out well in practice, and better ideas will occur.
- If it isn't the default interface distributed with a computer algebra system, then it is unlikely to be known to most users, and the system is likely to evolve in a way so that the alternative interface no longer works.
- For various reasons, corporations are often unwilling to adopt and maintain packages written by outsiders as first class parts of their system – thoroughly and seamlessly integrated into the system and the documentation with no need for building or loading.

Therefore, although I would be delighted to help, the purpose of this manifesto is to stimulate computer algebra users to request better interfaces and stimulate decision makers to build them.

"*If you build it, they will come.*"
– an apt misquote from *Field of Dreams*

Computer algebra users of the world: the squeaky wheels get the grease!

Acknowledgments

I thank Bill Gosper for information about Lisp Machine Macsyma and Norbert Kajler for helpful suggestions. Jacques Carette provided so many extraordinarily good suggestions that he should be a co-author – except for conflict of interest that he is an unmasked referee!

Appendix: More transformations for rational expressions

It is worthwhile to list in one place most of the many known transformations that might be of general interest. Subsection 4.3 discussed factoring, common denominators and expansion. Here are some additional transformations for the rational aspect of expressions:

A.1: Basic – of interest to most users

The wizard should automatically try the following transformations in the background because they can dramatically decrease the bulk of an expression and reveal important structure:

Polynomial shifts
Most of the factorization and expansion levels leave polynomial sub-expressions that often have more than two terms. Sometimes merely re-expressing such a multinomial in terms of optimally shifted variables can greatly reduce the number of terms and/or the size of coefficients. For example,

$$\left(y^2-4y+4\right)x^3+\left(3y^2-12y+12\right)x^2+\left(3y^2-12y+12\right)x+y^2-4y+1 \;\;\rightarrow\;\; (y-2)^2(x+1)^3+8,$$

$$531441\,x^6+2834352\,x^5+6298560\,x^4+7464960\,x^3+4976640\,x^2+1769472\,x+262151 \rightarrow (9x+8)^6+7\,.$$

Articles [13, 14], give algorithms for computing optimal shifts. As a quick preclusion test, it is not worth shifting a polynomial with respect to a variable if the polynomial is monomial or binomial with respect to that variable.

Polynomial decomposition
Complementary to such shift decompositions, Kozen and Landau [17] describe an efficient algorithm for completely decomposing a univariate polynomial $p(x)$ into nested polynomials

$$p_1\left(p_2\left(\ldots\left(p_m\left(x\right)\right)\ldots\right)\right)$$

with each $p_k\left(t\right)$ of degree at least 2 in t. For example, the irreducible polynomial

$$\begin{aligned} P(x) &:= x^{12}+4x^{10}+x^9+6x^8+3x^7+4x^6+3x^5+x^4+x^2+7 \\ &\rightarrow \left(x^3+x\right)^4+\left(x^3+x\right)^3+7. \end{aligned}$$

Their algorithm also applies recursively to multivariate polynomials represented recursively. As a quick preclusion test, such decompositions are inapplicable with respect to a variable of degree less

than 4 or having few terms. There are also algorithms for other kinds of univariate and multivariate polynomial decompositions, as described, for example, in [12, 34, 33, 32, 35].

Polynomials rewritten in these ways can reveal significant structure, help precondition an expression for efficient repeated numeric evaluation or reduced rounding error, and facilitate solutions of higher-degree polynomial equations or systems of equations. For example, with the above decomposition, masochists could apply the quartic formula then the cubic formula to express the zeros of $P(x)$ in terms of radicals.

Linear combinations of powers

At least since Pythagoras, people have been interested in representing numbers and non-numeric expressions as sums, differences, or general linear combinations of powers of other expressions. For example, if an expression can be rewritten as a sum of even powers of real subexpressions, then the expression is thereby proven to be nonnegative for all real values of these subexpressions. Algorithms for such transformations can be found in, for example, [21, 22, 31].

A.2: Advanced – of interest only to experts

Here are some transformations that should be tried only in response to the Advanced button because they would probably intimidate and distract most users without any compensating benefit to them.

Expression in terms of orthogonal polynomials

Important optional transformations include a change of basis from monomials to orthogonal polynomials, which can

- yield more concise results,
- yield results less subject to magnified rounding errors,
- reveal patterns that otherwise wouldn't be apparent, or
- suggest efficient accurate min-max truncated approximations.

Many computer algebra systems provide functions that return one of various classic orthogonal polynomials of a specified degree in a specified variable using the monomial basis. However, most of the systems provide little or no automation for:

- converting ordinary polynomials to linear combinations of orthogonal polynomials specified by symbols such as the Chebyshev polynomials of the first kind $T_0(z)$, $T_1(z)$, ... ;
- propagating linear combinations of such polynomials into other linear combinations thereof *exactly* through rules such as

$$\begin{aligned} T_m(z)\,T_n(z) &\rightarrow \frac{1}{2}\,T_{|n-m|}(z) + \frac{1}{2}\,T_{m+n}(z), \\ T_m\left(T_n(z)\right) &\rightarrow T_{mn}(z), \\ \int T_n(z)\,dz &\rightarrow \begin{cases} \frac{1}{4}\left(T_0(x) + T_2(x)\right), & \text{if } n = 1, \\ \frac{1}{2-2n}\,T_{n-1}(z) + \frac{1}{2+2n}\,T_{n+1}(z), & \text{otherwise,} \end{cases} \end{aligned}$$

  and *approximately* through infinite or truncated series expansions;

- efficiently and accurately substituting Floats into expressions involving combinations of such functions: Rather than substituting a number z_0 for z in the monomial-basis representations of the symbols $T_0(z)$, $T_1(z)$, ... in a combination thereof, it is much faster and more accurate to compute the successive $T_k(z_0)$ from the recurrence $T_n(z_0) \leftarrow 2z_0\,T_{n-1}(z_0) - T_{n-2}(z_0)$.

Of the various named orthogonal polynomials, $T_0(z)$, $T_1(z)$, ... are probably most important. Therefore, Trefethen and others [30] developed a powerful MATLAB package that automates effective use of these polynomials for many applications, using IEEE double Floats to represent the coefficients. Fateman [11] implemented some of these capabilities in Maxima to take advantage of its variable precision floating point and exact rational arithmetic. Such transformations and analogous ones for other orthogonal polynomials would be a welcome addition to many computer algebra systems.

Expression in terms of symmetric polynomials

Analogous transformations for the most common kinds of *symmetric polynomials* would be another welcome addition. Such polynomials can help reduce the curse of dimensionality for problems that exhibit symmetries when pairs of variables are interchanged. As a start toward this, the *Mathematica* function SymmetricReduction [*expression*, $\{v_1, v_2, \ldots v_n\}$, $\{s_1, s_2, \ldots, s_n\}$] returns the pair $\{p, r\}$ where *expression* depends on variables $\{v_1, v_2, \ldots v_n\}$, p is the symmetric component of *expression* in terms of symbols $\{s_1, s_2, \ldots, s_n\}$ representing the *elementary symmetric polynomials* through degree n, and r is the residual of *expression* that can't be so represented. To convert a symmetric result back to the original variables, function SymmetricPolynomial$[m, \{v_1, v_2, \ldots v_n\}]$, returns the m^{th} elementary symmetric polynomial using variables $\{v_1, v_2, \ldots v_n\}$. Sturmfels [28] contains algorithms for transforming to and from symmetric polynomials.

Rational Decomposition

We can attempt separate polynomial decompositions on every numerator and denominator polynomial in an expression. However, there are also algorithms for decomposing *ratios* of polynomials into *nested ratios*, as discussed in [4, 15, 39]. As an example from the first of these articles, but using recursive form and primitive normalization, the ratio

$$\frac{(y^2 + 2z^2y + z^4 - 81)\,x^2 - 2y\cdot(y^5 + z^2y^4 + 225\,z)\,x + y^2\,(y^8 - 625\,z^2)}{(y^2 + 2z^2y + z^4 - 162)\,x^2 - 2y\cdot(y^5 + z^2y^4 + 450\,z)\,x + y^2\,(y^8 - 1250\,z^2)}$$

$$\rightarrow \begin{cases} 1, & \text{if } 9x + 25zy = 0, \\ \dfrac{\left(\dfrac{(y+z^2)\,x - y^5}{9x + 25\,zy}\right)^2 - 1}{\left(\dfrac{(y+z^2)\,x - y^5}{9x + 25\,zy}\right)^2 - 2}, & \text{otherwise.} \end{cases}$$

This example also illustrates that rational decomposition can introduce new removable singularities in the nested form, such as on the manifold $9x + 25zy = 0$ for this example. We can avoid this by clearing the nested denominators but preserving the nested polynomial components thereof to obtain *correlated* polynomial decompositions of the numerator and denominator:

$$\frac{\left(\dfrac{(y+z^2)\,x - y^5}{9x + 25zy}\right)^2 - 1}{\left(\dfrac{(y+z^2)\,x - y^5}{9x + 25zy}\right)^2 - 2} \;\rightarrow\; \frac{((y+z^2)\,x - y^5)^2 - (9x + 25\,zy)^2}{((y+z^2)\,x - y^5)^2 - 2\,(9x + 25\,zy)^2}.$$

If desired, for this example we can then further factor the difference in two squares in the numerator and in the denominator to obtain the factorization over $Z[\sqrt{2}\,]$.

$$\frac{\left(\left(y+z^2+9\right)x-y^5+25\,zy\right)\left(\left(y+z^2-9\right)x-y^5-25\,zy\right)}{\left(\left(y+z^2+9\sqrt{2}\right)x-y^5+25\sqrt{2}\,zy\right)\left(\left(y+z^2-9\sqrt{2}\right)x-y^5-25\sqrt{2}\,zy\right)}.$$

The numerator factorization could easily have been computed from the original numerator. However, the required $\sqrt{2}$ algebraic extension necessary to factor the multivariate denominator would be more difficult to determine without the intervening rational decomposition.

Continued fractions

Continued fractions are another type of compound-fraction representation for rational expressions. Acton [1] lists three different variants of continued fractions together with algorithms for converting between them and a reduced ratio. Cuyt and Verdonk [10] review methods for multivariate continued fractions. As an example of a continued fraction expansion that reveals a simple pattern:

$$\frac{\left(z^4-105\,z^2+945\right)z}{15\left(z^4-28\,z^2+63\right)}\;|\;z^2\notin\left\{63,\;\frac{45}{2},\;\frac{3}{2}\left(35\pm\sqrt{805}\right)\right\}\rightarrow\cfrac{z}{1-\cfrac{z^2}{3-\cfrac{z^2}{5-\cfrac{z^2}{7-\cfrac{z^2}{9}}}}}.$$

Here a constraint was appended to the input to avoid the *appearance* of contracting the domain of definition because of removable singularities introduced by the continued fraction. If instead we used a piecewise result, then it would have 9 pieces, 8 of which are constants that can be determined by substituting the two square roots of each element in the constraint set into the original expression.

Actually, if the computer algebra system automatically handles unsigned zeros and infinities correctly, then with exact computation the continued fraction form evaluates to the correct finite values even at these removable singularities. However, unlike the reduced ratio, the continued fraction form *might* be less accurate due to catastrophic cancellation near those introduced singularities. Therefore it is worth alerting the user to these singularities by either appending an input constraint or producing a piecewise result.[16]

Hornerized forms

In its simplest form, Horner's rule is the factoring out of the least power of a variable from successively lower-degree terms. For example,

$$-1234321\,x^7-1234321\,x^5+2468642\,x^4+7x^3+14x-21$$
$$\rightarrow\left(\left(\left(\left(-1234321\,x^2-1234321\right)x+2458642\right)x+7\right)x^2+14\right)x-21.$$

[16] With correct handling of infinities, a continued fraction can be defined at infinity where a reduced common denominator is *not*. For example, $1/(1+1/z)\rightarrow 1$ at $z=\infty$, where the corresponding reduced ratio $z/(z+1)\rightarrow\infty/\infty$ is indeterminate. Most people would prefer having a result defined at all *finite* values of variables to being defined at *infinite* values, but a mere division can turn 0 into an infinite value.

It is also worth partially factoring out units and numeric content to the extent that it reduces bulk or the number of operations. For example,

$$-1234321\,x^7 - 1234321\,x^5 + 2468642\,x^4 + 7x^3 + 14x - 21$$
$$\rightarrow ((-1234321((x^2+1)\,x-2)\,x+7)\,x^2+14)\,x-21.$$

Horner's rule often leads to faster evaluation when substituting numbers for variables, which is particularly important in situations such as plotting, where substitution is done many times for different values. Horner's rule also often improves accuracy for approximate arithmetic, because the operands of a catastrophic cancellation are closer to the input numbers, hence less contaminated with rounding error.

Horner's rule can be viewed as factoring out term content term by term, starting with the highest-degree terms at each level. With this viewpoint, we can apply it to multinomials throughout an expression in all of the above special forms. Ceberio and Kreinovich [8] discuss greedy algorithms for computing efficient multivariate Hornerized forms.

Another transformation that can enable faster evaluation when substituting numbers for variables is to factor out an integer common divisor of the exponents from a product of powers. For example, if the powers are done with the help of repeated squaring, then

$$(y+3)^6x^4 \rightarrow \left((y+3)^3\,x^2\right)^2$$

uses only five multiplications rather than six.[17]

References

[1] Acton, F.S., *Numerical Methods that Work*, Chapter 11, Harper and Row, 1970 or The Mathematical Association of America, 1990.

[2] Avitzur, R., Milo (a Macintosh computer program), Paracomp Inc., 1988.

[3] Avitzur, R., The Graphing Calculator Story, `http://www.pacifict.com/Story/`

[4] Ayad, M. and Fleischmann, P., On the decomposition of rational functions, *Journal of Symbolic Computation* 43 (4), pp. 259-274, 2008.

[5] Bailey, D. and Borwein, J., Inverse Symbolic Calculator, `http://isc.carma.newcastle.edu.au/advanced`

[6] Barton, D. and Fitch, J.P., A review of algebraic manipulative programs and their application, *The Computer Journal* 15(4), pp. 362-381, 1972.

[7] Bonadio, A., Theorist (a computer program), Prescience Corporation, 1989.

[8] Ceberio, M. and Kreinovich, V., Greedy algorithms for optimizing multivariate Horner schemes, *ACM SIGSAM Bulletin* 38(1), pp. 8-15, 2004.

[17] However, for most systems default simplification automatically distributes the outer exponent 2 over the two factors unless something special is done to prevent it.

[9] Cheze, C. and Galligo, A., Four lectures on polynomial absolute factorization, *Algorithms and Computation in Mathematics* 14, pp. 339-392, 2005.

[10] Cuyt, A.A.M. and Verdonk, B.M., A review of branched continued fraction theory for the construction of multivariate rational approximants, *Applied Numerical Mathematics* 4 (2-4), pp. 263-271, 2005.

[11] Fateman, R.J., Notes on Chebyshev series and computer algebra, 2011, `http://www.cs.berkeley.edu/~fateman/papers/cheby.pdf`

[12] Faugère, J.C. and Perret, L.. High order derivatives and decomposition of multivariate polynomials. *Proceedings of ISSAC 2009*, pp. 207-214.

[13] Giesbrecht, M., Kaltofen, E. and Lee, Wen-shin., Algorithms for computing the sparsest shifts of polynomials via the Berlekamp/Massey algorithm, *Proceedings of ISSAC 2002*, pp. 101-108.

[14] Grigoriev, D.Y., Lakshman, Y.N., Algorithms for computing sparse shifts for multivariate polynomials, *Proceedings of ISSAC 1995*, pp. 96-103.

[15] Gutierrez, J., Rubio, R. and Sevella, D., On multivariate rational function decomposition, *Journal of Symbolic Computation* 33 (5), pp. 545-562, 2002.

[16] Kajler, N. and Soiffer, N., A survey of user interfaces for computer algebra systems, *Journal of Symbolic Computation* 25 (2), pp. 127-159, 1998.

[17] Kozen, D. and Landau, S., Polynomial decomposition algorithms, *Journal of Symbolic Computation* 7 (5), pp. 445-456, 1989.

[18] Krausz, F., A better user interface for Symbolics Lisp Machine Macsyma, Macsyma Newsletter 5(3), pp. 3-5, 1988.

[19] MathMonkeys, LLC, LiveMath, formerly known as Theorist, MathView, MathPlus, and Live Math Maker., `http://www.livemath.com/`

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

[21] Powers, V. and Wörmann, T., An algorithm for sums of squares of real polynomials, `http://www.mathcs.emory.edu/~vicki/pub/sos.pdf`

[22] Reznick, B.E., *Sums of Even Powers of Real Linear Forms*, Memoirs of the American Mathematical Society, 1992, and `http://www.math.uiuc.edu/~reznick/memoir.html`

[23] Shneiderman, B. and Plaisant, C., *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, 4th edition, Pearson Addison Wesley, p. 367, 2004.

[24] Stoutemyer, D.R., Ten commandments for good default expression simplification, *Journal of Symbolic Computation*, 46(7), pp. 859-887, 2011.

[25] Stoutemyer, D.R., Unit normalization of multinomials over Gaussian integers, *ACM Communications in Computer Algebra* 43 (3/4), pp. 73-76, 2009.

[26] Stoutemyer, D.R, Multivariate partial fraction expansion. *ACM Communications in Computer Algebra* 42 (4), pp. 206-210, 2008.

[27] Stoutemyer, D.R., Useful computations need useful numbers, *ACM Communications in Computer Algebra* 41 (3), pp. 75-99, 2007.

[28] Sturmfels, B., *Algorithms in invariant theory*, 2nd edition, Springer 2008.

[29] Théry, L., Bertot, Y. and Kahn, G., Real theorem provers deserve real user-interfaces, *SDE 5 Proceedings of the fifth ACM SIGSOFT symposium on Software development environments* 17(5), pp. 120-129, 1992.
Preprint at `http://hal.inria.fr/docs/00/07/69/07/PDF/RR-1684.pdf`

[30] Trefethen, L.N. and others, Chebfun Version 4.2, The Chebfun Development Team, 2011, `http://www.maths.ox.ac.uk/chebfun/`

[31] Vandenberghe, L. and Boyd, S., "Semidefinite Programming", *SIAM Review* 38, pp. 49-95, March 1996.

[32] von zur Gathen, J., Functional Decomposition of Polynomials: The Wild Case., *Journal of Symbolic Computation* 10(5): pp. 437-452, 1990.

[33] von zur Gathen, J., Gutierrez, J., Rubio, R., Multivariate Polynomial Decomposition, *Applicable Algebra in Engineering, Communication and Computing* 14(1), pp. 11-31, 2003.

[34] von zer Gathen, J. and Weiss, J., Homogeneous bivariate decompositions, *Journal of Symbolic Computation* 19, pp. 409-434, 1992.

[35] Watt, S.M., Functional Decomposition of Symbolic Polynomials, *Proceedings of the International Conference on Computational Sciences and its Applications*, IEEE Computer Society, pp. 353-362, 2008.

[36] Wikipedia, Model-view-controller,
`http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller#References`

[37] UITP, User Interfaces for Theorem Provers, `http://www.informatik.uni-bremen.de/uitp/`

[38] Wolfram|alpha, `http://www.wolframalpha.com/`

[39] Zippel, R., Rational function decomposition, *Proceedings of ISSAC-91*, pp. 1-6, 1991.

ACM Communications in Computer Algebra, Vol. 47, No. 4, Issue 186, December 2013

ISSAC 2013 Software Presentations

Communicated by Joris van der Hoeven

Arb: a C library for ball arithmetic

Fredrik Johansson *
RISC
Johannes Kepler University
4040 Linz, Austria
fjohanss@risc.jku.at

1 Introduction

Arb [1] is a new open source C library for provably correct arbitrary-precision numerics, extending FLINT [3] (which provides fast arithmetic over various exact rings) to the real and complex numbers. Following the example of iRRAM [7] and Mathemagix [13], Arb performs automatic error propagation using ball arithmetic [12] (not to be confused with heuristic significance arithmetic as used e.g. in Mathematica [9]). This gives performance close to floating-point arithmetic such as provided by MPFR [2] while avoiding the cost at high precision of endpoint-based interval arithmetic as provided for instance by MPFI [8].

One of our motivations for developing a new library has been to provide a low-level, low-overhead interface, and our implementation differs from others in some technical aspects. Arb also provides fast polynomial arithmetic, to our knowledge only available in Mathemagix and without error control in MPFRCX [1], as well as matrix arithmetic. Finally, Arb implements some special functions that have been absent from arbitrary-precision interval software, with performance that compares favorably to available nonrigorous implementations. The presentation covers implementation details and shows some benchmarks.

2 Feature overview

Arb provides the following types:

- `fmpr_t`: floating-point real numbers $\mathbb{R}_D = \mathbb{Z} \times 2^{\mathbb{Z}} \cup \{-\infty, +\infty, \mathrm{NaN}\}$
- `fmprb_t`: real numbers implemented as balls $\mathbb{R}_B = \{[m-r, m+r] : m, r \in \mathbb{R}_D, r \geq 0\}$
- `fmpcb_t`: complex numbers in rectangular form $\mathbb{C}_B = \mathbb{R}_B[i]$
- `fmprb_poly_t`, `fmpcb_poly_t`: polynomials (and truncated power series) over $\mathbb{R}_B$, $\mathbb{C}_B$
- `fmprb_mat_t`, `fmpcb_mat_t`: matrices over $\mathbb{R}_B$, $\mathbb{C}_B$

Each type has a set of associated methods for memory management, conversions, arithmetic and special functions, with an interface resembling that of FLINT. For example, the power series multiplication $a \leftarrow b \times c \bmod x^n$ with rounding to *prec* bits, where a, b, c are of type `fmprb_poly_t`, is written as:

```
fmprb_poly_mullow(a, b, c, n, prec)
```

Polynomial methods have corresponding "underscore" versions that act directly on coefficient arrays, reducing overhead and giving more control over memory allocation and copying (like the `mpn` layer of GMP):

```
_fmprb_poly_mullow(a->coeffs, b->coeffs, b->length, c->coeffs, c->length, n, prec)
```

*Supported by the Austrian Science Fund (FWF) grant Y464-N18.

[1] `http://fredrikj.net/arb/` (Arb is licensed GNU GPL version 2 or later)

3 Representation of numbers

Arb does not directly base its arithmetic on MPFR (but does call MPFR for a few operations, and the test suite extensively verifies correctness against MPFR). MPFR attaches a precision to each variable, and allocates memory for a full-precision number even if only a few bits are used. In Arb, the precision is always passed as an argument to each function; the components of an `fmpr_t` are FLINT integers, and can grow dynamically. A mantissa or exponent with at most 62 bits (30 bits on a 32-bit system) is particularly efficient, as it takes up a single word in the `fmpr_t` struct without allocating memory on the heap.

We have found this approach convenient for mixed-precision algorithms and particularly valuable for computations involving integer coefficients of variable size (such as binary splitting and various polynomial operations). The same type also works well for low-precision arithmetic such as error bound calculations. The drawback is some overhead at precisions up to a few hundred digits, although experiments suggest that this overhead could be reduced with further implementation effort.

Bits	`mpfr_mul`	`fmpr_mul`	`fmprb_mul`
32	1.0	0.6	2.3
128	1.0	1.4	2.7
512	1.0	0.9	1.4
2048	1.0	1.1	1.2
8192	1.0	1.2	1.2
32768	1.0	1.2	1.2
131072	1.0	1.1	1.1
524288	1.0	1.0	1.0

Table 1: Time relative to MPFR of floating-point and ball multiplication. The difference below approximately 1000 bits results from implementation overhead, and the $10\% - 20\%$ difference around 10^3 – 10^5 bits is due to MPFR using the mulhigh algorithm.

An `fmprb_t` consists of a midpoint and a radius, both of type `fmpr_t`. Radius operations use a predefined precision (30 bits). Midpoint arithmetic is always carried out at the requested working precision. It would be more efficient to round midpoints to the accuracy indicated by the radius, though such a normalization naturally can be performed explicitly, and the present convention is sometimes useful for detecting when the computed error bound greatly overshoots the actual numerical error.

Complex numbers are represented as pairs of real balls. This seems preferable to a complex midpoint with a single radius, for reasons of convenience, and it is frequently useful to track whether either the real or imaginary part is exact. Similarly, polynomials and matrices are represented as arrays of coefficients to maximize flexibility. Where a different data order is required, temporary copies are relatively cheap since the base `fmpr_t` type takes up only two words and usually only needs to be copied shallowly.

4 Special functions

Except for some special cases, the elementary functions in Arb call the MPFR implementations of exp, log, sin, cos and atan (our future plan is to develop faster implementations for precisions up to a few thousand digits), using function derivatives to bound propagated errors. Care has been taken to ensure numerically satisfactory behavior on the whole complex plane, for example when evaluating $\tan(x+yi)$ for large $|y|$. Extremely large numbers are handled specially: we allow arbitrary-precision exponents, and we restrict the internal working precision allowed for argument reduction to a small multiple of the requested precision. For example, an attempt to evaluate $\cos(2^{10^{10}})$ quickly returns a crude bounding interval (e.g. $[-1, 1]$) unless the precision is set in the hundreds of millions of digits. This makes worst-case evaluation

time at a given precision predictable and avoids unnecessary stalls caused by tiny terms that might not even contribute to the final result, particularly aiding "black-box" use in computer algebra settings.

Interval software has historically been limited to the elementary functions and some special functions of a real variable, while software with good support for special functions (e.g. [10], [5]) has not guaranteed correctness. We wish to improve this situation. As of the current version, Arb provides Bernoulli numbers, the Hurwitz zeta function $\zeta(s,a)$ and its derivatives with respect to s for complex s and a, and the gamma and digamma functions for real and complex arguments. The implementations are tuned for different sizes and precisions, incorporating many optimizations. Arb also contains code for binary splitting evaluation of generic rational hypergeometric series with automatic error bounding, used for evaluation of mathematical constants, as well as code for rigorous polynomial root refinement, used for some algebraic numbers.

Evaluation	Digits	MPFR 3.1.1	Pari/GP 2.5.3	Mathematica 8.0	Arb
A: γ (Euler's constant)	10^6	93 s	$>$ 1 h	30 s	18 s
B: $\cos(\pi/31)$	10^5	6.1 s	42	12 s	0.48 s
C: ${}_3F_2(\frac{1}{2},\frac{1}{3};\frac{1}{4},\frac{1}{5},\frac{1}{6};\frac{1}{7})$	10^5	n/a	n/a	1396 s	0.45 s
D: $\Gamma(\sqrt{2})$	10^4	60 s	1.9 (233) s	13 s	0.21 (1.3) s
E: $\Gamma(\sqrt{2}+i\sqrt{3})$	10^4	n/a	2.9 (235) s	5.8 (44) s	0.67 (1.7) s
F: $\zeta(1/2+1000i)$	10^4	n/a	24 (1571) s	672 s	22 (25) s
G: $\zeta(1+2i,3+4i)$	10^3	n/a	n/a	2.4 s	0.38 s

Table 2: Special function timings, measuring repeated calls with the initial call inside parentheses. Algorithms in Arb: A) binary splitting B) minimal polynomial root refinement C) generic binary splitting D-E) Stirling's series F-G) Euler-Maclaurin summation.

5 Polynomials and power series

Polynomial operations are implemented in an asymptotically fast way by reducing to multiplication using standard techniques such as Newton iteration for division, series logarithm and series exponential, divide-and-conquer for composition [4], rectangular splitting for power series composition, and product trees for fast multipoint evaluation and interpolation. We have implemented three algorithms for multiplication in $\mathbb{R}[x]$: classical, sloppy, and blockwise. The latter two translate to $\mathbb{Z}[x]$ and call FLINT (which uses classical, Karatsuba, Kronecker substitution, and Schönhage-Strassen FFT multiplication).

The sloppy algorithm cuts off the coefficients of each input polynomial *prec* bits below the top bit of the polynomial as a whole, performs a single multiplication over $\mathbb{Z}[x]$, and bounds errors using max norms. This is fast, and numerically satisfactory if all coefficients have the same magnitude, but not used by default due to the poor numerical stability for polynomials with coefficients of varying magnitude.

The blockwise algorithm splits the input polynomials into blocks of similarly-sized coefficients and multiplies each pair of blocks exactly in $\mathbb{Z}[x]$. In the worst case, this degenerates to multiplication of 1×1 blocks equivalent to classical multiplication. In the typical case, it only performs slightly worse than the sloppy multiplication. Accurate per-coefficient error bounds are computed using an $O(n^2)$ loop running over exponents. The algorithm could be improved further using scaling and by discarding parts of the inputs that do not contribute to the result, as discussed in [11].

We illustrate the importance of polynomial arithmetic that is both fast and numerically stable. Letting $\xi(s) = (s-1)\pi^{-s/2}\Gamma\left(1+\frac{1}{2}s\right)\zeta(s)$, Li's criterion [6] states that the Riemann hypothesis is equivalent to the positivity for all $n > 0$ of the coefficients λ_n defined by $\log \xi\left(z/(z-1)\right) = \sum_{n=0}^{\infty} \lambda_n z^n$. We prove positivity of the first 10,000 coefficients by evaluation. This requires derivatives of $\zeta(s)$, a series logarithm, derivatives of $\log\Gamma(s)$, and a series composition with $z/(z-1)$. In this example, the final composition catastrophically

magnifies the error bounds if sloppy multiplication is used, making a precision of nearly $10n$ bits necessary. With classical multiplication, about $1.3n$ bits suffice, speeding up the the zeta function evaluation, but the subsequent power series operations now dominate. Blockwise multiplication allows using the same precision as with classical multiplication, and the power series operations only take a fraction of the time.

	Sloppy	Classical	Blockwise
Working precision	100000 bits	13000 bits	13000 bits
Zeta	147180 s	1242 s	1272 s
Logarithm	56 s	2760 s	8.3 s
Gamma	781 s	3.4 s	3.4 s
Composition	1994 s	7971 s	185 s
Total	150011 s	11976 s	1469 s

Table 3: Precisions and timings for computing the Li coefficients λ_n up to $n = 10000$ with correctly determined signs, using three different multiplication algorithms.

References

[1] A. Enge. MPFRCX: a library for univariate polynomials over arbitrary precision real or complex numbers, 2012. `http://www.multiprecision.org/index.php?prog=mpfrcx`.

[2] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software*, 33(2):13:1–13:15, June 2007. `http://mpfr.org`.

[3] W. B. Hart. Fast Library for Number Theory: An Introduction. In *Proceedings of the Third international congress conference on Mathematical software*, ICMS'10, pages 88–91, Berlin, Heidelberg, 2010. Springer-Verlag. `http://flintlib.org`.

[4] W. B. Hart and A. Novocin. Practical divide-and-conquer algorithms for polynomial arithmetic. In *Computer Algebra in Scientific Computing*, pages 200–214. Springer, 2011.

[5] F. Johansson et al. *mpmath: a Python library for arbitrary-precision floating-point arithmetic, version 0.17*, 2011. `http://mpmath.org`.

[6] Xian-Jin Li. The positivity of a sequence of numbers and the Riemann Hypothesis. *Journal of Number Theory*, 65(2):325–333, 1997.

[7] N. Müller. The iRRAM: Exact arithmetic in C++. In *Computability and Complexity in Analysis*, pages 222–252. Springer, 2001. `http://irram.uni-trier.de`.

[8] N. Revol and F. Rouillier. Motivations for an arbitrary precision interval arithmetic library and the MPFI library. *Reliable Computing*, 11(4):275–290, 2005. `http://perso.ens-lyon.fr/nathalie.revol/software.html`.

[9] M. Sofroniou and G. Spaletta. Precise numerical computation. *Journal of Logic and Algebraic Programming*, 64(1):113–134, 2005.

[10] The PARI Group, Bordeaux. *PARI/GP, version 2.5.3*, 2012. `http://pari.math.u-bordeaux.fr`.

[11] J. van der Hoeven. Making fast multiplication of polynomials numerically stable. Technical Report 2008-02, Université Paris-Sud, Orsay, France, 2008.

[12] J. van der Hoeven. Ball arithmetic. Technical report, HAL, 2009. `http://hal.archives-ouvertes.fr/hal-00432152/fr/`.

[13] J. van der Hoeven, G. Lecerf, B. Mourrain, P. Trébuchet, J. Berthomieu, D. N. Diatta, and A. Mantzaflaris. Mathemagix: the quest of modularity and efficiency for symbolic and certified numeric computation? *ACM Communications in Computer Algebra*, 45(3/4):186–188, January 2012. `http://mathemagix.org`.

ACM Communications in Computer Algebra, Vol. 47, No. 4, Issue 186, December 2013

NAClab: A Matlab Toolbox for Numerical Algebraic Computation

(extended abstract)

Zhonggang Zeng* Tien-Yien Li†

1 Introduction

We present a Matlab toolbox NAClab for numerical algebraic computation. This toolbox includes Matlab implementations of the basic numerical algorithms in algebraic computations and utility functions. Those functions can be used either directly in applications or as building blocks for implementing advanced computing methods. NAClab is a result of collective effort and its contributors include Liping Chen, Tianran Chen, Wenrui Hao, Tsung-Lin Lee, Andrew Sommese and Wenyuan Wu.

Numerical algebraic computation, particularly numerical polynomial algebra, emerges as a growing area of study in recent years with a solid foundation in place for building numerical and hybrid computing methods [8, 9]. Many robust numerical and numeric-symbolic algorithms have been developed for solving polynomial systems, polynomial factorizations, polynomial GCD, computing dual bases and multiplicity structures of polynomial ideals, etc, with implementations such as in [1, 2, 3, 5, 6, 10] Those algorithms have a broad spectrum of applications in scientific computing such as robotics, control, image processing, computational biology and chemistry, and so on.

One of the main difficulties for numerical algebraic computation is the ill-posedness that frequently occurs when the solution of a problem does not possess Lipschitz continuity with respect to data. Those ill-posed problems are not directly suitable for floating point arithmetic since the solutions are infinitely sensitive to rounding errors. However, it has been shown in recent studies that such a difficulty can be overcome by seeking a regularized numerical solution with a proper formulation. Algorithms implemented in NAClab are designed to regularize the problem for removing ill-posedness rather than extending machine precision for accurate computation. The package also includes generic routines for matrix building and Gauss-Newton iteration that are the main engines for handling ill-posed algebraic computaiton.

NAClab is an on-going project as a major upgrade and expansion from its predecessor ApaTools [13]. At this stage, the main objective is to provide researchers in numerical algebra with generic and versatile tools that simplify and accelerate algorithm development, experimentation, and implementation. We emphasize on achieving the highest possible accuracy and robustness in algorithm design and implementation.

NAClab is maintained at its permanent website[1] and freely accessible to academic researchers and educators for the foreseeable future.

2 Differences between symbolic and numerical algebra

Conventional symbolic computation assumes both data and arithmetic to be exact. In practical applications, however, problem data are likely to be empirical. As a result, exact solutions of those inexact problems may not serve the practical purposes. Using the polynomial

$$p = 81\,x^4 + 16\,y^4 - 648.001\,z^4 + 72x^2y^2 + .002x^2z^2 + .001y^2z^2 - 648x^2 - 288y^2 - .007z^2 + 1296 \qquad (1)$$

*Department of Mathematics, Northeastern Illinois University, Chicago, IL 60625, USA, email: zzeng@neiu.edu.
†Department of Mathematics, Michigan State University, E. Lansing, MI 48824, USA, email: li@math.msu.edu
[1] http://www.neiu.edu/∼naclab

in [7] as an example, Kaltofen proposed the Open Problem 1 in challenges of symbolic computation: Is there factorable polynomial nearby? While polynomial p in (1) is not factorable in conventional sense, it is a perturbed data representation of a factorable polynomial $\tilde{p}$. The objective of numerical factorization is to calculate the factorization of the hidden underlying polynomial $\tilde{p}$ using the imperfect data p. The function `PolynomialFactor` in `NAClab` is built to carry out this computation as follows. First of all, `NAClab` allows polynomials to be entered intuitively into Matlab as strings:

```
>> p = '81*x^4+16*y^4-648.001*z^4+72*x^2*y^2+.002*x^2*z^2+.001*y^2*z^2-648*x^2-288*y^2-.007*z^2+1296';
```

Then the numerical factorization can be obtained using the known relateve data error bound 10^{-5} as the error tolerance:

```
>> F = PolynomialFactor(p,1e-5,'row')     % factor p within error tolerance 1e-5, showing result in a row
F =
1296 * (1-0.25000013640167*x^2-0.111111232357041*y^2+0.707104626181296*z^2) * (1-0.249999863598339*x^2-0.111110989865191*y^2-0.707110027415863*z^2)
```

This result is an accurate approximation to the factorization of the hidden polynomial $\tilde{p}$ and reveals the graph of $p = 0$ is the union of an ellipsoid and a hyperboloid of one sheet.

In contrast to symbolic computation, another subtle issue in numerical algebraic computation is that the given empirical data may have different underlying solutions depending on the error tolerance. This phenominon can be further illustrated in polynomial factorization on

$$f \quad = \quad 0.47619031y + 0.5714288y^2 + 0.55555493x + 1.3809278yx + 0.8571143y^2x + 0.8333328x^2 + yx^2$$

Two numerical factorizations within different error tolerances can be computed by `NAClab` function `PolynomialFactor`:

```
>> PolynomialFactor(f,1e-4,'row')  % within tolerance 1e-4
ans =
0.999996404282881 * (0.833329772039985 + y)  * (0.666677452701414 + x)  * (0.85712030479082*y + x)

>> PolynomialFactor(f,1e-6,'row')  % within tolerance 1e-6
ans =
1.00000001100681 * (0.833332777619492 + y)  * (0.571428774473131*y + 0.666666352807009*x + 0.857114290319387*y*x + x^2)
```

They are accurate approximations to the factorizations of two nearby polynomials from the same data.

Many other algebraic computations follow a similar pattern: An accurate solution of an algebraic problem is to be computed but the problem data are imperfect so that the exact solution is meaningless since the solution is infinitely sensitive to data perturbations. The objective of the numerical algebraic computation is to solve the problem using the slightly perturbed data within an error tolerance similar to the factorization example above. `NAClab` is built for this purpose.

Algebraic problems are often ill-posed because the set of problems whose solutions possessing a distinct structure form a manifold of positive codimension, and perturbations generically pushes a given problem away from the manifold. Our strategy starts with formulating the *numerical solution* of an ill-posed algebraic problem following a "three strikes" principle consisting of *backward nearness, maximum codimension* and *minimum distance* [13] for removing the ill-posedness. Based on those formulation principles, computing the numerical solution can be carried out in two optimization processes: maximizing the codimension of manifolds followed by minimizing the distance to the manifold, leading to a two-staged strategy for designing robust algorithms.

The main mechanism at Stage I is matrix rank-revealing, while Stage II relies on solving nonlinear least squares problems. In `NAClab`, we provide matrix building/computation tools for Stage I and nonlinear least squares tools for Stage II.

3 NAClab overview

NAClab originated from its predecessor ApaTools [13]. Among many improvement areas, we implemented a user-friendly mechanism for direct polynomial manipulations and included a major package in numerical solution of polynomial systems by homotopy continuation method. For example, solving the polynomial system

$$x^5 - y^5 + 3y + 1 \quad = \quad 5y^4 - 3 \quad = \quad 20x - y + z \quad = \quad 0$$

can now be carried out in a simple call:

```
>> P = {'x^5-y^5+3*y+1','5*y^4-3','20*x-y+z'};    % enter the polynomial system directly and intuitively
>> [Solutions, variables] = psolve(P)             % call the polynomial system solver and obtain all the isolated solutions
   Solutions =
     Column 1
     0.778497746685646 + 0.893453081179308i
    -0.000000000000000 + 0.880111736793394i
   -15.569954933712914 -16.988949886792764i
     ... ...
     Column 20
     0.778497746685646 - 0.893453081179308i
     0.000000000000000 - 0.880111736793393i
   -15.569954933712925 +16.988949886792767i

   variables =
    'x'    'y'    'z'
```

Compared to Maple and Mathematica, Matlab has an advantage in efficient numerical matrix computations with an disadvantage in user interface. Cumbersome representations are required for carrying out polynomial and other algebraic computations. Using NAClab, polynomials can be entered and displayed as character strings in Matlab in a way similar to Maple.

```
>> f = '3*x^2 - (2-5i)*x^3*y^4 - 1e-3*z^5-6.8'
>> g = '-2*y^3 - 5*x^2*z + 8.2'
```

Using such an intuitive polynomial representation, users can now perform common polynomial operations such as addition, multiplication, power, evaluation, differentiation, factorization, extracting coefficients, finding greatest common divison (GCD), etc, by calling NAClab functions, such as

```
>> p = pplus('2*x^5-3*y','4+x*y')           % add any number of polynomials
>> q = ptimes(f,g,h)                        % multiply any number of polynomials
>> v = PolynomialEvaluate(f,{'x','z'},[3,4]) % evaluate f(x,y,z) for x=3, z=4

>> % and a lot more.  For example, to compute a greatest common divisor:
>> f = '10 - 5*x^2*y + 6*x*y^2 - 3*x^3*y^3';
>> g = '30 + 10*y + 18*x*y^2 + 6*x*y^3'
>> u = PolynomialGCD(f,g)
u =
33.5410196624968 + 20.1246117974981*x*y^2
```

In summary, NAClab is developed for numerical algebraic computations in Matlab including solving polynomial systems, polynomial factorizations, polynomial greatest common divisors, multiplicity and dual spaces of nonlinear systems at isolated zeros, numerical Jordan Canonical Forms, and numerical rank revealing. The package also contains a comprehensive library of programming utilities for building further algorithms for numerical algebraic computations.

NAClab is an on-going project. While continuing to refine the existing functions, we shall expand the package by developing more algorithms and their implementations for numerical algebraic computations.

References

[1] D.J. Bates, C. Peterson and A.J. Sommese, *A numerical-symbolic algorithm for computing the multiplicity of a component of an algebraic set*, IEEE Trans. Signal Processing, 52 (2003), pp. 3394–3402.

[2] D.J. Bates, J.D. Hauenstein, A.J. Sommese and C.W. Wampler II, *Software for numerical algebraic geometry: A paradigm and progress towards its implementation*, in Software for Algebraic Geometry, IMA Volume 148, M. Stillman, N. Takayama, and J. Verschelde, eds., Springer, 2008, pp. 1–14.

[3] R. M. Corless, S. M. Watt, and L. Zhi, *QR factoring to compute the GCD of univariate approximate polynomials*, IEEE Trans. Signal Processing, 52 (2003), pp. 3394–3402.

[4] B. Dayton, T.Y. Li and Z. Zeng, *Multiple zeros of nonlinear systems* Mathematics of Computation, Vol. 80, pp. 2143-2168, 2011

[5] S. Gao, E. Kaltofen, J. May, Z. Yang, and L. Zhi, *Approximate factorization of multivariate polynomials via differential equations.* Proc. ISSAC '04, ACM Press, pp 167-174, 2004.

[6] C.-P. Jeannerod and G. Labahn, *The SNAP package for arithemetic with numeric polynomials.* In International Congress of Mathematical Software, World Scientific, pages 61-71, 2002.

[7] E. Kaltofen, *Challenges of symbolic computation: My favorite open problems*, J. Symb. Comput., 29, pp.161-168, 2000.

[8] A.J. Sommese and C.W. Wampler II, *The Numerical Solution of Systems of Polynomials*, World Scientific Pub., Hackensack, NJ. 2005

[9] H. J. Stetter, *Numerical Polynomial Algebra*, SIAM, 2004.

[10] J. Verschelde, *Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation*, ACM Trans. on Math. Software, 25(1999), pp. 251–276.

[11] Z. Zeng, *A polynomial elimination method for symbolic and numerical computation.* 409(2008) pp. 318-331.

[12] Z. Zeng, *Computing multiple roots of inexact polynomials*, Math. Comp., 74 (2005), pp. 869–903.

[13] Z. Zeng, *ApaTools: A Maple and Matlab toolbox for approximate polynomial algebra*, in Software for Algebraic Geometry, IMA Volume 148, M. Stillman, N. Takayama, and J. Verschelde, eds., Springer, 2008, pp. 149–167.

[14] Z. Zeng and B. Dayton, *The approximate GCD of inexact polynomials. II: A multivariate algorithm.* Proceedings of ISSAC'04, ACM Press, pp 320-327. (2006).

ACM Communications in Computer Algebra, Vol. 47, No. 4, Issue 186, December 2013

The new `GroupTheory` package in Maple 17

E.J. Postma

Abstract

The `GroupTheory` package was added to Maple 17 [1]. It deals with discrete groups. The package offers visualizations of group theoretic concepts; accepts input and generates output that is relatively close to the classical textbook notation; and can deal with parameterized groups where the parameters are given only symbolically.

Maple [1] is a well-known computer algebra package, initially developed in the early 1980s by the Symbolic Computation Group at the University of Waterloo. Maple has had a package dealing with group theory dating back to the first decade of its existence. It is called `group` and has been showing its limitations for a long time. For Maple 17, a completely new package called `GroupTheory` was added that replaces the older package. It was written at Maplesoft, incorporating many ideas and a substantial amount of code and data from two earlier packages written by the Computer Algebra Group at Simon Fraser University. One is a library of group presentations and permutation representations, written by Vahid Dabbaghian. The other package [2] focuses on visualization (it contained most of the code described in Section 4) and also performs substantial computation; it was written by Asif Zaman and Michael Monagan.

Other notable software packages dealing with discrete groups are GAP [3], Magma [4], and Mathematica [5]'s group theory features. We believe that, while some features offered by these packages are similar, each of the Sections 2, 3, and 4 describes some functionality that is unique to Maple.

1 Basic functionality

Groups can be represented in one of five ways: by permutations, by generators and relations, by an explicit multiplication table, as a set with manually defined group operations, and as an abstract group where elements cannot be listed but only properties computed. (This last option is explored in Section 2.) There is a substantial library of groups, containing all small groups of up to 200 elements (numbered consistently with the small group databases in GAP and Magma), all simple groups, many linear groups, and many individually named groups. These groups can typically be constructed in multiple representations: for example, using the `DihedralGroup` command, one can construct either the permutation or the finitely presented representation.

Such a group can then be interrogated about its properties, such as its order, whether it is simple, or its Fitting subgroup. Isomorphism testing between arbitrary groups is also supported. In total, the package contains about 120 commands for constructing and interrogating groups.

2 Symbolic groups

The `GroupTheory` package can deal with groups where it cannot list any elements explicitly. There are two such classes of groups: those where the group is defined only up to a symbolic parameter (such as `DihedralGroup(n)` or $\mathbf{D}_n$, the dihedral group with $2n$ elements), and those where listing elements is merely highly impractical, such as the Monster group $\mathbb{M}$.

For groups of the latter kind, Maple simply has many properties stored explicitly. Here are a few examples concerning the Monster group:

```
> GroupOrder(Monster());
```

$$808017424794512875886459904961710757005754368000000000$$

```
> IsSimple(DirectProduct(Monster(), TrivialGroup()));
```

true

```
> IsSimple(DirectProduct(Monster(), CyclicGroup(2)));
```

false

For groups defined with a symbolic parameter, Maple has some predefined properties as for the purely symbolic groups, but it can also deduce some properties from assumptions made on the parameters:

```
> IsNilpotent(DihedralGroup(6*n)) assuming n :: posint;
```

true

3 Interface

The `GroupTheory` package attempts to accept input and generate output in typography that is as close as possible to what one would traditionally find it in a textbook.

The requirements for input are clearly much stricter than for output, in that the input has to be processed by the parser for the general Maple language. Nonetheless, the package understands as input finitely presented groups in the format

$$\langle g_1, \ldots, g_n | r_1, \ldots, r_k \rangle,$$

where $g_1, \ldots, g_n$ are the generators and $r_1, \ldots, r_k$ are the relations.

For output, there is much more functionality. A few examples follow.

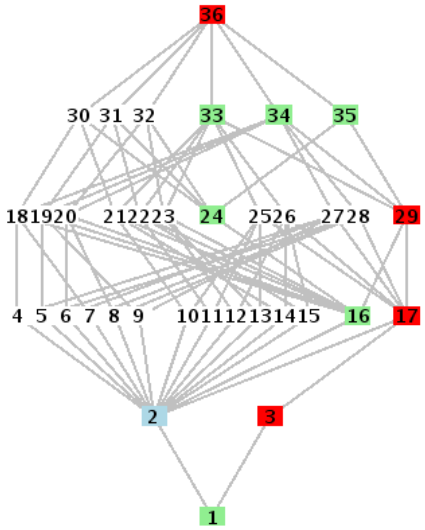


Figure 1: The subgroup lattice of the small group (48, 8). Highlighted in light blue is the centre, in light green are the normal subgroups, and in red is the lower central series.

Input and output	Discussion
`> DirectProduct(Monster(), Symm(n));` $\mathbb{M} \times \mathbf{S}_n$	The direct product of the monster group and the symmetric group of order n.
`> DerivedSeries(Symm(4));` $\mathbf{S}_4 \triangleright \langle (1,3,2),(2,4,3) \rangle \triangleright \langle (1,3)(2,4),(1,4)(2,3) \rangle \triangleright \langle \rangle$	The derived series of the symmetric group of order 4.

`> g := PSL(2, 3);`
$g := \mathbf{PSL}(2,3)$
`> h := SylowSubgroup(3, g);`
$h := \langle$a permutation group on 4 letters$\rangle$
`> k := Normaliser(h, g);`
$k := N_{\mathbf{PSL}(2,3)}(\langle$a permutation group on 4 letters$\rangle)$
`> GroupOrder(h);`
3
`> k;`
$N_{\mathbf{PSL}(2,3)}(\langle (2,3,4) \rangle)$
	The normalizer in $\mathbf{PSL}(2,3)$ of a Sylow-3 subgroup, h. Note that the generators of h are computed lazily, but once they are computed (in order to compute the order of h), they are remembered. This is discussed in Section 5.

4 Visualization

The `GroupTheory` package supports two visualizations: a subgroup lattice and a Cayley table.

An example of the subgroup lattice visualization can be found in Figure 1. This figure was obtained with the commands

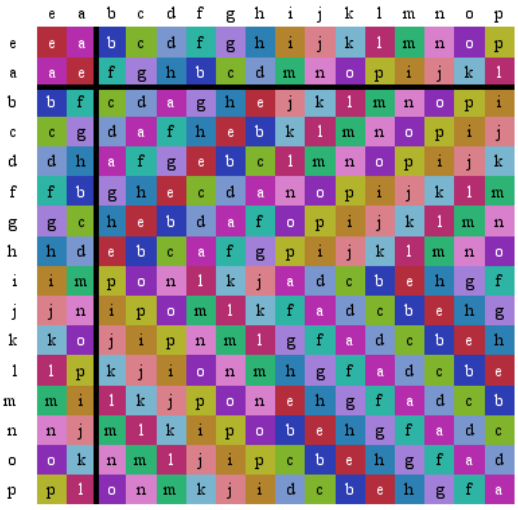


Figure 2: The Cayley table of the dicyclic group of order 16.

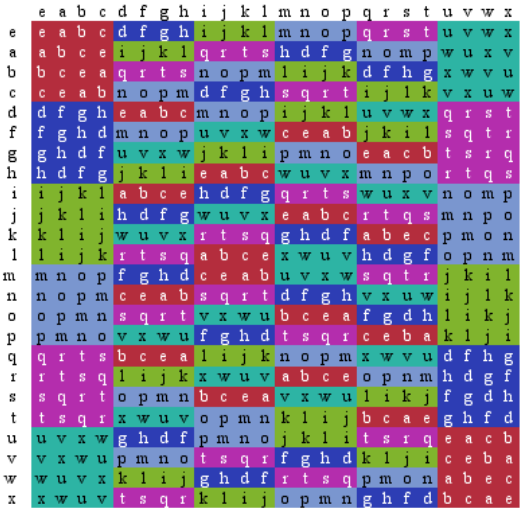


Figure 3: The Cayley table of a group of order 24.

```
> g := SmallGroup(48, 8):
> DrawSubgroupLattice(g, highlight=LowerCentralSeries(g));
```

Conjugate subgroups are placed next to each other, a little closer together than non-conjugate subgroups. This already shows, for example, what the normal subgroups are: the groups of size 1. For extra emphasis, any subset of subgroups can be highlighted in any combination of colours; the default is the centre and all normal subgroups. In the example, the lower central series is additionally highlighted (overriding the colours for the normal subgroups).

Two examples of the Cayley table visualization are shown in Figures 2 and 3. In Figure 2, the centre of the group is indicated by a thick black line. It consists of two elements. In Figure 3, the elements are highlighted depending on the coset of a particular subgroup they occur in.

5 Performance: memoization and autocompiled code

Groups are represented by objects (a type of modules). This provides part of the dispatching logic that selects the appropriate piece of code when a property needs to be computed for a given tuple of arguments. Another part is a memoization feature: most properties that are defined on a single argument, such as group order or simplicity, are computed only when necessary, but are remembered. That is, once such a property is computed, it is stored in a hash table in the object, with the property name as the key, and subsequent computations take advantage of this by looking it up rather than recomputing. A demonstration of this functionality is contained in Section 3.

For many of the low level algorithms, it was decided that the best tradeoff between performance and ease of programming was obtained by writing them in the subset of the Maple language that can be directly compiled into C code, and using the `autocompile` option. This means that the code is compiled on the fly.

As an example of these techniques, let us examine the `IdentifySmallGroup` command. It is used for identifying the isomorphism type of small groups (presently of size 200 or less) and uses the same numbering as its GAP and Magma equivalents. There are many shortcuts, for example if the group order n is prime. If

these do not apply, the algorithm determines, for each possible order $d|n$, the numbers of group elements of order d, and additionally the number of elements of the derived subgroup. These numbers are determined from the Cayley table of the group by autocompilable code. It then evaluates a perfect hash function on this sequence of numbers to find all groups that share these characterstics. (The hash function value is one of the values remembered as described in Section 5.) Subsequently, as long as there is more than one candidate left, the code uses its general isomorphism testing command `AreIsomorphic`.

References

[1] *Maple 17*. Maplesoft, a division of Waterloo Maple Inc., Waterloo, Ontario.

[2] Asif Zaman and Michael Monagan, *Visualizing Groups in Maple* (poster), `http://www.cecm.sfu.ca/research/posters/zaman09.pdf`.

[3] The GAP Group, *GAP – Groups, Algorithms, and Programming, Version 4.6.3*; 2013, (`http://www.gap-system.org`).

[4] Wieb Bosma, John Cannon, and Catherine Playoust, *The Magma algebra system. I. The user language*, J. Symbolic Comput., 24 (1997), 235265.

[5] Wolfram Research, Inc., *Mathematica, Version 9.0*, Champaign, IL (2012).

ACM Communications in Computer Algebra, Vol. 47, No. 4, Issue 186, December 2013

Holonomic Functions in Mathematica

Christoph Koutschan
Johann Radon Institute for Computational and Applied Mathematics
Austrian Academy of Sciences
Altenberger Straße 69, A-4040 Linz, Austria
christoph.koutschan@ricam.oeaw.ac.at

Abstract

We present the Mathematica package HolonomicFunctions which provides a powerful framework for the automatic manipulation of multivariate holonomic functions, in the spirit of Zeilberger's holonomic systems approach. Its top-level functionalities are: converting a mathematical expression into a holonomic description, executing holonomic closure properties, and creative telescoping for general holonomic functions. To achieve these goals, many other, lower-level, functionalities had to be implemented which were not available in the Mathematica system: finding rational solutions of linear systems of (q-) difference / differential equations, noncommutative arithmetic in Ore algebras and computing Gröbner bases in such domains.

In his seminal paper *A holonomic systems approach to special functions identities* [11], Zeilberger writes: "I hope that more professional programmers and algorithmic designers will soon expand the rudimentary ideas in this paper and develop a symbolic software package to prove general special function identities." Meanwhile, his dream has certainly become true: numerous papers refining and extending his "rudimentary ideas" have appeared, new and faster algorithms have been developed (some of them by Zeilberger himself), and there are several software packages available which implement (at least parts of) his holonomic systems approach.

Our package HolonomicFunctions is definitely one of the most general ones, since it allows to "prove general special function identities", as anticipated by Zeilberger. As special cases, our package contains his fast algorithm for proving hypergeometric identities [10] and its q-analogue, as well as the Almkvist-Zeilberger algorithm [1] for evaluating integrals of hyperexponential functions. Moreover, it can likewise deal with sums (resp. integrals) of functions which don't satisfy first-order recurrences (resp. differential equations), but higher-order ones. The only other software package we know of, providing this degree of generality, is Chyzak's Mgfun package for Maple. Our main motivations to reimplement the algorithms already contained in Mgfun were 1) to make them available to Mathematica users, 2) to provide an independent implementation in a different computer algebra system, 3) and to create an easily accessible, user-friendly interface to methods that are relevant to many disciplines outside of computer algebra. Additionally, we implemented some very recent algorithms for creative telescoping [7, 2] and closure properties [5].

Due to space restrictions, we will limit this short exposition to the top-level functionalities of HolonomicFunctions, which are: 1) converting a mathematical expression into a holonomic description, 2) executing holonomic closure properties, and 3) creative telescoping for general holonomic functions. More details, also about lower-level functionalities, can be found in [6], and detailed descriptions of all available commands are listed in the user's guide [8]. These documents, the package itself, and a collection of examples can be downloaded from the website

http://www.risc.jku.at/research/combinat/software/HolonomicFunctions/

(the required password is given for free to researchers and non-commercial users). HolonomicFunctions has first been released in 2009 and since then it has been extended constantly.

In order to apply the holonomic systems approach to some special function identity, the first step consists in converting all input expressions to holonomic descriptions (strictly speaking, our package works with descriptions of *∂-finite functions* [3], but for sake of simplicity, we don't want to insist on this subtle difference for now). This means, for a given mathematical function f, determine all linear partial (q-) difference / differential equations with polynomial coefficients that f satisfies ("holonomic system"). This usually infinite set of equations has the structure of a left ideal in the corresponding operator algebra; we use *Ore algebras* for representing such equations. The command to obtain an *annihilating ideal* (ideal of annihilating operators) for f is **Annihilator**; it takes as input a mathematical expression f, together with a list of operator symbols like D_x, S_n, etc. to specify the type of equations, and outputs the reduced left Gröbner basis of an annihilating ideal for f (note, however, that it is not guaranteed to be the maximal one so that it may not contain all annihilating operators for f). Following [4], we have extended our package such that it can also deal with certain non-holonomic functions. The **Annihilator** command recognizes more than 100 elementary and special functions (holonomic and non-holonomic) that are available in the Mathematica system.

The class of holonomic functions exhibits some nice and useful closure properties, among them addition, multiplication, certain substitutions and application of operators: given annihilating ideals of functions f and g, respectively, there are algorithms for computing an annihilating ideal of $f+g$, $f \cdot g$, etc. Since these operations are implemented on the level of ∂-finite functions, the corresponding commands are **DFinitePlus**, **DFiniteTimes**, **DFiniteSubstitute**, and **DFiniteOreAction**. We refer to [6] where these closure properties are explained in detail. We want to emphasize that the **Annihilator** command analyzes the syntactical structure of an input expression and then makes use of the above closure properties, e.g., multiplication in the input line In[2], see below.

It is a classic result that holonomic functions are also closed under taking integrals and sums; for these operations, the method of *creative telescoping* is employed. The problem of computing creative telescoping relations (consisting of the *telescoper* and the *certificate*) has attracted a great deal of attention during the last years. Zeilberger's solution in [11] (later coined "the slow algorithm") is based on elimination; in HolonomicFunctions it can be achieved via the **OreGroebnerBasis** command or the **FindRelation** command. The latter finds an element in a given left ideal that satisfies certain properties, to be specified by the user. Takayama's algorithm [9] is also based on elimination, but is more efficient than the slow algorithm; the command **Takayama** exports it to the user. The command **CreativeTelescoping** computes telescopers and certificates using Chyzak's algorithm [3], whereas the command **FindCreativeTelescoping** executes the heuristic fast approach proposed in [7]. Very recently, a creative telescoping algorithm for bivariate hyperexponential functions based on Hermite reduction [2] has been implemented by the author; it will be available in the next release of HolonomicFunctions.

At the end, we come back to Zeilberger's seminal paper [11] and demonstrate the usage of our package on the toy example that he discusses in the introduction, namely the generating function of the venerable Legendre polynomials:

$$\sum_{n=0}^{\infty} P_n(x) t^n = (1 - 2xt + t^2)^{-1/2}.$$

We start by computing an annihilating ideal for the summand on the left-hand side, and perform creative telescoping on it; this gives a list of telescopers and corresponding certificates:

```
In[1]:= << HolonomicFunctions.m
        HolonomicFunctions package by Christoph Koutschan, RISC-Linz, Version 1.6 (12.04.2012)

In[2]:= ann = Annihilator[LegendreP[n, x]*t^n, {S[n], Der[t], Der[x]}]

Out[2]= {t Dt - n, (n + 1) Sn + (t - t x^2) Dx + (-n t x - t x), (x^2 - 1) Dx^2 + 2 x Dx + (-n^2 - n)}
```

In[3]:= **{ts, cs} = CreativeTelescoping[ann, S[n] − 1, {Der[t], Der[x]}]**

Out[3]= $\left\{\left\{(-t^2+2tx-1)D_x+t,\ (t^2-2tx+1)D_t+(t-x)\right\},\left\{(tx-1)D_x-nt,\ (x-1)(x+1)D_x+\frac{n-ntx}{t}\right\}\right\}$

The correctness of this result can be verified by showing that the creative telescoping operators are indeed in the annihilating ideal (done here only for the first one, note the usage of noncommutative arithmetic):

In[4]:= **ct1 = ts[[1]] + (S[n] − 1) ** cs[[1]]**

Out[4]= $(tx-1)S_nD_x-(n+1)tS_n+(tx-t^2)D_x+(nt+t)$

In[5]:= **OreReduce[ct1, ann]**

Out[5]= 0

Finally, one computes an annihilating ideal for the right-hand side: it agrees with the left ideal generated by the two telescopers above.

In[6]:= **Annihilator[(1 − 2*x*t + t^2)^(−1/2), {Der[t], Der[x]}]**

Out[6]= $\left\{(t^2-2tx+1)D_x-t,\ (t^2-2tx+1)D_t+(t-x)\right\}$

Comparing initial values (not done here) completes the proof.

References

[1] Gert Almkvist and Doron Zeilberger. The method of differentiating under the integral sign. *Journal of Symbolic Computation*, 10(6):571–591, 1990.

[2] Alin Bostan, Shaoshi Chen, Frédéric Chyzak, Ziming Li, and Guoce Xin. Hermite reduction and creative telescoping for hyperexponential functions. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC)*, New York, NY, USA, 2013. ACM. To appear (preprint on arXiv:1301.5038).

[3] Frédéric Chyzak. An extension of Zeilberger's fast algorithm to general holonomic functions. *Discrete Mathematics*, 217(1-3):115–134, 2000.

[4] Frédéric Chyzak, Manuel Kauers, and Bruno Salvy. A non-holonomic systems approach to special function identities. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 111–118, New York, NY, USA, 2009. ACM.

[5] Stavros Garoufalidis and Christoph Koutschan. Twisting q-holonomic sequences by complex roots of unity. In Joris van der Hoeven and Mark van Hoeij, editors, *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 179–186. ACM, 2012.

[6] Christoph Koutschan. *Advanced applications of the holonomic systems approach*. PhD thesis, Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, Austria, 2009.

[7] Christoph Koutschan. A fast approach to creative telescoping. *Mathematics in Computer Science*, 4(2-3):259–266, 2010.

[8] Christoph Koutschan. HolonomicFunctions (user's guide). Technical Report 10-01, RISC Report Series, Johannes Kepler University, Linz, Austria, 2010. http://www.risc.jku.at/research/combinat/software/HolonomicFunctions/.

[9] Nobuki Takayama. An algorithm of constructing the integral of a module—an infinite dimensional analog of Gröbner basis. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 206–211, New York, NY, USA, 1990. ACM.

[10] Doron Zeiberger. A fast algorithm for proving terminating hypergeometric identities. *Discrete Mathematics*, 80(2):207–211, 1990.

[11] Doron Zeilberger. A holonomic systems approach to special functions identities. *Journal of Computational and Applied Mathematics*, 32(3):321–368, 1990.

ACM Communications in Computer Algebra, Vol. 47, No. 4, Issue 186, December 2013

Classifying Discrete Objects with *Orbiter*

Anton Betten

Abstract

Orbiter is a software package to classify discrete objects such as designs, graphs, codes, and objects from finite geometry. It employs the method of *breaking the symmetry* to attack difficult problem instances by means of subobjects that serve as a stepping stone. The algorithms combine techniques from Group Theory and from Combinatorics. *Orbiter* is a library of C++ functions that provide functionality for Discrete Mathematics. In order to be applied to a specific problem, code has to be written taylored to the specific application.

The Experimental Approach

Research in Mathematics often benefits from examples. There are many areas where the existing theory does not suffice to explain all the examples that are known. Therefore, studying examples is often the first step in finding new constructions or new theory. This new theory helps to explain where the known examples come from, and often predicts more examples for larger instances of the parameter set. In some cases, infinite families of objects can be constructed. In order to help with this process, knowing examples and their automorphism groups is crucial. In Discrete Mathematics and Combinatorics, we are able to examine (at least for small cases) complete lists of objects up to isomorphism. This kind of data is very valuable to researchers. *Orbiter* is intended to assist with these classification problems. One could call this the *experimental approach* to mathematics. Designs, graphs and codes are related topics, and all are well-suited to computer investigation. The various links between these areas are stressed in [7].

Classifying all orbits of a permutation group means listing a set of representatives for the orbits, together with their respective stabilizer subgroups. Also, if an object is given, we can compute a group element that maps the given object to its representative in the classification. These kinds of problems are notoriously difficult, since they involve the isomorphism problem as a subproblem and isomorphism is usually NP-hard.

Breaking The Symmetry

While group theoretic algorithms to classify orbits are available, experience shows that many problems require a combination of methods to be solved efficiently. The method of *breaking the symmetry* allows to classify objects using subobjects that serve as a stepping stone. The subobjects are easier to classify, and it is possible to lift the classification of subobjects to the classification of the original objects. The method combines group theory with traditional "solvers." Here, we understand solvers as any kind of computational primitive to solve the problem of *finding* all objects that arise from a given *starter object*. Once these objects have been found, an additional isomorph rejection step is performed to solve the classification problem. The theory behind this is explained in [2], where the method is applied to the classification of packings in $\mathrm{PG}(3,3)$. It is important to realize that the method is very general, and can be applied to broad classes of problems.

The use of subobjects is well-known. In [11], homomorphisms of group actions are discussed. In [5] and [6], the technique of "breaking the symmetry" is developed. In [16], an algebraic algorithm to compute the orbit decomposition is presented. In *Orbiter*, many of these algorithms are combined, and an isomorph classification module based on the theory described in [2] is present. *Orbiter* offers some algorithms to solve these systems of equations but also allows to interface with third party software. The communication between *Orbiter* and the outside solvers can happen through files. Once the data from the solver is received, the isomorph classification module starts its job and computes the final list of isomorphism types together with the stabilizer groups. Data from the classification is stored in files to allow identifying objects of the givem type at a later point in time. This means that given an object, a group element can be computed that maps the object to one of the representatives from the classification. An implementation of Knuth's dancing links (DLX) [10] is available. Wassermann's algorithm [17] or any other suitable piece of software can be used as an external solver.

The underlying idea behind *Orbiter* is to provide isomorph classification for a variety of types of objects. To be able to handle things uniformly, we rely on the use of C++ function pointers to realize permutation group algorithms for arbitrary objects. The only requirement is that the object can be represented as a set (or set of sets). The entries of the set are integers that represent the components of the object. For instance, when classifying sets of points in a finite projective space subject to certain conditions, the components are projective points, and they are represented numerically. When classifying combinatorial designs, the objects consist of sets of subsets of a set X. These subsets are known as blocks, and they form the components of the object. We can use the lexicographic ordering of subsets of X to identify blocks with integers. The process of converting components into integers and integers into components is called *ranking* and *unranking*. Sometimes, the terms indexing or enumerating are used also. Basically, the possible components are mapped bijectively to an interval of integers. We require that rank and unrank functions to encode the object under consideration are available. For may types of combinatorial objects, such functions exist or can be devised easily. Using this kind of methodology, *Orbiter* is able to realize permutation groups acting on objects. The permutation group algorithms and the functionality for the specific object are completely separate. The group does not know what objects it is acting on, and the objects does not know what group is acting on them. This methodology makes the code easily adaptable to different actions. The most basic group actions are that of the symmetric group acting on a set, and the projective linear group acting on projective space (as well as the affine group acting on a vector space). From these basic actions, one can define induced actions in various ways. For instance, the symmetric group induces an action on the k-subsets. The projective linear group induces an action on the Grassmannian of subspaces. Many other induced actions are available.

Orbiter's predecessor is DISCRETA [4], which is specialized to t-designs with prescribed groups of automorphisms, and comes with a graphical user interface. Since *Orbiter* applies to a much more general class of problems, there was no longer the possibility for a graphical user interface. Instead, the user of *Orbiter* will have to write code to facilitate the algorithms that are provided. *Orbiter* is available from the website [15].

Applications

Recently, *Orbiter* has been used to classify packings [2], unitals [1] and BLT-sets [3]. Other applications exist. Some are described in the Orbiter Manual.

Other Work

A general reference for the problem of classifying designs and codes is [9]. This book emphasizes the use of canonical forms, facilitated for instance via the software package nauty [14], or the partition backtrack approach [12]. Both of these algorithms are available through the computer algebra system MAGMA [13]. Nauty is also available through *Orbiter*. A different computer algebra system with an emphasis on Group Theory is GAP [8].

References

[1] John Bamberg, Anton Betten, Cheryl Praeger, and Alfred Wassermann. Unitals in the Desarguesian Projective Plane of Order Sixteen. To appear in Journal of Statistical Planning and Inference.

[2] Anton Betten. The Packings of PG(3, 3). Submitted to Journal of Combinatorial Designs.

[3] Anton Betten. Rainbow Cliques and the Classification of Small BLT-Sets. Accepted for the Proceedings of ISSAC 2013.

[4] Anton Betten, Reinhard Laue, and Alfred Wassermann. DISCRETA, a tool for constructing t-designs. In: Computer Algebra Handbook, Edited by Johannes Grabmeier, Erich Kaltofen, Volker Weispfennig, Springer 2003, pp 372-375.

[5] Cynthia A. Brown, Larry Finkelstein, and Paul Walton Purdom, Jr. Backtrack searching in the presence of symmetry. In *Applied algebra, algebraic algorithms and error-correcting codes (Rome, 1988)*, volume 357 of *Lecture Notes in Comput. Sci.*, pages 99–110. Springer, Berlin, 1989.

[6] Cynthia A. Brown, Larry Finkelstein, and Paul Walton Purdom, Jr. Backtrack searching in the presence of symmetry. *Nordic J. Comput.*, 3(3):203–219, 1996.

[7] P. J. Cameron and J. H. van Lint. *Designs, graphs, codes and their links*, volume 22 of *London Mathematical Society Student Texts*. Cambridge University Press, Cambridge, 1991.

[8] GAP – Groups, Algorithms, and Programming, Version 4.4. The GAP Group, Aachen, Germany and St. Andrews, Scotland, 2004.

[9] P. Kaski and P. Östergård. *Classification algorithms for codes and designs*, volume 15 of *Algorithms and Computation in Mathematics*. Springer-Verlag, Berlin, 2006.

[10] D. E. Knuth. Dancing links. *eprint arXiv:cs/0011047*, November 2000. in Davies, Jim; Roscoe, Bill; Woodcock, Jim, Millennial Perspectives in Computer Science: Proceedings of the 1999 Oxford-Microsoft Symposium in Honour of Sir Tony Hoare, Palgrave, pp. 187-214.

[11] R. Laue. Construction of combinatorial objects—a tutorial. *Bayreuth. Math. Schr.*, 43:53–96, 1993. Konstruktive Anwendungen von Algebra und Kombinatorik (Bayreuth, 1991).

[12] J.S. Leon. Partitions, refinements, and permutation group computation. In *Groups and computation, II (New Brunswick, NJ, 1995)*, volume 28 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 123–158. Amer. Math. Soc., Providence, RI, 1997.

[13] Magma. The Computational Algebra Group within the School of Mathematics and Statistics of the University of Sydney, 2004.

[14] Nauty User's Guide (Version 2.4), Brendan McKay, Australian National University, Nov 4, 2009.

[15] Orbiter – A Program To Classify Discrete Objects,
http://www.math.colostate.edu/∼betten/orbiter/orbiter.html, Anton Betten, 2013.

[16] B. Schmalz. t-Designs zu vorgegebener Automorphismengruppe. *Bayreuth. Math. Schr.*, 41:1–164, 1992. Dissertation, Universität Bayreuth, Bayreuth, 1992.

[17] Alfred Wassermann. Finding simple t-designs with enumeration techniques. *J. Combin. Des.*, 6(2):79–90, 1998.

Jenks Prize 2013 Award Citation

Communicated by Erich Kaltofen

The 2013 Richard D. Jenks Memorial Prize for Excellence in Software Engineering Applied to Computer Algebra was announced by members of the Prize Selection Committee, Mark Giesbrecht representing its chair Erich Kaltofen, at ISSAC in Boston, MA, on June 28, 2013 to have been awarded to **Professor William Arthur Stein of the Sage Project at the University of Washington**. The Prize includes a Plaque and has a monetary award of $1,000.

William Stein is the creator of the Sage platform (`www.sagemath.org`), which, based on the Python programming language, makes available for integrated use a number of computer algebra programs such as Magma, Maple, Mathematica, MuPAD, and which includes Axiom, GAP, GP/PARI, LinBox, Macaulay2, Maxima, Octave, and Singular, among others. Sage also provides its own programming language Cython.

William Stein has been able to attract hundreds of followers to Sage, who have contributed to the open source base of Sage and who have already met for dozens of Sage Days on several continents. Some of the algorithms in Sage constitute the fastest implementations for the given problems.

Sage's free software philosophy has attracted thousands of users all over the world to undertake computations for mathematical research and development.

Mark Giesbrecht ACM Distinguished Scientist

On November 20, 2013, ACM has selected distinguished members in recognition of their individual achievements and contributions. ACM Sigsam is proud to announce that our colleague **Mark Giesbrecht** from the University of Waterloo has been selected as Distinguished Scientist.

Quoting from the corresponding ACM press release:

> ACM (the Association for Computing Machinery) has named 40 Distinguished Members for their individual contributions and their singular impacts on the vital field of computing. Their achievements have advanced the science, engineering, and education of computing, and highlight the widening role that computing plays in a range of disciplines and domains around the globe. The 2013 Distinguished Members hail from universities in Denmark, Japan, Israel, Italy, China, and the United Kingdom in addition to North America, and from leading international corporations and research institutions.
>
> ACM President Vinton G. Cerf described the recipients as "the problem solvers, prophets, and producers who are powering the future of the digital age." He noted that these ACM members "are the driving force for enabling the computing community to change how we live and work. They demonstrate the advantages of ACM membership, which empowers self-improvement and inspires a bold vision for their own careers as well as their impact on the future."
>
> The ACM Distinguished Member program can recognize the top 10 percent of ACM worldwide membership based on professional experience as well as significant achievements in the computing field.

`http://www.acm.org/press-room/news-releases/2013/distinguished-2013`

Abstracts of Recent Doctoral Dissertations in Computer Algebra

Each month we are pleased to present abstracts of recent doctoral dissertations in Computer Algebra and Symbolic Computation. We encourage all recent Ph.D. graduates (and their supervisors), who have defended in the past two years, to submit their abstracts for publication in CCA.
Please send abstracts to the CCA editors `<editors_SIGSAM@acm.org>` *for consideration.*

Author: Nan Li
Title: On Isolated Singular Solutions in Polynomial System Solving
Institution: Chinese Academy of Sciences, Beijing, China
Thesis Advisor: Lihong Zhi
Defended: June 2013

Nowadays, polynomial models are ubiquitous and widely applied across the engineering and sciences, such as in robotics, coding theory, optimization, mathematical biology, computer vision, game theory, statistics, machine learning, control theory, cryptography, and numerous others. A main challenge in algebra and geometry computing is to identify and tackle singular points, which naturally occur when computing the topology of implicit curves or surfaces, the intersection of parametric surfaces in geometric modeling.

A numerical approximation is usually computed to identify an isolated solution of a polynomial system. In practice, we often need to improve the quality of numerical approximations, but numerical methods such like Newton's method converge slowly at singular solutions (or not converge). On the other hand, it is well known that to certify whether a polynomial system has an isolated singular solution is an ill-posed problem, since arbitrary small perturbations of coefficients may transform the singular solution into a cluster of simple roots (or even make it disappear). Therefore, it is hardly possible to verify this problem, if not the entire computation is performed without any rounding error (exact arithmetics).

In this thesis, we first introduce the local dual space for characterizing an isolated singular solution of a polynomial system. By employing some regularization and reduction techniques, we present a novel algorithm for computing a reduced basis of such a space for the special case of breadth one. The algorithm also works for inputs only with limited accuracy, and is efficient both in time and memory use. Moreover, it leads to a parametric representation for a reduce basis (multiplicity structure) of the local dual space.

Based on such a parametric representation and presolving a regularized least squares problem, we propose a regularized Newton's method for refining an approximate singular solution of a given polynomial system. By a careful analysis, we prove the quadratic convergence of the algorithm if the numerical approximation is close to a breadth-one isolated singular solution.

By introducing some well-chosen smoothing parameters to the given system, we develop an improved deflation technique, which derives a square and regular augmented system from an isolated singular solution in a finite number of deflations. Based on this technique, we propose an algorithm for computing verified error bounds such that a slightly perturbed polynomial system is guaranteed to possess an isolated singular solution within the computed bounds.

Author: Maximilian Jaroschek
Title: Removable Singularities of Ore Operators
School: RISC, Johannes Kepler University, Linz
Thesis Advisor: Manuel Kauers
Defended: December 2013

Ore algebras are an algebraic structure used to model many different kinds of functional equations like differential and recurrence equations. The elements of an Ore algebra are polynomials for which the multiplication is defined to be usually non-commutative. As a consequence, Gauß' lemma does not hold in many Ore polynomial rings and hence the product of two primitive Ore polynomials is not necessarily primitive. This observation leads to the distinction of non-removable and removable factors and to the study of desingularizing operators.

Desingularization is the problem of finding a left multiple of a given Ore operator in which some factor of the leading coefficient of the original operator is removed. We derive a normal form for such left factors and unify known results for differential and shift operators into one desingularization algorithm. Furthermore, we analyze the effect of removable and non-removable factors on computations with Ore operators.

The set of operators of an Ore algebra that give zero when applied to a given function forms a left ideal. The cost of computing an element of this ideal depends on the size of the coefficients (the degree) and the order of the operator. In order to be able to predict or reduce these costs, we derive an order-degree curve. For a given Ore operator, this is a curve in the (r, d)-plane such that for all points (r, d) above this curve, there exists a left multiple of order r and degree d of the given operator. We show how desingularization yields order-degree curves which are extremely accurate in examples. When computed for the generator of an operator ideal from applications like physics or combinatorics, the resulting bound is usually sharp.

The generator of a left ideal in an Ore polynomial ring is the greatest common right divisor of the ideal elements, which can be computed by the Euclidean algorithm. Polynomial remainder sequences contain the intermediate results of the Euclidean algorithm when applied to (non-)commutative polynomials. The running time of the algorithm is dependent on the size of the coefficients of the remainders. Different methods have been studied to make these as small as possible. The subresultant sequence of two polynomials is a polynomial remainder sequence in which the size of the coefficients is optimal in the generic case, but when taking the input from applications, the coefficients are often larger than necessary. We generalize two improvements of the subresultant sequence to Ore polynomials, in which we show that the non-removable factors of the greatest common right divisor appear as content. Based on this result we show how to divide out this content during the Euclidean algorithm and derive a new bound for the minimal coefficient size of the remainders. Our approach also yields a new proof for the results in the commutative case, providing a new point of view on the origin of the extraneous factors of the coefficients.

Recent and Upcoming Events

March 31–April 2, 2014
Functional Equations in Limoges (FELIM'14)
Limoges, France
Organizers: Moulay Barkatou, Thomas Cluzeau, and Jacques-Arthur Weil

---

May 15–17, 2014
6. Tagung Fachgruppe Computeralgebra
Kassel, Germany
Organizers: Wolfram Koepf (local organization)
Dates: Registration with talk: March 15, 2014; Registration without talk: May 1, 2014
Website: `http://www.fachgruppe-computeralgebra.de/tagung-kassel-2014/`

May 21–22, 2014
17th Workshop on Computer Algebra
Dubna, Russia
Organizers: Sergei Abramov, Vladimir Gerdt, Alla Bogolubskaya
Dates: Submission: May 11, 2014 (for visa arrangements: March 20, 2014)
Website: `http://compalg.jinr.ru/Dubna2014/index.html`

June 18–20, 2014
XIV Encuentro de Algebra Computacional y Aplicaciones (EACA 2014)
Barcelona, Spain
Dates: Submission: February 21, 2014; Notification: April 15, 2014; Final Version: May 10, 2014
Website: `http://www.ub.edu/eaca2014`

June 23–25, 2014
9th GASCom conference on random generation of combinatorial structures
Bertinoro, Italy
Organizers: Marilena Barnabei (scientific chair), Flavio Bonetti (organization chair)
Dates: Submission: February 28, 2014; Notification: April 11, 2014; Final version: May 9, 2014.
Website: `http://gascom2014.dm.unibo.it/`

July 2–4, 2014
8th International Workshop on Parallel Matrix Algorithms and Applications (PMAA'14)
Lugano, Switzerland
Organizers: Peter Arbenz, Ahmed Sameh, Rolf Krause, Olaf Schenk
Dates: Session proposals: March 15, 2014; Talk submission: March 30, 2014; Notification: April 6, 2014
Website: `http://pmaa14.ics.usi.ch/`

July 7–11, 2014
Conferences on Intelligent Computer Mathematics
Coimbra, Portugal

Organizers:	Stephen Watt (general chair)
Dates:	Workshop proposals: January 17, 2014; Conference submissions: February 28 (abstracts) and March 7 (full papers)
Website:	`http://cicm-conference.org/2014/cicm.php`

July 23–25, 2014
39th International Symposium on Symbolic and Algebraic Computation (ISSAC'14)
Kobe, Japan

Organizers:	Kosaku Nagasaka and Franz Winkler (general chairs), Agnes Szanto (PC chair)
Dates:	Submission: January 12, 2014 (abstracts) and January 19, 2014 (full papers), Notification: March 30, 2014, Final Version: April 30, 2014
Website:	`http://www.issac-symposium.org/2014`

July 28–31, 2014
Symbolic Numeric Computation (SNC 2014)
Shanghai, China

Organizers:	Lihong Zhi (general chair), Stephen Watt (PC chair), Zhengfeng Yang (Local chair)
Dates:	Submission: March 24, 2014; Notification: April 28, 2014; Final version: May 19, 2014
Website:	`http://symbolic-numeric-computation.org/snc-2014/`

August 5–9, 2014
The 4th International Congress on Mathematical Software (ICMS)
Seoul, Korea

Organizers:	Chee K. Yap (general chair), Hoon Hong (program chair), Deok-Soo Kim (local chair)
Dates:	Session proposals: Jan 31 2014
Website:	`http://voronoi.hanyang.ac.kr/icms2014/`

September 8–12, 2014
16th Computer Algebra in Scientific Computing (CASC 2014)
Warsaw, Poland

Organizers:	Vladimir P. Gerdt and Werner M. Seiler (General Chairs); Wolfram Koepf and Evgenii V. Vorozhtsov (PC Chairs)
Dates:	Submission: April 06, 2014; Notification: May 25, 2014; Final version: June 08, 2014
Website:	`http://www14.in.tum.de/CASC2014/`

Call for Papers

39th International Symposium on Symbolic and Algebraic Computation (ISSAC 2014)

http://www.issac-symposium.org/2014/
July 23–25 2014, Kobe, Japan

The International Symposium on Symbolic and Algebraic Computation is the premier conference for research in symbolic computation and computer algebra. ISSAC 2014 is the 39th meeting in the series. The conference traditionally presents a range of invited speakers, tutorials, poster sessions and software demonstrations with a centre-piece of contributed research papers.

ISSAC 2014 is held July 23-25, 2014 at Kobe University, Japan. ISSAC 2014 is affiliated with "Kobe Computing Week 2014", an event of Academic Exchange Weeks, Graduate School of Human Development and Environment, Kobe University.

ISSAC 2014 is one of satellite conferences of ICM 2014 (International Congress of Mathematicians), Korea. Also, SNC 2014 (Symbolic-Numeric Computation), Shanghai, China, is a satellite conference of ISSAC 2014.

Important Dates
Event:

- Workshops and tutorials: July 21–22, 2014
- ISSAC 2014 conference: July 23–25, 2014

Regular papers:

- Paper abstract submission: January 12, 2014
- Full paper submission deadline: January 19, 2014
- Notification of acceptance/rejection: March 30, 2014
- Final version due: April 30, 2014

Posters and Software presentations:

- Abstract submission: April 20, 2014
- Notification: May 16, 2014
- Final version: June 6, 2014

Conference Topics:

ISSAC 2014 invites the submission of original research contributions to be considered for publication and presentation at the conference. All areas of computer algebra and symbolic mathematical computation are of interest. These include, but are not limited to:
Algorithmic aspects:

- Exact and symbolic linear, polynomial and differential algebra
- Symbolic-numeric, homotopy, perturbation and series methods
- Computational algebraic geometry, group theory and number theory

- Computer arithmetic
- Summation, recurrence equations, integration, solution of ODEs & PDEs
- Symbolic methods in other areas of pure and applied mathematics
- Complexity of algebraic algorithms and algebraic complexity

Software aspects:

- Design of symbolic computation packages and systems
- Language design and type systems for symbolic computation
- Data representation
- Considerations for modern hardware
- Algorithm implementation and performance tuning
- Mathematical user interfaces

Application aspects:

Applications that stretch the current limits of computer algebra algorithms or systems, use computer algebra in new areas or new ways, or apply it in situations with broad impact.

Invited Speakers:

Nokiko Arai, David Stoutemyer, and Bernd Sturmfels

Organizers:

General Chairs: Kosaku Nagasaka and Franz Winkler
PC Chair: Agnes Szanto
Local Chair: Kosaku Nagasaka
Publicity: Ekaterina Shemyakova
Treasurer: Akira Terui
Poster Chair: Wen-shin Lee
Software Presentations Chair: Daniel Lichtblau
Tutorial Chair: Tetsu Yamaguchi
Workshop Chair: Takuya Kitamoto
Webmaster: Masaru Sanuki
Program Committee: Shaoshi Chen (Chinese Academy of Sciences, China), Carlos D'Andrea (U. Barcelona, Spain), Wayne Eberly (U. Calgary, Canada), Ioannis Emiris (U. Athens, Greece), Jean-Charles Faugere (INRIA, France), Mark Giesbrecht (U. Waterloo, Canada), Jonathan Hauenstein (North Carolina State University, USA), Evelyne Hubert (INRIA, France), Alexander Hulpke (Colorado State University, USA), Gabor Ivanyos (MTA SZTAKI, Hungary), Joseph Maurice Rojas (Texas A&M University, USA), Julio Rubio (Universidad de La Rioja, Spain), Mohab Safey el Din (Univ. Pierre and Marie Curie, France), Tateaki Sasaki (University of Tsukuba, Japan), Yosuke Sato (Tokyo U. of Science, Japan), Josef Schicho (RISC, Austria), Michael Singer (North Carolina State University, USA), Elena Smirnova (Texas Instruments, USA), Agnes Szanto (North Carolina State University, USA), Chee Yap (NYU, USA)

Call for Session Proposals

4th International Congress on Mathematical Software (ICMS 2014)

http://voronoi.hanyang.ac.kr/icms2014
August 5–9 2014, Seoul Korea
Satellite Conference of ICM 2014
http://www.icm2014.org

The 4th International Congress on Mathematical Software will consist of several topical sessions. Each session will provide an overview of the challenges, achievements and progress in a subfield of mathematical software research, development and use. The program committee will consist of the session organizers. We solicit session proposals.

You are invited to propose a session if you

are active in mathematical software research, development and use, want to serve the research community by nurturing and facilitating mathematical software work in your area, and would like to focus only on the scientific matters in the organization (not on other matters such as administrative, logistic, etc).

How to propose a session

- Prepare a session proposal with the following contents.
  - title of the session
  - name(s) of the organizer(s), with contact addresses and emails
  - aim and scope of the session (at most 150 words)
- Submit it
  - to the program chair (Hoon Hong)
  - by email hong@ncsu.edu
  - at latest by Jan 31 2014.
- The decision on the proposal will be made by the program chair, the general chair and the advisory committee within a week of the submission.

How to organize a session

- Maintain a session web page (a template will be provided).
- Send a call for abstracts to the potential speakers in the topic area of the session (a template will be provided).
- Review the submitted abstracts and make decision on their acceptance, as soon as each one arrives.

- Complete the process by May 15 2014.
- During the meeting, chair the session.

Format of a session

- A session will consist of one or more time slots.
- A time slot will consist of about 5 talks.
- Each talk will last about 30 minutes (including Q/A).
- We encourage that each session begins with one general overview talk (may be given by a session organizer).

Topics for sessions

Any mathematical software topics are welcome. For suggestions, see "Call for Session Proposals" at `http://voronoi.hanyang.ac.kr/icms2014`