A Cache-Oblivious Engineering of the G2V Algorithm for Computing Gröbner Bases

Fatima K. Abu Salem
Computer Science Department, American University of Beirut
fatima.abusalem@aub.edu.lb

We develop a cache-oblivious variant of Gao et al.'s incremental algorithm (G2V) for computing Gröbner bases [2], by applying the funnel heap implementation of a priority queue. This improves on two main functionalities of G2V that otherwise pose a bottleneck for memory performance. The resulting design is geared towards high performance computing with reference to the memory hierarchy.

Incremental algorithms for computing Gröbner bases have improved over classical algorithms, typically as in Buchberger's, by integrating mechanisms to detect redundant polynomials arising in the reductions. G2V further makes use of polynomials that reduce to zero to identify even more "useless" polynomials. Experimental results in [2] show that G2V indeed outperforms other existing incremental algorithms like F5 and F5C. Let \( F \) be any field and \( R = F[x_1, \ldots, x_n] \), and fix an arbitrary monomial order \( \prec \) on \( R \).

At a typical iterative step of G2V, a Gröbner basis \( G = \{ f_1, f_2, \ldots, f_m \} \) for an ideal \( I \) in \( R \) with respect to \( \prec \) is already computed, and one wishes to compute a Gröbner basis for the new ideal \( \langle I, g \rangle \), where \( g \) is a given polynomial in \( R \). The regular top-reduction that G2V performs is similar to that of F5, and a new kind of reduction called super top-reduction is introduced. The version we recall here only produces a Gröbner basis for \( \langle I, g \rangle \), though G2V is also able to produce a Gröbner basis for the colon ideal \( (I : g) \). Let \( \text{LM}(u) \) denote the leading monomial of \( u \in R \). Given \( (u_1, v_1) \) and \( (u_2, v_2) \) in \( R^2 \), let \( t = \text{lcm}(\text{LM}(v_1), \text{LM}(v_2)) \) and \( t_i = t/\text{LM}(v_i) \). Find \( t_i \text{LM}(u_i) = \max(t_1 \text{LM}(u_1), t_2 \text{LM}(u_2)) \), for \( i = 1 \) or \( 2 \). Then \( t_i \text{LM}(u_i) \) is called the \( J \)-signature of \( (u_1, v_1) \) and \( (u_2, v_2) \), and \( t_i(u_i, v_i) \) is its \( J \)-pair. For the sake of brevity, we present in Alg. 1 below only the main loop of G2V appearing in [2], and we refer the reader to the original paper for the exact description of regular and super top-reductions. Hereafter, let \( U \) denote a list of polynomials for \( u \), \( V \) a list of polynomials for \( v \), \( H \) a list for \( \text{LM}(u) \) that form part of the output, and \( JP \) a list of pairs \( (t, i) \) where \( t(u_i, v_i) \) is the \( J \)-pair of \( (u_i, v_i) \) and some \( (u_j, v_i) \). We also omit the proper initialisations of each of these data structures (see [2]). We identify the two junctures in Alg. 1 where memory performance suffers. The first one concerns the successive reduction operations which take place via polynomial division, introducing expression swell in the intermediary values. The second concerns the set \( JP \) and the fact that it ought to be maintained dynamically, with support for Insert, Extract-Min and Delete operations (e.g. Steps 1, 2, and 4 in Alg. 1). Both of these requirements are encountered within the main loop of G2V, which exacerbates their effect on memory performance. A priority queue (PQ) data structure is able to address both problems. Particularly, a PQ improves on the memory performance of polynomial division (e.g., see [3]), whilst it improves on the computational cost for maintaining \( JP \) under Insert and Extract-min. Several cache-oblivious PQ implementations exist as opposed to the ordinary binary heap. Of all, the funnel heap achieves optimal amortised I/O cost for each of Insert and Extract-Min [1]. Of interest to us is the SWEEP function, which pushes the elements of
We observe that the funnel heap can improve on polynomial division as follows. In [1], it is shown that the amortised cost for Insert and Extract-Min can be at $O\left(\frac{1}{B} \log_{M/B} N_i\right)$, where $B$ and $M$ denote the cache-line length and cache size respectively, and $N_i = j - i$ if the $i'$th inserted element is removed by Extract-Min prior to the $j$'th insertion. We recall that the sequence of inserts required by polynomial division via a PQ occurs in decreasing order of monomial degrees (see [3]), followed immediately by Extract-Max. Heuristically speaking, this increases the likelihood that higher order elements which are inserted earlier, are short-lived in the Max-PQ. In such cases, $N_i << N$, where $N$ is the maximum occupancy of the Max-PQ. Polynomial division can also benefit from the fact that an Extract-Max is able to deliver in one batch, all elements whose key is maximal, without violating the structure of the heap. Also, access to these elements as they are being removed happens contiguously in memory, which respects spatial locality. All of this could not be achieved using the binary heap. The funnel heap can also improve on maintaining the list $JP$ as follows. We maintain this list as a Min-PQ under $(\prec)$ on $R$. Like the binary heap, the funnel heap only supports Insert and Extract-Min but no Delete operations. Whilst random deletes to an ordinary binary heap destroy the heap structure, we are able to achieve random deletes by deferring them and then overlapping them with a number of the funnel heap's functionalities, (1) without destroying its structure and (2) at no degradation in asymptotic performance. Each element residing in the PQ is assigned an extra field in addition to their own priority key, to denote a flag whether or not it ought to be deleted. If an element is marked for deletion during some iteration, it can still reside in the PQ, and its deletion can be deferred. Such elements can be eventually removed from the queue upon two instances. Firstly, while Extract-Min is returning an element marked for deletion, this element is evicted but a new minimum is sought, with the iteration stopping upon the first minimum that has not been marked for deletion. Secondly, while a SWEEP is merging elements residing in the PQ into one sorted stream, the merging is modified so that an element marked for deletion gets overwritten by elements with larger keys and not marked for deletion.

**Algorithm 1** The G2V Algorithm

```
repeat
1. Extract $(t, i)$ with minimal signature from $JP$. Use regular top-reductions to reduce $t(u_i, v_i)$ repeatedly by the pairs in $(U, V)$, until a pair $(u, v)$ is obtained that is not regular top-reducible.

   if $v = 0$ then
   2. Append $\text{LM}(u)$ to $H$ and delete each $(t, \ell)$ in $JP$ whose signature $t\text{LM}(u_\ell)$ is divisible by $\text{LM}(u)$.
   else if $v \neq 0$ and $(u, v)$ is super top-reducible by some pair $(u_j, v_j)$ in $(U, V)$ then
   3. Discard the pair $(t, i)$.
   else
   4. Append $u$ to $U$ and $v$ to $V$. Form new $J$-pairs of $(u, v)$ and $(u_j, v_j)$, for $1 \leq j \leq |U| - 1$. Insert into $JP$ all such $J$-pairs whose signature are not reducible by monomials in $H$.

end if
until $JP$ is not empty
```

**References**


On the complexity of computing the GCD of two polynomials via Hankel matrices

Skander Belhaj\textsuperscript{1,2} and Haïthem Ben Kahla\textsuperscript{2}

\textsuperscript{1} Manouba University, ISAMM, Campus Universitaire de la Manouba, 2010 Tunis, Tunisia
\textsuperscript{2} University of Tunis El Manar, ENIT-LAMSIN, 1002 Tunis Belvédère, Tunisia
skander.belhaj@lamsin.rnu.tn & haithembk@gmail.com

Abstract

This paper is devoted to present a revised algorithm that permits to preserve the beautiful relation between the classical Euclidean algorithm and the block diagonalization of Hankel matrices for two non-coprime polynomials. Our algorithm for Greatest Common Divisor (GCD) computation which has a cost of $O(n^2)$ arithmetic operations is tested for several sets of polynomials. A complexity comparison is given with respect to other existing methods based on structured matrix computations.

1 Introduction

A basic problem in algebraic computing is the evaluation of polynomial GCD: given the coefficients of two polynomials $u(x) = \sum_{i=0}^{n} u_i x^i$ and $v(x) = \sum_{i=0}^{m} v_i x^i$ where $n > m$, compute the coefficients of their greatest common divisor. The oldest algorithm whose computes the GCD is the classical Euclidean algorithm applied with $u(x)$ and $v(x)$ which generates the quotients and remainders sequences. The last nonzero remainder is the GCD($u(x), v(x)$). Naturally, the classical algorithm proves to be inefficient when applied with perturbed polynomials. In order to avoid this problem, Noda and Sasaki \cite{5} are among the first to introduce some numerical efficient versions via Euclidean method; so the complexity becomes cubic. Several papers have been consecrated to describe the natural relation between the Euclidean algorithm and the block diagonalization of Hankel matrix. Indeed, Bini and Gemignani \cite{3} introduce a parallel algorithm for the block diagonalization of the Bezoutian $JB(u, v)J$ and generate the quotients and remainders sequences. This block diagonalization of $JB(u, v)J$ can be given from the block diagonalization of $H(u, v)$ and vice-versa based on the following relation : $B(u, v) = B(u, 1)H(u, v)B(u, 1)$. Lately \cite{1}, the quotients and remainders sequences are computed via an approach different of the classical version based on an approximate block diagonalization of the Hankel matrix $H(u, v)$ such that $u$ and $v$ are coprime. Indeed, $A^tH(u, v)A = D$ where $D$ is a block diagonal matrix and every block is a lower triangular Hankel matrix. In this paper, we propose a revised algorithm which compute the GCD via an approximate block diagonalization of a Hankel matrix. Thus, the beautiful relation between the Euclidean algorithm and the approximate block diagonalization of a Hankel matrix $H(u, v)$ is preserved with non-coprime departure polynomials. Finally, we illustrate our approach by an arithmetic complexity comparison with respect to other existing methods based on Bézout \cite{3, 2} and Sylvester \cite{6, 7} matrices.

2 Algorithm and complexity comparison

Let’s now derive a revised version of the algorithm introduced in \cite[Algorithm 3.4]{1} which exhibited initially a fast method to block-diagonalize of Hankel matrix.
Algorithm 1 Given $u(x) = \sum_{i=0}^{n} u_i x^i$ and $v(x) = \sum_{i=0}^{m} v_i x^i$ two polynomials of degree $n$ and $m$, respectively, where $m < n$; this algorithm computes the approximate GCD polynomial.

1. Construct $H(u,v) = H(\varepsilon_1, \ldots, \varepsilon_{n-m-1}, h_{n-m}, \ldots, h_{2n-1})$.
2. Define an upper triangular Toeplitz matrix $t = uT(h_{n-m}, \ldots, h_{2n-m-1})$.
3. Compute $t^{-1}$ and $t^{-1}H(u,v)t^{-1}$.
4. Set $h'_{11} = h'(1 : n - m, 1 : n - m)$ and $h'_{22} = h'(n - m + 1 : n, n - m + 1 : n)$.
5. Recover the coefficients of the quotient and the remainder polynomial.
6. Recursively apply Algorithm 1 to $h = H(h'_{22}(1 : m, 1)h'_{22}(m : 1))$ and extract the GCD.

Theorem 1 The GCD evaluation of two polynomials via Algorithm 1 requires $O(3n^2)$ flops.

Idea on the cost of Algorithm 1. Steps 1-3. construct $H(u,v): O(3n^2 - 2)$, compute $t^{-1}$ via revised Bini [1]: $O((4n - 2p) \log(4n - 2p))$, and compute $t^{-1}ht^{-1}: O((4n) \log(2n))$. Step 5. and 6. compute the remainders matrix: $O((2n) \log(2n))$ and verify the last remainder: $O((n + 1) \log(n + 1) + (n + 1))$. Total: $O(3n^2 - 2 + (4n - 2p) \log(4n - 2p) + (6n) \log(2n) + (n + 1) \log(n + 1) + (n + 1)) = O(3n^2)$.

Remark 1 The numerical stability for a triangular Toeplitz matrix inversion via revised Bini algorithm is shown [4]. Moreover, Algorithm 1 tested for several non-coprime polynomials is numerically stable [2].

Remark 2 As in [7], we can extract from Algorithm 1 rank and null space of a Hankel matrix.

Let’s now explore a complexity comparison between our approach and other methods based on Bézout and Sylvester matrices in one dimension (See [7]):

<table>
<thead>
<tr>
<th>SVD</th>
<th>RRQR</th>
<th>QRCP</th>
<th>Gj</th>
<th>m.Gj</th>
<th>LU</th>
<th>m.LU</th>
<th>QR</th>
<th>m.QR</th>
<th>Disp.rank</th>
<th>Bézout</th>
<th>Hankel</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{2}{3}n^3$</td>
<td>$\frac{16}{3}n^3$</td>
<td>$\frac{16}{3}n^3$</td>
<td>$\frac{1}{6}n^3$</td>
<td>$\frac{1}{6}n^3$</td>
<td>$\frac{1}{3}n^3$</td>
<td>$\frac{1}{3}n^3$</td>
<td>$\frac{1}{3}n^3$</td>
<td>$\frac{1}{3}n^3$</td>
<td>$O(n^2)$</td>
<td>$7n^2$</td>
<td>$3n^2$</td>
</tr>
</tbody>
</table>

The methods via Hankel and Bézout [2] are tested for several sets of polynomials. The following table shows the flops of Algorithm 1 with respect to existing algorithms based on [6, Examples 1 and 4]:

<table>
<thead>
<tr>
<th></th>
<th>LU</th>
<th>m.LU</th>
<th>QR</th>
<th>m.QR</th>
<th>Bézout</th>
<th>Hankel</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 7$, $m = 4$</td>
<td>1321</td>
<td>561</td>
<td>2298</td>
<td>1218</td>
<td>960</td>
<td>333</td>
</tr>
<tr>
<td>$n = 16$, $m = 4$</td>
<td>7772</td>
<td>2096</td>
<td>12237</td>
<td>4078</td>
<td>4956</td>
<td>1258</td>
</tr>
</tbody>
</table>

References

Partition Analysis via Polyhedral Geometry

Felix Breuer\textsuperscript{(1)}, Zafeirakis Zafeirakopoulos\textsuperscript{(2)}

\textsuperscript{(1)} Department of Mathematics, San Francisco State University, funded by DFG grant BR 4251/1-1
\textsuperscript{(2)} Research Institute for Symbolic Computation (RISC) / DK-compmath, funded by Austrian Marshall Plan Foundation and Austrian Science Fund (FWF)
felix@fbreuer.de, zafeirakopoulos@risc.jku.at

September 28, 2012

Linear Diophantine systems appear in many different areas of mathematics (from number theory and combinatorics to optimization) and computer science (i.e. compiler theory). Mathematicians are concerned with their solution for more than 2 millenia, since the time of Diophantus. Partition analysis is a general methodology for the treatment of linear Diophantine systems. The Ω\textsubscript{≥} operator, the central tool of partition analysis introduced by MacMahon 100 years ago, has received attention in recent years. There have been various theoretical improvements of algorithms implementing the Ω\textsubscript{≥} operator. Andrews, Paule and Riese, with the Omega package\cite{2}, gave a completely algorithmic implementation of the Ω\textsubscript{≥} operator powered by symbolic computation. At the same time, significant progress have been made in the geometric theory of lattice point enumeration. In this work, we connect these two branches of research: We give a new algorithm for computing the Ω\textsubscript{≥} operator using geometric methods.

The problem under consideration is “Given \( A \in \mathbb{Z}^{m \times n} \) and \( b \in \mathbb{Z}^m \) find all \( x \in \mathbb{N}^n \) satisfying \( Ax \geq b \)”. We will illustrate through an example how partition analysis works. The example of a linear inhomogeneous Diophantine inequality is essentially the building block of the algorithm.

\textbf{Example 1.} Let \( A = \begin{bmatrix} 2 & 3 & -5 \end{bmatrix} \) and \( b = \begin{bmatrix} 4 \end{bmatrix} \). Find all \( x \in \mathbb{N}^3 \) satisfying \( Ax \geq b \).

Our goal is to obtain the generating function \( g(S) = \sum_{z \in S} z^b \) (multi-index notation) of the solution set \( S \). A reformulation of the problem using formal power series is \( g(S) = \sum_{x, y, z \in \mathbb{N}} \lambda^{2x+3y-5z-4} z^x y^y z^z \). Introducing an extra variable to encode the inequality we have \( \sum_{x, y, z \in \mathbb{N}} \lambda^{2x+3y-5z-4} z^x y^y z^z \) requiring the exponent of \( \lambda \) to be non-negative in the monomials that appear in the generating function of the solution set. We take the definition of the Ω\textsubscript{≥} operator from [2]:

\textbf{Definition 1.} The Ω\textsubscript{≥} operator is defined on functions with absolutely convergent multisum expansions in an open neighborhood of the complex circles \(|\lambda_i| = 1\) and the action is given by

\[ \Omega_{\geq} \sum_{s_1=-\infty}^{\infty} \sum_{s_2=-\infty}^{\infty} \cdots \sum_{s_r=-\infty}^{\infty} A_{s_1,s_2,\ldots,s_r} \lambda_1^{s_1} \lambda_2^{s_2} \cdots \lambda_r^{s_r} := \sum_{s_1=0}^{\infty} \sum_{s_2=0}^{\infty} \cdots \sum_{s_r=0}^{\infty} A_{s_1,s_2,\ldots,s_r} \]

Based on the geometric series expansion formula \((1 - z)^{-1} = \sum_{x \geq 0} z^x\) and the definition of Ω\textsubscript{≥} we transform the series into a rational function: \( g(S) = \Omega_{\geq} \sum_{x, y, z \in \mathbb{N}} \lambda^{2x+3y-5z-4} z^x y^y z^z = \Omega_{\geq} \frac{\lambda^{-4}}{(1-z_1 \lambda^2)(1-z_2 \lambda^3)(1-z_3 \lambda^{-4})} \)

The last expression is the crude generating function for Example 1. If we can evaluate Ω\textsubscript{≥} on such an expression then we recursively eliminate λ’s in order to solve linear Diophantine systems.
In order to employ polyhedral geometry, we need to translate the problem to a problem about cones. Let $A_i$ denote the $i$-th column of $A$. Let $v_i = [e_i : A_i]^T \in \mathbb{Z}^n$ and $V = (v_1, \ldots, v_n) \in \mathbb{Z}^{(n+1) \times n}$, using multi-index notation and denoting concatenation by colon. Although there is a geometric way to eliminate multiple $\lambda$ at once, for simplicity and presentation clarity, in what follows we present an algorithm to eliminate a single $\lambda$ assuming $b > 0$. This is the building block of all traditional Partition Analysis implementations. Define the cone $C := \mathbb{R}_+(v_1, \ldots, v_n)$ spanned by the $v_i$, and the set $H_b = \{(x_1, \ldots, x_n, x_{n+1}) \mid b \leq x_{n+1}\}$ of vectors with last component greater than $b$. Since we have one $\lambda$, the column $A_i$ is an integer denoted by $a_i$. Then the rational function \( \frac{\lambda^b}{(1 - z_1\lambda^{a_1}) \cdots (1 - z_n\lambda^{a_n})} \) is the generating function of the set of integer points in the cone $C - b$. Computing \( \Omega_{\lambda^b} \frac{\lambda^b}{(1 - z_1\lambda^{a_1}) \cdots (1 - z_n\lambda^{a_n})} \) now amounts to computing the generating function \( g(\pi(C \cap H_b); z_1, \ldots, z_{n+1}) \), where $\pi$ denotes projection wrt the last coordinate, as follows:

1. Compute a vertex description of the tangent cones $K_v$ at the vertices of $C \cap H_b$.
2. Compute the generating functions $g(K_v, z_1, z_2, \ldots, z_{n+1})$, either using Barvinok’s algorithm [1] or by explicit formulas using modular arithmetic (similar to the expressions in [2]).
3. Substitute $z_{n+1} \mapsto 1$ to obtain $\lambda$-free generating functions (projection of the polyhedron).
4. Sum all the projected generating functions for the tangent cones. By Brion’s theorem [1], this yields the desired generating function $\Omega_{\lambda^b} \frac{\lambda^b}{(1 - z_1\lambda^{a_1}) \cdots (1 - z_n\lambda^{a_n})}$.

To illustrate the algorithm we present it with an example. Given an inequality $2x_1 + 3x_2 - 5x_3 \geq 4$, we want to compute $\Omega_{\lambda^b} \frac{\lambda^b}{(1 - z_1\lambda^{a_1})(1 - z_2\lambda^{a_2})(1 - z_3\lambda^{a_3})}$.

Our goal is to apply Brion’s theorem to the polyhedron $P := C \cap H_0$. Any polyhedron is the Minkowski sum of a cone and a polytope. In our case $P = \text{conv}(0, u_i : i \in \{1, \ldots, l\}) + \mathbb{R}_+ \{w_{i,j} : i \in \{1, \ldots, l\}, j \in \{l+1, \ldots, n\}\}$ where $u_i = \frac{b}{a_i}v_i = (\frac{b}{a_i}, \ldots, b)$ and $w_{i,j} = a_i v_i + a_j v_j = (\ldots - a_j \ldots a_i \ldots 0)$.

The generators of $P$ are linear combinations of the generators $v_i$ that are pointing “up” ($a_i \geq 0$). The the vertices of $P$, denoted by $u_i$, are the intersection points of the hyperplane at height $x_{n+1} = b$ with the generators of $P$ that are pointing “up” ($a_i \geq 0$). The $w_{i,j}$ are positive linear combinations of the generators $v_i$ that are pointing “up” and the generators $v_j$ that are pointing “down” ($a_j < 0$) such that $w_{i,j}$ has last coordinate equal to zero.

The generators of the tangent cone $K_{u_i}$ are $-u_i$, $u_i$, and $u_{i'} - u_i$ for $i' \in \{1, \ldots, l\}$, $i' \neq i$ and $w_{i,j}$ for $j \in \{l+1, \ldots, n\}$. These $n$ generators are linearly independent. Thus, $K_{u_i}$ is a simplicial cone. We denote by $U_i$ the matrix that has as columns the vectors $c_i u_i, \text{lcm}(c_i, c_{i'}) u_{i'} - u_i$ for $i' \in \{1, \ldots, l\} \setminus \{i\}$ and $w_{i,j}$ for $j \in \{l+1, \ldots, n\}$. $U_i$ is an $(n+1) \times n$ integer matrix that has a full-rank diagonal submatrix. The generating functions of the tangent cones $K_{u_i}$ are \(\frac{\lambda^b}{(1 - z_1\lambda^{a_1}) \cdots (1 - z_n\lambda^{a_n})} \) and \(\frac{\lambda^b}{(1 - z_1\lambda^{a_1}) \cdots (1 - z_n\lambda^{a_n})} \) for $i = 1, \ldots, l$, $U_i$. The generating function for $\pi(C \cap H_4)$, which is equal to $\Omega_{\lambda^b} \frac{\lambda^b}{(1 - z_1\lambda^{a_1})(1 - z_2\lambda^{a_2})(1 - z_3\lambda^{a_3})}$.

References


Testers

Nader H. Bshouty
Department of Computer Science
Technion, Haifa, Israel, 32000
bshouty@cs.technion.ac.il

Abstract

We develop a new notion called tester of a class of multivariate polynomials $M \subseteq \mathbb{F}[x_1, \ldots, x_n]$ that maps the elements $a \in \mathcal{A}$ of an $\mathbb{F}$-algebra $\mathcal{A}$ to a finite number (the size of the tester) of elements $b_1, \ldots, b_t$ in a smaller sub-algebra $\mathcal{B} \subset \mathcal{A}$ where the property $f(a) \neq 0$ is preserved for all $f \in M$. I.e., for all $f \in M$ and $a \in \mathcal{A}$ if $f(a) \neq 0$ then $f(b_i) \neq 0$ for some $i$.

We use tools from elementary algebra and algebraic function fields to construct testers of almost optimal size in deterministic polynomial time in the size of the tester. We then apply testers to deterministically construct new set of objects with some combinatorial and algebraic properties that can be used to derandomize some algorithms.

We show that those new constructions are almost optimal and for many of them meet the union bound of the problem. Constructions include, $d$-restriction problems, perfect hash, universal sets, cover-free families, separating hash functions, polynomial restriction problems, black box polynomial identity testing for polynomials and circuits over small fields and hitting sets.

1 Testers

One application of tester is the following: Suppose we need to construct a small set of vectors $S \subset \Sigma^n$ for some alphabet $\Sigma$ that satisfies some property $P$. We map $\Sigma$ into a field $\mathbb{F}$ and find a set of functions $M_P$ where $S \subset \mathbb{F}^n$ satisfies property $P$ if and only if $S$ is a hitting set for $M_P$, i.e., for every $f \in M_P$ there is $a \in S$ such that $f(a) \neq 0$. We then extend $\mathbb{F}$ to a larger field $\mathbb{K}$ (or $\mathbb{F}$-algebra $\mathcal{A}$). Find $S' \subset \mathbb{K}^n$ that is a hitting set for $M_P$ (which supposed to be easier). Then use tester to change the hitting set $S' \subset \mathbb{K}^n$ over $\mathbb{K}$ to a hitting set $S \subset \mathbb{F}^n$ over $\mathbb{F}$.

The main classes considered in this research are the class $\mathcal{P}(\mathbb{F}_q, n, d)$ of all multivariate polynomials with $n$ variables and total degree $d$ over the finite field $\mathbb{F}_q$, and the class $\mathcal{DML}(\mathbb{F}_q, n, d)$ of multilinear forms of degree $d$. We use elementary algebra and algebraic function fields to build in deterministic polynomial time tester for $\mathcal{P}(\mathbb{F}_q, n, d)$, $q \geq d + 1$, of size $O(d^c t)$ that maps elements from $\mathcal{A} = \mathbb{F}_q$ to $\mathcal{B} = \mathbb{F}_q$. Here $c \leq 6$ depends on the field size $q$ and $d$. For the class $\mathcal{DML}(\mathbb{F}_q, n, d)$, $q \leq d$, we build testers that map elements from $\mathcal{A} = \mathbb{F}_q$ to $\mathcal{B} = \mathbb{F}_q$ of size $2^{O((\log q/d)t)} \cdot t$. We also prove the lower bounds $\Omega(dt)$ and $2^{\Omega(1/q)d} \cdot t$, respectively, and give some classification of such testers. See more details in [1].
2 Applications

\textbf{d-Restriction Problems:} A \textit{d-restriction problem} is a problem of the following form: Given an alphabet \( \Sigma \) of size \(|\Sigma| = q \), a length \( n \) and a class \( \mathcal{M} \) of nonzero functions \( f_i : \Sigma^d \to \{0,1\}, \) \( i = 1, 2, \ldots, t \). Find a set \( A \subseteq \Sigma^n \) of small size such that: For any \( 1 \leq i_1 < i_2 < \cdots < i_d \leq n \) and \( f \in \mathcal{M} \) there is \( a \in A \) such that \( f(a_{i_1}, \ldots, a_{i_d}) \neq 0 \). Our new technique finds small sets for many \( d \)-restriction problems. We list the following four

1) \textbf{Perfect Hash:} For \( d \leq q \) we say that the set \( H \) of function \( h : [n] \to \mathbb{F}_q \) is a \((n, q, d)\)-perfect hashing (or \((n, d, q)\)-splitter) if for all subsets \( S \subseteq [n] \) of size \(|S| = d \) there is a hash function \( h \in H \) such that \( h|_S \) is injective (one-to-one) on \( S \), i.e., \(|h|_S(S)| = d|.

One of our testers gives a polynomial time construction of \((n, d^4, d)\)-perfect hashing of size \( O((d^2 / \log d) \log n) \). This improves all the results from the literature. See other results in [1].

2) \textbf{Universal sets:} \( A \subseteq \mathbb{F}_q^n \) is \((n, k)\)-universal set if for every \( 1 \leq i_1 < i_2 < \cdots < i_d \leq n \) and \( \alpha_1, \ldots, \alpha_d \in \mathbb{F}_q \) there is \( a \in A \) such that \( a_{i_1} = \alpha_1, a_{i_2} = \alpha_2, \ldots, a_{i_d} = \alpha_d \).

For \( q = \Omega(d^2) \) and perfect square, for infinite number of integers \( n \), we give a deterministic polynomial time construction of size \( O(d^2 (q^d / \log q) \log n) \) and an explicit construction of size \( O(d (q^d / \log q) \log n) \). This bound is a surprising result since it exceeds the union bound \( O(dq^d \log n) \) achieved by probabilistic method.

3) \textbf{Cover-Free Families (CFF):} Let \( X \) be a set with \( N \) elements and \( \mathcal{B} \) be a set of subsets (blocks) of \( X \). We say that \((X, \mathcal{B})\) is \((w, r)\)-cover-free family \(((w, r)\text{-CFF})\), if for any \( w \) blocks \( B_1, \ldots, B_w \in \mathcal{B} \) and any other \( r \) blocks \( A_1, \ldots, A_r \in \mathcal{B} \), we have \( \bigcap_{i=1}^w B_i \nsubseteq \bigcup_{j=1}^r A_j \). The goal is to find \((w, r)\text{-CFF}\) with small \( N \).

We use testers to give the first polynomial time almost optimal contraction of \((w, r)\text{-CFF}\) when \( w = o(r) \). Our results improve the results in the literature.

4) \textbf{Separating Hash Family:} Let \( X \) and \( \Sigma \) be sets of size \( n \) and \( q \), respectively. We call a set \( \mathcal{F} \) of functions \( f : X \to \Sigma \) an \((M; n, q, \{d_1, d_2, \ldots, d_r\})\) \textit{separating hash family} (SHF), if \(|\mathcal{F}| = M \) and for all pairwise disjoint subsets \( C_1, C_2, \ldots, C_r \subseteq X \) with \(|C_i| = d_i \) for \( i = 1, 2, \ldots, r \), there is at least one function \( f \in \mathcal{F} \) such that \( f(C_1), f(C_2), \ldots, f(C_r) \) are pairwise disjoint subsets.

We apply tester to get a deterministic polynomial time construction of such family. Our constructions improves the results in the literature.

\textbf{Black Box PIT} The second application of tester is getting asymptotically optimal black box PIT sets for classes of multivariate polynomials over finite fields with substitutions from an extension field of optimal dimension. For this problem we use tester in the following way. First we construct an optimal black box PIT set \( S \) for the class over a large field \( \mathbb{F}_{q^T} \). We then use a tester to map the assignments in \( S \) to assignments in a smaller field \( \mathbb{F}_q \). See the results in [1].

\textbf{Reduction of Black Box PIT Sets over Large Field to PIT Sets over Small Field:} We give several polynomial time reductions of black box PIT sets over large fields to black box PIT sets over small fields. See results in [1]. Other extensions and results will appear in [2, 3].

References


Closed form solutions of linear difference equations
in terms of symmetric products

Yongjae Cha *  
RISC-Linz  
Johannes Kepler University  
4040 Linz (Austria)  
ycha@risc.jku.at

Veronika Pillwein  
RISC-Linz  
Johannes Kepler University  
4040 Linz (Austria)  
vpillwei@risc.jku.at

In [2, 3, 4] an algorithm was presented that returns closed form solutions for second order linear difference operators with polynomial coefficients. This algorithm was implemented as a Maple package \textit{solver}. Closed form solutions are understood here as linear combinations of elementary and special functions over $\mathbb{C}$. Gegenbauer polynomials need to be computed. This invariant data (also called local data) are generalized exponents and valuation growth, the given $L$ of $(2 + 1)$ is known, then by means of this transformation a solution of $L_n(x)$ denotes the $n$th Laguerre polynomial. In this work we extend the previous algorithm to find closed form solutions of third order linear difference operators in terms of second order linear difference operators.

More precisely, we are looking for solutions that are (finite) sums of squares in order to apply the procedure for proving positivity of a given (third order) sequence. The problem setting here is: given a sequence only in terms of its defining recurrence relation, show that for all $n \in \mathbb{N}$ its elements are positive. An existing approach [5, 7] uses Cylindrical Algebraic Decomposition (CAD) and, if terminating, returns either “true” or “false”. Whereas a negative answer also provides a concrete counter-example, the answer “true” does not give any certificate or further insight.

Our procedure needs the same kind of input, but, if successful, returns a representation that allows to easily verify positivity by hand. Let $a_n$ be a given sequence satisfying a third order recurrence relation. Then we consider the following two cases: either find a closed form solution of the type $a_n = \sum_j c_j b_j^n$, or as a finite linear combination $a_n = c_0 b_0^n + c_1 b_1^n + c_2 b_2^n$. Then it remains to check positivity of the coefficients $c_k$ (e.g., by CAD).

In the following, let $\tau$ denote the shift operator acting on $n$ by sending $n \mapsto n + 1$ and let $\mathbb{C}(n)[\tau]$ be a ring of difference operators.

1 The existing algorithm

Next we briefly summarize the scope and main idea of \textit{solver}. Given a second order operator $L_1 \in \mathbb{C}(n)[\tau]$ the aim is to reduce it to an operator $L_2 \in \mathbb{C}(n)[\tau]$ for which the solution space is known. This reduction can be made by a bijective map, called GT-transformation, that sends solutions of $L_2$ to solutions of $L_1$. If a closed form solution of $L_2$ is known, then by means of this transformation a solution of $L_1$ can be constructed. In order to identify the given $L_1$ with an operator $L_2$ with known solutions, certain data that is invariant under GT-transformations needs to be computed. This invariant data (also called local data) are generalized exponents and valuation growth, for details see [2, 4]. Gegenbauer polynomials $C_n^{(m)}(x)$, for instance, are annihilated by the second order operator $(2 + n)\tau^2 - 2(1 + n + m)x\tau + (2m + n)$. The parameters of the corresponding base equation are $m$ and $x$, and the local data of this operator is $[[0, -1, 0], [2m, 0, 1], (x \pm \sqrt{x^2 - 1}(1 + (m - 1)T))].$

A table TB of base equations has been assembled that contains a collection of second order operators in parametrized form including the invariant data with known solutions. This table has been assembled using classical equations from the literature [1], or by generating them using existing algorithms.

Summarizing, for an input operator, \textit{solver} computes the invariant data and based on this data searches for possible candidates in TB and computes the corresponding parameter values. Finally, it checks whether there is a transformation and, if so, it computes the transformation. As a result, it returns a closed form solution of the input operator by applying the transformations to the known solutions from the database.

*Supported by the Austrian Science Fund (FWF) grant P22748-N18.
2 The extended algorithm

The extension solver\(^2\) covers third order difference operators, where the main steps of the algorithm are as in solver, but it aims at finding solutions in terms of symmetric squares that are defined as follows.

**Definition 1** Let \( K \in \mathbb{C}(n)[\tau] \) be a linear difference operator and \( u, v \) be solutions of \( K \). Then the symmetric square of \( K \), denoted \( K^{\circ2} \), is a minimal, w.r.t order, and monic operator in \( \mathbb{C}(n)[\tau] \) such that \( uv \) is a solution of \( K^{\circ2} \).

If \( u \) is a solution of \( K \) then \( u^2 \) is a solution of \( K^{\circ2} \) and the symmetric square of \( K \) can be computed from \( K \). The relation between third order operators and symmetric squares of second order operators is stated next.

**Lemma 2** If \( K = a_2(n)\tau^2 + a_1(n)\tau + a_0(n) \in \mathbb{C}(n)[\tau] \), \( a_0(n)a_1(n)a_2(n) \neq 0 \) then \( \text{ord}(K^{\circ2}) = 3 \).

For the extended algorithm a new table \( TB_2 \) has been built, whose entries are the symmetric squares of the operators in \( TB \). If a given third order linear difference operator \( L \) can be reduced to \( K^{\circ2} \) in \( TB_2 \), then it will have a closed form solution in terms of (sums of) squares. Either positivity follows trivially from this representation by positivity of the coefficients, or at least further insight in the problem was gained that allows to apply further methods like, e.g., [6] or classical approaches. Since the entries in the table cover all common sequences in general parametrized form, the procedure is fairly complete.

The algorithm for computing GT-transformation that was used in solver is not suitable for computing GT-transformations for higher order operators. This part was replaced by the new algorithm \( \text{Hom} \) developed by Yongjae Cha and Mark van Hoeij that will be described in a forthcoming paper.

3 An example

Turán type inequalities \( \Delta_n := a_{n+1} - a_n a_{n+2} \geq 0 \) hold for several families of orthogonal polynomials, e.g., for Hermite polynomials \( H_n(x) \) for all \( n \geq 0 \) and \( x \in \mathbb{R} \). The Turán determinant \( \Delta_n(x) \) for Hermite polynomials is annihilated by the third order operator \( L = \tau^3 + (2n + 2 - 4x^2)\tau - 4(n + 2)(n - 2x^2 + 4)\tau - 8(n + 1)(n + 2)^2 \). The local data for this operator is \([0, 0, 2], 2T^{-1}(1 + \frac{2}{3}T), -2T^{-1}(1 \pm \sqrt{2} \text i\sqrt{T^2 - (x^2 - \frac{1}{2})}) \) and solver\(^2\) determines the candidate \( H_n(x)^2 \) and computes a transformation that applied to \( H_n(x)^2 \) yields the following equivalent formulation

\[
H_{n+1}(x)^2 - H_n(x)H_{n+2}(x) = \frac{1}{2}H_{n+1}^2(x) + 2(n + 1 - x^2)H_n(x)^2 + 2n^2H_{n-1}^2(x)
\]

This representation gives the positivity of the Turán determinant for \( x \in [-\sqrt{n+1}, \sqrt{n+1}] \), \( n \geq 0 \).

**References**


On the structure of lexicographic Gröbner bases in dimension zero

Xavier Dahan
Kyūshū University, Department of Mathematics
Fukuoka, Japan, 819-0395
dahan@math.kyushu-u.ac.jp

Introduction. Because of the elimination property held by lexicographic monomial orders, the corresponding Gröbner bases possess strong structural properties from which meaningful informations can be easily extracted. In the case of ideals of dimension zero, a main application is the possibility to solve polynomial systems [2], but also for the primary decomposition [3]. It is also in this case that the structure is the most understood and has attracted a lot of research, starting with [3, 2]. The most achieved result is in [6], constraint to radical ideals, and its generalization to slightly more general ideals in [7].

The present work also focuses on the structure of lexicographic Gröbner bases in dimension zero. One outcome is the similar structural property obtained in [6]. However, it has some special features as detailed below:

1. the proof in [6, 7] is quite unwieldy, which is not commonplace considering that algorithmic facts are based on it. On the contrary, the new proof is simple. It generalizes the case of two unknowns treated in [1, Sec. 3].

2. Algorithmic consequences can be easily deduced. Supporting this fact, as a byproduct comes a first rigorous proof (of a version) of the "lextriangular" algorithm [4, Section 5].

3. The proof relies on interpolation formulas. On this also, a previous work [5] already considers such formulas but the viewpoint adopted here provides directly more explicit formulas.

Notations. Let $I$ be a radical zero-dimensional ideal of a polynomial ring $k[X_1, \ldots, X_n]$. We set the lexicographic monomial order $X_1 < \cdots < X_n$. Given a polynomial $p \in k[X_1, \ldots, X_n]$, we let $\text{lC}(p)$ and $\text{LM}(p)$ be its leading coefficient in $k$ and its leading monomial respectively. Moreover, for $1 \leq i \leq n-1$, we let $\text{lC}_i(p) \in R_i$ be the leading coefficient of $p$ seen as a polynomial in $R_i[X_{i+1}, \ldots, X_n]$ with $R_i = k[X_1, \ldots, X_i]$. We assume that $\text{char}(k) = 0$, or else that the degree $d(I) := \dim_k k[X_1, \ldots, X_n]/I$ of $I$ verifies: $d(I) < \text{char}(k)$. Let $V \subset \bar{k}^n$ be the zero-set of the polynomials in $I$, taken in the algebraic closure of $k$.

Theorem 1 (Structure) Let $G$ be a minimal lexicographic Gröbner basis of $I$, $g \in G \setminus k[X_1, \ldots, X_{n-1}]$ and $I_j := I \cap k[X_1, \ldots, X_j]$. The polynomial $g$ verifies:

$$g \in \langle \text{lC}_1(g) \rangle \iff \text{lC}_1(g) | g, \quad \forall 2 \leq t \leq n-1 \quad g \in \langle \text{lC}_t(g) \rangle + I_{t-1}.$$ 

A key feature of the present work is a well-suited decomposition of $V$, quite technical to define (skipped in the present abstract). To a multi-integer $i \in \mathbb{N}^n$, we associate "naturally" a subset $V^i \subset V$. Naturally means that the very definition of $V^i$ depends on the multi-integer $i$. Let $\mathcal{L}(V) := \{i : V^i \neq \emptyset\}$. The nature of the decomposition permits to consider an inductive reasoning on the cardinal $|\mathcal{L}(V)|$.

Definition 1 By using an interpolation formula, we can construct a polynomial $g_t \in k[X_1, \ldots, X_n]$ vanishing on $V$ such that $\text{LM}(g_t) = X_1^{i_1} \cdots X_n^{i_n}$. 

82
This uses crucially properties of (iterated) Lagrange interpolation polynomials, made possible by the feature of the decomposition of $V$. Let $G := \{ g_i : i \in L(V) \}$. Besides, an induction on the number of variables $n$ allows to consider a minimal Gröbner basis $G_{n-1}$ of $k[X_1, \ldots, X_{n-1}] \cap I$. From the above, we have $(G_{n-1} \cup G) \subseteq I$. As expected, it is a Gröbner basis of $I$.

**Theorem 2** We have $(\text{LM}(G_{n-1} \cup G)) = (\text{LM}(I))$. It follows that the set $G_{n-1} \cup G$ is a minimal Gröbner basis of $I$.

Note that it is not necessarily reduced. For this, extra work on the definition 1 must be done. As it stands, the polynomials have smaller coefficients though.

We prove that $(\text{LM}(I)) = (\text{LM}(G_{n-1} \cup G))$ by induction on the cardinal of $L(V)$. If this cardinal is larger than 1, we let $j$ be the smallest element in $L(V)$ and $V' := V \setminus V_j$. Let also $I' := I(V')$ and $J := I(V_j^3)$. The specificity of $V^3$ allows to deduce $\text{LM}(J)$ without effort. By induction $\text{LM}(I')$ is also known. From these two data, $\text{LM}(I)$ can be recovered by describing a natural isomorphism of $k$-vector spaces:

$$k[X_1, \ldots, X_n]/\text{LM}(I) \simeq k[X_1, \ldots, X_n]/\text{LM}(I') \times k[X_1, \ldots, X_n]/\text{LM}(J).$$

(1)

With this end, consider $g_j \in G$, and $f := \text{LC}_{n-1}(g_j)$. Then one has $I + \langle f \rangle = I'$ and $I : \langle f \rangle = J$, and hence, the isomorphism of $k$-algebras as well:

$$k[X_1, \ldots, X_n]/I \simeq k[X_1, \ldots, X_n]/I' \times k[X_1, \ldots, X_n]/J.$$

(2)

Therefore, the isomorphism of $k$-vector spaces (1) is given on the monomial basis by $m \mapsto (m \text{ quo } \text{LM}(f), m \text{ mod } \text{LM}(f))$. This concludes the proof by induction on $|L(V)|$.

**Lextriangular.** The decomposition (2) is at the heart of the aforementioned version of the “lextriangular” algorithm. This algorithm consists in computing a triangular decomposition of $I$ starting from a Gröbner basis. The inductive nature of the proof above is indeed reflected directly in the design of the algorithm. Moreover, its simplicity permits also to consider the reciprocal algorithm: reconstructing a Gröbner basis from this triangular decomposition (in the case of two unknowns, this is already treated in [1] Algo. 1 and Algo. 2).

**References**


[^1]: the output is a family of triangular sets, which are lexicographic Gröbner bases and also regular sequences
Estimating region of attraction for polynomial vector fields
by homotopy methods

Max Demenkov
Institute of Control Sciences, Russian Academy of Sciences
Profsoyuznaya str. 65, Moscow 117997, Russia
demenkov@ipu.ru, max.demenkov@gmail.com

We describe an initial stage of research into applicability of the homotopy methods from numerical algebraic geometry to the investigation of stability of the following polynomial vector fields:

\[ \dot{x}(t) = f(x(t)), \]

where \( x \in \mathbb{R}^n \) and \( t \) represents time. We assume that \( x_0 = 0 \) is the exponentially stable equilibrium. In this case, an important system characteristic is the size of its region of attraction (ROA), which is a set of initial points \( x(0) \) with the property \( x(t) \to 0 \) as \( t \to \infty \).

We can estimate the region of attraction through the sublevel sets of a Lyapunov function. A Lyapunov function for the equilibrium \( x_0 \) is a positive definite continuous function \( V(x,p) \) of \( x \) (parameterized by vector \( p \)), which is decreasing along the solution trajectories \( x(t) \) of the system:

\[ \dot{V}(x(t),p) < 0, x(t) \neq 0, \]

where \( \dot{V}(x,p) = \nabla_x V(x,p), f(x) \). An estimation of ROA is given by a compact set \( \Omega(p,\gamma) \) defined as follows [1]:

\[ \Omega(p,\gamma) = \{x: V(x,p) \leq \gamma\}, \Omega(p,\gamma) \subseteq \Psi(p), \Psi(p) = \{x: \dot{V}(x,p) \leq 0\}. \]

Many approaches to this problem exist. The proposed algorithm follows a particular method originated in [2, 3] and consists in two repeated stages. On the first stage, we use already known Lyapunov function to estimate ROA. On the second stage, we use the obtained information to change vector of parameters \( p \) so that to improve this estimation. After that, we fix \( p \) and repeat the second stage or go to the first stage according to some criteria. We stop if no improvement is achieved.

**Stage 1.** Note that for a fixed \( p \), \( \Omega(p,\gamma) \) of maximum size in terms of inclusion is defined by the following choice of \( \gamma \):

\[ \gamma = \min_{x \in H(x,p)} V(x,p), \quad H(x,p) = \{x: \dot{V}(x,p) = 0, x \neq 0\}. \]

It is possible to solve this optimization task by finding all solutions of a system of equations. Using first order conditions from optimization theory and supposing that \( V(x,p) \) is polynomial with \( p \) affecting its coefficients, we can write the following system:

\[ F(x,\lambda,p) = \left[ \begin{array}{c} \nabla_x V(x,p) - \lambda \nabla_x \dot{V}(x,p) \\ \dot{V}(x,p) \end{array} \right] = 0, \]

where \( \lambda \) is the Lagrange multiplier. If we have calculated all its isolated real solutions \((x,\lambda)\), they can be checked for additional conditions to determine those corresponding to the actual minimum and certifying that the resulting \( \Omega(p,\gamma) \) is inside the region of attraction. Note that previously it was proposed to use Gröbner basis for this purpose [4].
To obtain all isolated solutions, we perform the following artificial-parameter homotopy:

$$\alpha(1 - \beta)G(x, \lambda) + \beta F(x, \lambda, p) = 0, \; \beta \in [0, 1),$$

where $G(x, \lambda)$ is a start system obtained via mixed volume computations [5, 6], $\alpha$ is a random constant number and $\beta$ is the continuation parameter. Note also that varying $p$ does not change the structure of our polynomial equations. Therefore, we can calculate the start system only once and use it in all subsequent stages of the algorithm.

**Stage 2.** Suppose that we have picked a new point $p_{k+1}$ in the parameter space to improve ROA estimation, while the previous point was $p_k$. According to the implicit function theorem, we can recalculate $x$ and $\lambda$ locally using Newton-based continuation method applied to the following system of equations:

$$F(x, \lambda, (1 - \beta)p_k + \beta p_{k+1}) = 0, \; \beta \in [0, 1),$$

until we cross the bifurcation manifold defined as follows:

$$\det(J_F) = 0,$$

where $J_F$ is the Jacobian matrix of $F$ with respect to $x$ and $\lambda$. If no bifurcations occur along the path between $p_k$ and $p_{k+1}$, we repeat this stage. The homotopy on Stage 1 is performed only after bifurcation manifold crossing, as this could mean the appearance of solutions not detectable locally.

The novelty of our approach in comparison with [2, 3] is in applying continuation methods to perform numerical computations of the algorithm stages, while in the cited papers the proposed solution methods are based on general-purpose optimization or zero-finding techniques and the results are therefore not guaranteed. In our approach, with some assumptions we can guarantee, at least locally, estimation improvement. In comparison with approaches for polynomial systems based on semidefinite programming (see e.g. [9]), the method we follow can handle functions which are not representable as a sum of squares. The applicability of our approach is crucially dependent on the ability of recovering all existing real solutions and, therefore, on the robustness of the homotopy, which is the most complex part of our approach. This problem is currently an active research topic in numerical algebraic geometry (see e.g. [7, 8]).

**References**


Dual families in Enveloping algebras

Matthieu Deneufchâtel       Gérard H. E. Duchamp
Vincel Hoang Ngoc Minh
Laboratoire d’Informatique de Paris Nord, CNRS-UMR 7030, Université Paris 13
Villetaneuse, France, 93430
{matthieu.deneufchatel,ghed,minh}@lipn.univ-paris13.fr

The setting of Schützenberger’s factorisation

\[
\sum_{w \in X^*} w \otimes w = \prod_{\ell \in \text{Lyn}(X)} e^{S_\ell \otimes P_\ell} \tag{1}
\]

within the context of polyzetas leads us to consider several products on the free algebra \(k\langle X \rangle\), such as the shuffle product \(\shuffle\) and the stuffle product \(\shuf\), as well as different pairs of bases \((S_\ell, P_\ell)\) in duality for the scalar product \(\langle \cdot | \cdot \rangle\). Our goal is to derive the properties of a basis obtained by duality from the properties of the original basis.

Let \(k\) be a field of characteristic zero and \(X\) an alphabet. Then \(k\langle X \rangle\) denotes the free algebra and \(k\langle\langle X \rangle\rangle\) the set of all non-commutative series. These algebras are in duality for the scalar product \(\langle S_u | P_v \rangle = \delta_{uv}\) for all \(u, v \in X^*\).

A general theorem ([2]) which holds in every enveloping algebra, allows us to write Schützenberger’s factorisation (1) for these bases. Roughly speaking, for any Poincaré-Birkhoff-Witt basis \(i.e.\) computed thanks to Eq. (4)), the dual basis is multiplicative \(w.r.t. \shuffle\) (up to a coefficient Eq. (7)) so that Eq. (1) holds.

We are interested in the study of the reverse property (which does not necessarily hold):

\[
S'_w = \begin{cases} 
\ell & \text{if } \ell \in \text{Lyn}(X); \\
\frac{S'_{\ell_1} \shuffle \cdots \shuffle S'_{\ell_k}}{\alpha_1! \cdots \alpha_k!} & \text{if } w = \ell_{i_1}^{\alpha_1} \cdots \ell_{i_k}^{\alpha_k}. 
\end{cases} \tag{8}
\]
and in its properties w.r.t. Schützenberger’s factorisation. Therefore, we have to consider its dual family, which will be denoted by \((B’_w)_w \in X^*\) and which is defined by \(\langle S_{u,v} B’_v \rangle = \delta_{uv}\).

We obtain the following characterization of the elements of the family \((B’_w)_w\):

**Theorem 1** Let \(P\) belong to \(k\langle X\rangle\) and \(\ell \in \text{Lyn}(X)\). Then

\[
P = B’_\ell \iff \begin{cases} 
P \text{ is primitive } ; \\
|\text{supp}(P) \cap \text{Lyn}(X)| = 1; \\
\langle P|\ell \rangle = 1.
\end{cases}
\]  

(9)

**Theorem 2** Let \(\alpha \in \mathbb{N}^{(X)}\) be a multiax and \(\ell_1 < \cdots < \ell_m\) the Lyndon words that are multihomogeneous of multidegree \(\alpha\). Then the elements \(B’_{\ell_k}\) for those Lyndon words can be computed as follows:

\[
B’_{\ell_m} = P_{\ell_m}; \\
B’_{\ell_{m-k}} = P_{\ell_{m-k}} - \sum_{j=0}^{k-1} \langle P_{\ell_{m-j}} | \ell_{m-k} \rangle B’_{\ell_{m-j}}.
\]  

(10)

Moreover, a simple counter-example shows that the family \((B’_w)_w\) does not satisfy the Poincaré-Birkhoff-Witt property (i.e. is computed as in (4)) which is a necessary condition to write the factorisation.

We are also interested in the same constructions but in the case of the stuffle algebra: the stuffle product is defined, for an alphabet \(Y = \{y_i\}_{i \in \mathbb{N}}\), for all \(y_i, y_j \in Y\) and for all \(u, v \in Y^*\), by

\[
\begin{align*}
\omega \omega 1 &= 1 \omega u = u; \\
y_i u \omega y_j v &= y_i (u \omega y_j v) + y_j (y_i u \omega v) + y_i + j (u \omega v).
\end{align*}
\]  

(11)

More precisely, we would like to find the dual family of the basis defined by similar relations as we used for the family \(S_w\) but with the stuffle product instead of the shuffle product. We are still investigating the behaviour of some families in order to find two dual families which satisfy the properties allowing the statement of Schützenberger’s factorisation.

Note that these results are also interesting because they give some ideas about the general combinatorial configuration of a Lie algebra and its enveloping algebra. Indeed, the investigation of the case of the free algebra leads us to the statement of a criterion about the Poincaré-Birkhoff-Witt type of a basis which holds in any enveloping algebra.

**References**


Sparse Implicitization using Support Prediction

Ioannis Z. Emiris and Tatjana Kalinka
Dept. of Informatics and Telecommunications, University of Athens, 15784, Greece
emiris@di.uoa.gr, kalinkat@di.uoa.gr

Exploring the recently developed method for support prediction of sparse resultants [1] we revisit implicitization by interpolation. This method allows us to compute the Newton polytope of the resultant, or implicit polytope. Thus we can exploit the sparsity of the input and the output. Based on the computation of a superset of the support of the implicit equation, or implicit support, implicitization is reduced to computing the nullspace of a numeric matrix. The approach is applicable to polynomial and rational parameterizations of curves and (hyper)surfaces of any dimension, including the case of parameterizations with base points. Our techniques extend to the case of approximate computation, which is important in tackling larger problems.

The implicitization problem asks for the smallest algebraic variety containing the image of the parametric map $f : \mathbb{R}^n \to \mathbb{R}^{n+1} : t \mapsto f(t)$, $t \in \Omega$. $f := (f_0, \ldots, f_n)$. There have been numerous approaches, including resultants, Gröbner bases, and moving lines and surfaces as well as interpolation methods. Most of the latter fail to exploit the sparseness or structure of the input.

The image of $f$ is contained in the variety defined by the ideal of all polynomials $p$ in $\mathbb{R}[x_0, \ldots, x_n]$ s.t. $p(f_0(t), \ldots, f_n(t)) = 0$, $\forall t \in \Omega$. We restrict ourselves to the case when this is a principal ideal, and we wish to compute its defining polynomial $p(x_0, \ldots, x_n) = 0$, given its Newton polytope, or a polytope that contains it. If the parameterization is polynomial or rational: $x_i = f_i(t)/g_i(t), \ i = 0, \ldots, n$, then implicitization is reduced to eliminating $t$ from the polynomial system

$$F_i := x_i g_i(t) - f_i(t), \ i = 0, \ldots, n, \quad F_{n+1} := 1 - y g_0(t) \cdots g_n(t),$$

where $y$ is a new variable and $F_{n+1}$ assures that $g_i(t) \neq 0$. If one omits $F_{n+1}$, the generator of the corresponding (principal) ideal may be a multiple of the implicit equation. Our implicitization algorithm:

**Input:** Polynomial or rational parametrization $x_i = f_i(t_1, \ldots, t_n)$.

**Output:** Implicit polynomial $p(x_i)$ in the monomial basis in $\mathbb{N}^{n+1}$.

1. Support prediction yields polytope $Q \supseteq P$, where $P$ is the implicit polytope.
2. Compute all lattice points $S \subseteq \mathbb{N}^{n+1}$ in $Q$.
3. Repeat $\mu \geq |S|$ times: Select $\tau \in \mathbb{C}^n$, evaluate $x_i = f_i(\tau)/g_i(\tau)$, then evaluate monomials in $S$.
4. Construct the $\mu \times |S|$ matrix $M$, compute all vectors $\vec{p}_i$, s.t. $M \vec{p}_i = 0$.
5. Let $\text{corank}(M) = k \geq 1$, then compute $\gcd(\vec{p}_i^\top S, \ldots, \vec{p}_k^\top S)$; return its primitive part.

Let us describe the construction of $M$. Let $S := \{s_1, \ldots, s_{|S|}\}$; each $s_j = (s_{j0}, \ldots, s_{jn})$ is an exponent of a (potential) monomial $m_j := x_0^{s_{j0}} \cdots x_n^{s_{jn}}$ of the implicit polynomial, where $x_i = f_i(t)/g_i(t)$. We evaluate $m_j$ at some $\tau_k$, $k = 1, \ldots, \mu$, and obtain $m_j|_{t=\tau_k} := \prod_{i} f_i(\tau_k)/g_i(\tau_k)^{s_{ij}}$. Thus, we build an $\mu \times |S|$ matrix $M$ with rows indexed by $\tau_k$’s and columns indexed by $m_j$’s. For $\tau$, we have experimented with random integers in range $-\mu^2 \ldots \mu^2$, complex $\mu$-th roots of unity and random complex numbers modulo 1; the latter seem to offer the most stable results, from the numerical viewpoint. Random integers give matrices which are closer to having numerical corank 1, when the implicit support is not redundant.
If the predicted support corresponds to a multiple of the implicit equation, the matrix kernel yields more than one multiples of the implicit equation; this is quantified in [2]. Since $M$ is constructed using values $\tau$ that correspond to points on the parametric surface, we have:

**Proposition 1** Any polynomial in the basis of monomials indexing $M$, with coefficient vector in the kernel of $M$, is a multiple of the implicit polynomial $p(x)$. Under certain genericity assumptions on the evaluation of the entries of $M$, our algorithm yields $p(x)$.

When using numerical SVD, the method is about 10 times faster than exact linear algebra (commands NullSpace or LinearSolve on Maple) on significant inputs, as expected. However, the computed implicit equation is not a polynomial with rational coefficients. We can improve the accuracy by "filtering": disregarding all real or complex entries of the coefficient vector with norm close to zero.

Moreover, we need to evaluate how close the approximation is to the exact implicit equation. We employ two measures to quantify the accuracy of approximate implicitization: (a) Coefficient difference: measured as the norm of the difference of the two coefficient vectors $\vec{p}_{exact}, \vec{p}_{app}$, obtained from exact and approximate implicitization. (b) Evaluation norm: measured by considering the maximum norm of the approximate implicit equation when evaluated at a set of sampled points in $\Omega$. This is a lower bound on how far from zero can such a value be.

The complexity of the support prediction is given in [1, thm.10]. Finding all the lattice points in the polytope is NP-hard problem. Complexity of constructing a $\mu \times |S|$ matrix $M$ is $O^*(\mu |S| n)$ and kernel computation costs $O(m^2.376)$ arithmetic operations. The overall complexity of our implicitization algorithm is $O(\mu |S|^2)$.

We have implemented on Maple 13 [2] our algorithm and several experiments, including discriminant and resultant computations. The main functions are `imcurve`, incorporating the support-predicting algorithm for curves [3], and `imgen`.

Our experiments show that, while in the case of planar curves $\mu$-bases yield faster runtimes, our method is applicable to geometrical objects of higher dimensions. Compared with the Maple function `implicitize` our method proves to be faster and provides exact results in cases the former fails. Besides, most existing methods rely on exact computation over the rationals, whereas we apply approximate computation with satisfactory accuracy.

<table>
<thead>
<tr>
<th></th>
<th>Degree</th>
<th>LinearSolve</th>
<th>SVD</th>
<th>Maple <code>implicitize()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>runtime, s</td>
<td></td>
<td>runtime, s</td>
</tr>
<tr>
<td>Sparse Bézier curve</td>
<td>8</td>
<td>0.092</td>
<td>4.57</td>
<td>4.79 $\cdot 10^{-4}$</td>
</tr>
<tr>
<td>Dense Bézier curve</td>
<td>8</td>
<td>0.168</td>
<td>13.153</td>
<td>8.92 $\cdot 10^{-3}$</td>
</tr>
<tr>
<td>Plücker conoid surface</td>
<td>9</td>
<td>0.093</td>
<td>0.063</td>
<td>1.35 $\cdot 10^{-7}$</td>
</tr>
<tr>
<td>Plücker conoid surface</td>
<td>11</td>
<td>0.109</td>
<td>0.078</td>
<td>1.07 $\cdot 10^{-6}$</td>
</tr>
</tbody>
</table>

**References**


Zero-dimensional ideals of limited precision points

C. Fassino and M. Torrente
Dipartimento di Matematica
Università di Genova
Genova, Italy, I-16146
fassino@dima.unige.it, torrente@dima.unige.it

As part of the current movement of extending the classical concepts of Computational Commutative Algebra to the “empirical” case (see [7]), we present a new numerical approach to characterizing the vanishing ideal associated to a set of limited precision points. A similar concept has already been presented in recent literature: for instance in [1], [2], [4], [5], [6] it is defined in different ways as an ideal, not necessarily zero-dimensional, containing polynomials whose evaluations at the original points assume small values. We follow a different approach and define the numerical vanishing ideal associated to a set of limited precision points as a zero-dimensional complete intersection which is generated by polynomials of low degree and whose exact zero-set contains a small perturbation of the input points. More precisely, given a set $X$ of limited precision points of $\mathbb{R}^n$ and a tolerance $\varepsilon$ on the coordinates, we require that the polynomials of the numerical vanishing ideal satisfy the following three conditions: the ideal they generate is a zero-dimensional complete intersection, the minimum of their degrees is upper bounded by the minimum of the degrees of the elements of the exact vanishing ideal of $X$, and their zero-set contains an admissible perturbation of $X$, that is a set of points differing from $X$ by less than $\varepsilon$. A single polynomial satisfying the last two requirements can be computed using the Low-degree Polynomial Algorithm (LPA), an iterative symbolic-numerical method presented in [3].

Our idea is to compute the numerical vanishing ideal of $X$ w.r.t. $\varepsilon$ using an iterative method whose generic step is a generalization of the LPA. Our algorithm, like LPA, processes the power products of $x = (x_1, \ldots, x_n)$ w.r.t. a previously fixed term ordering $\sigma$, and it can be described as follows. The first iteration is just an application of the LPA to the pair $X$ and $\varepsilon$, so that a single polynomial is computed whose degree is upper bounded by the minimum of the degrees of the elements of the exact vanishing ideal of $X$. We consider the generic iteration of the algorithm. Let $f_1, \ldots, f_{k-1}$ be the polynomials computed after $k-1$ iterations and $\mathcal{O} = \bigcup_{i=1}^{k-1} \text{tail}(f_i)$; at the $k$-th iteration the smallest power product $t \succ_{\sigma} r$, with $r \in \mathcal{O}$, is considered and a new family of $k$ polynomials is computed such that the ideal they generate has dimension $n-k$ and its zero-locus passes close to the points of $X$ by less than $\varepsilon$. More precisely, the $k$-th iteration of the algorithm can be summarized as follows: we check whether there exists an admissible perturbation $X^*$ of $X$ such that the evaluation vector $\text{LT}_\sigma(f_i)(X^*)$ linearly depends on the vectors $\{r(X^*) \mid r \in \text{tail}(f_i)\}$ for each $i = 1 \ldots k-1$, and the evaluation vector $t(X^*)$ linearly depends on the vectors $\{r(X^*) \mid r \in \mathcal{O}\}$. If the check is verified we use the coefficients of the previous linear relations to construct the new family of polynomials $f_1^*, \ldots, f_k^*$; otherwise we add $t$ to $\mathcal{O}$ and consider the next power product $t$. After $n$ iterations the algorithm returns a zero-dimensional complete intersection whose zero-set contains a small perturbation of the original points. We illustrate our ideas in the following example.

**Example** Let $X \subset \mathbb{R}^2$ be the following set of points

$$X = \{(−0.988, −1.176), (1.153, −0.808), (1.383, −0.388), (−1.481, 0.253), (−1.193, 0.958), (0.884, 1.110)\}$$
The set $X$ has been created by perturbing by less than 0.1 the coordinates of the intersection points of the curves $f_1 = 0$ and $f_2 = 0$ where

$$f_1 = x^2 + y^2 - 2 \quad \quad f_2 = y^3 - y - \frac{1}{5}x$$

Using standard techniques of Computational Algebra we obtain that the minimal degree of the polynomials vanishing at $X$ is equal to 3; this degree is too high to point out that the set $X$ lies close to $f_1 = 0$. We apply our method to the set $X$ with a tolerance $\varepsilon = 0.1$. After 2 iterations we get the following output (the coefficients of the polynomials are displayed as truncated decimals):

$$g_1 = x^2 + 0.9370y^2 - 2.1157 + 10^{-1}(0.3798xy + 1.1964x + 0.3024y)$$

$$g_2 = y^3 - 1.0972y - 0.2183x + 10^{-1}(0.8200xy + 0.1298y^2 - 0.2731)$$

Note that, as required, $\min\{\deg(g_1), \deg(g_2)\} < 3$, the ideal $I$ generated by $g_1$ and $g_2$ is a zero-dimensional complete intersection, and each point of the zero-locus $\tilde{X}$ of $I$

$\tilde{X} = \{(-0.978, -1.163), (1.191, -0.809), (1.358, -0.384), (-1.495, 0.260), (-1.176, 0.972), (0.895, 1.100)\}$

differs from the corresponding point of $X$ by less than $0.04 < \varepsilon$. Finally, we observe that since for each $i = 1, 2$ the coefficients of $g_i$ differ only slightly from the corresponding coefficients of $f_i$, our method allows us to recover the curves $f_1 = 0$ and $f_2 = 0$ on which the points of $X$ “approximately” lie.

References


Fast Change of Ordering with Exponent \( \omega \)

Jean-Charles Faugère\(^1\), Pierrick Gaudry\(^2\), Louise Huot\(^1\) and GuénaëL Renault\(^1\)

1 : POLSYS Project INRIA Paris-Rocquencourt; UPMC, Univ Paris 06; CNRS, UMR 7606, LIP6
4 place Jussieu, Boîte courrier 169, 75252 Paris Cedex 05, France; firstname.lastname@lip6.fr
2 : CARAMEL Project INRIA Grand-Est; Université de Lorraine; CNRS, UMR 7503; LORIA
Campus Scientifique Boîte postale 239, 54506 Vandoeuvre-lès-Nancy, France; Pierrick.Gaudry@loria.fr

It is well known that Gröbner bases are a powerful tool to solve polynomial systems in particular in finite fields. In many applications, the resolution of the original problem is equivalent to solve polynomial systems. Hence, it is necessary to have an efficient strategy for polynomial system solving by using Gröbner basis and also efficient algorithms to compute Gröbner basis.

Let \( \mathbb{K} \) be a finite field and \( I \subset \mathbb{K}[x_1, \ldots, x_n] \) be the ideal generated by the system to solve with \( x_1 > \cdots > x_n \). The solutions of the system can be easily deduced of the lexicographical (denoted LEX) Gröbner basis of \( I \). Usually, to compute the LEX Gröbner basis of the ideal \( I \) one can proceed in two steps. First we compute \( G_{DRL} \), the degree reverse lexicographical (denoted DRL) Gröbner basis of \( I \) by using \( F_4 \) [2] or \( F_5 \) [3] algorithms of complexities bounded by \( O \left( \left( \frac{n+d_{\text{reg}}}{d_{\text{reg}}} \right) \omega \right) \) where \( \omega \) is the exponent in the complexity of multiplying two dense matrices with coefficients in \( \mathbb{K} \), \( n \) is the number of variables of the system and \( d_{\text{reg}} \) is the degree of regularity of \( I \) that is a bound on the maximal degree reached by polynomials during the computation of the DRL Gröbner basis (see [1]). Then we compute \( G_{\text{LEX}} \), the LEX Gröbner basis of \( I \) by using a change of ordering algorithm. The traditional change of ordering algorithm is FGLM [5] and has a complexity bounded by \( O(nD^3) \) where \( D \) is the degree of \( I \).

From a theoretical point of view, the second step is often the bottleneck of the polynomial systems solving. Hence, developing efficient algorithms of change of ordering is still an open issue. In general, the expected form of the LEX Gröbner basis of a 0-dimensional ideal is the shape position. That is, the LEX Gröbner basis of \( I \) has the form \( \{ h_n(x_n), x_{n-1} - h_{n-1}(x_n), \ldots, x_1 - h_1(x_n) \} \) where \( \deg(h_n) = D \) and \( \deg(h_i) < D \) for \( i = 1, \ldots, n - 1 \). In this work, we investigate the generic case of shape position ideals.

In 2011, Faugère and Mou [6] proposed a new change of ordering algorithm whose complexity for shape position ideals is bounded by \( O(D(\#T_n + n \log(D))) \) where \( T_n \) is the multiplication matrix describing the multiplication by \( x_n \) in \( \mathbb{K}[x_1, \ldots, x_n]/\langle G_{DRL} \rangle \) and \( \#T_n \) is the number of nonzero entries in \( T_n \). Although this algorithm requires no assumption about the structure of the matrix \( T_n \) in input, its complexity is better than the one of FGLM only when the matrix \( T_n \) is sparse.

In this poster, we propose a new change of ordering algorithm for ideals in shape position whose complexity does not depend of the structure of \( T_n \) and that is better than the one of FGLM. Our algorithm follows the one of Faugère and Mou. The idea is to replace some matrix-vector products by a smaller number of products of two matrices; by using an usual approach introduced by Keller-Gehrig in [8]. This leads to our first contribution:

**Theorem 1** Given \( T_n \) the multiplication matrix by \( x_n \) in \( \mathbb{K}[x_1, \ldots, x_n]/\langle G_{DRL} \rangle \), computing the LEX Gröbner basis of an ideal in shape position can be done in \( O(\log(D)(nD + D^2)) \).

Sometimes, the bottleneck of the change of ordering step is the construction of the matrix \( T_n \). Hence, we propose a new strategy of polynomial systems solving to make the cost of computing \( T_n \) negligible. Let \( B = \{ \epsilon_1, \ldots, \epsilon_D \} \) be the canonical basis of \( \mathbb{K}[x_1, \ldots, x_n]/\langle G_{DRL} \rangle \) seen as a \( D \)-dimensional vector space.
The $i$th column of the matrix $T_n$ is construct as the normal form of $\epsilon_i x_n$. From [5], the term $\epsilon_i x_n$ can be of three types:

I. $\epsilon_i x_n$ is in $B$, then $\epsilon_i x_n = \epsilon_j$ for some $j \in [i + 1, \ldots, D]$ and $\text{NF}(\epsilon_i x_n) = \epsilon_j$.

II. $\epsilon_i x_n = \text{LT}(g)$ for some $g \in G_{\text{DRL}}$ then $\text{NF}(\epsilon_i x_n) = \text{LT}(g) - g$.

III. otherwise, the normal form of $\epsilon_i x_n$ has to be computed.

The Moreno-Socías conjecture [9] is about the structure of the leading term ideal of $G_{\text{DRL}}$. In 2010, Pardue [10] showed that this conjecture implies that of Fröberg on the Hilbert series of a generic system which is widely admitted in the computer algebra community. In [7], it is shown that for generic systems, under the Moreno-Socías conjecture, all terms $\epsilon_i x_n$ are of type I or II. Hence, no arithmetic operations are required to construct the matrix $T_n$ and we can extend our theorem.

**Theorem 2** Let $\mathcal{I}$ be a generic ideal in shape position. Given the DRL Gröbner basis of $\mathcal{I}$, under the Moreno-Socías conjecture, its LEX Gröbner basis can be computed in $O(\log(D)(nD + D^\omega))$.

However, polynomial systems coming from applications are in general not generic. Hence, we propose another strategy for polynomial systems solving. First we compute the DRL Gröbner basis of $\mathcal{I}$ as usual. Then we try to compute the matrix $T_n$. If all terms $\epsilon_i x_n$ are of type I or II then we compute the LEX Gröbner basis by using a change of ordering algorithm. Else, if a term $\epsilon_i x_n$ is of type III we stop the computation of $T_n$ and we consider the ideal $\mathcal{I}^{(i)} \subset \mathbb{K}[x_1, \ldots, x_n, t]$, with $x_1 > \cdots > x_n > t$, generated by $G_{\text{DRL}}$ and the equation $t - \lambda_1 x_1 - \lambda_2 x_2 - \cdots - \lambda_n x_n$ where the $\lambda_i$’s are randomly chosen in $\mathbb{K}$. Finally, we apply the usual strategy to solve the polynomial systems on the ideal $\mathcal{I}^{(i)}$. We propose the heuristic that all terms required to compute the multiplication matrix by the variable $t$ are of type I or II. This heuristic has been checked on various examples (for instance systems coming from [4]) presented in the poster. As a consequence, we propose the following conjecture:

**Conjecture.** Let $\mathcal{I}$ be an ideal in shape position. Given the DRL Gröbner basis of $\mathcal{I}$, its LEX Gröbner basis can be computed in $O(\log(D)(nD + D^\omega))$.

As previously said, polynomial systems coming from applications are not generic. Consequently, in this case our work provides only heuristic complexity. Nevertheless, in practice we obtain measurable gains on the resolution of problems coming from applications. For instance, this new strategy allows to solve some instances, never solved before, of an elliptic curve cryptology problem [4].

**References**


Structured Computation in Optimization and Algorithmic Differentiation

Laurent Hascoët
INRIA Sophia-Antipolis, France  Laurent.Hascoet@inria.fr

Shahadat Hossain
Department of Mathematics and Computer Science
University of Lethbridge
Alberta, Canada  Shahadat.Hossain@uleth.ca

Trond Steihaug
Department of Informatics
University of Bergen, Norway  Trond.Steihaug@ii.uib.no

1 Introduction

It is frequently observed that effective exploitation of problem structure plays a significant role in computational procedures for solving large-scale nonlinear optimization problems. A necessary step in this regard is to express the computation in a manner that exposes the exploitable structure. The formulation of large-scale problems in many scientific applications naturally give rise to “structured” representation. Examples of computationally useful structures arising in large-scale problems include “unary functions” [1], “partially separable functions” [2], and “factorable functions” [5]. The main purpose of this note is to show that when the function is provided in a suitable form, these structures can be exploited automatically. We demonstrate that source transformation algorithmic differentiation [3] yields a result that is equivalent to classical derivative formulae on functions with the aforementioned structures, except for a few initializations and for some obvious dead code that are left for the compiler to simplify.

2 Factorable functions and algorithmic differentiation

A function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is a factorable function [5] if it can be represented as the last function in a finite sequence of functions $\{\phi_i\}_{i=1}^L$ where $\phi_i : \mathbb{R}^n \rightarrow \mathbb{R}$:

- $\phi_i(x) \equiv x_i, i = 1, \ldots, \ell$;
- $\phi_i(x) \equiv \phi_{j<i}(x) \circ \phi_{k<i}(x), i = \ell + 1, \ldots, L$, $\circ \in \{*, +, -, /, \hat{\ }\}$; or
- $\phi_i(x) \equiv \tau_i(\phi_{j<i}(x)), \tau_i : \mathbb{R} \rightarrow \mathbb{R}$;

$f(x) = \phi_L(x)$.

Here $\ell = n$, and the sequence $\{\phi_1(x), \ldots, \phi_i(x), \ldots, \phi_L(x)\}$ is called a “factored sequence”. The notation $\phi_{j<i}(x)$ means that there exists $j < i$ such that $\phi_j(x)$ is an element of the factored sequence defined above. The function $\tau_i$ is a “transformation function” such as exponential, trigonometric, logarithm etc. In general, $\phi_i(x) = u_i^T x$, $i = 1, \ldots, \ell$ and $u_i \in \mathbb{R}^n$ are given vectors; in the classical definition [5] $\ell = n$ and $u_i$ is a column of the identity matrix. Since each function $\phi_i$ in a factored sequence is explicitly available, we contend that a source transformation AD tool [4] will naturally yield the derivatives of function $f$ in the structure-preserving polyadic form [5]. To illustrate, we consider an example from [5]:

$$f(x) = a^T x \sin(b^T x)e^{c^T x},$$

(1)
as a factored sequence

\[ \begin{align*}
\phi_1(x) &= a^T x, \\
\phi_2(x) &= b^T x, \\
\phi_3(x) &= c^T x, \\
\phi_4(x) &= \sin(\phi_2(x)), \\
\phi_5(x) &= \exp(\phi_3(x)), \\
\phi_6(x) &= \phi_1(x) \ast \phi_4(x),
\end{align*} \]

implemented in Fortran 90. The source transformation AD tool Tapenade [4] yields the following function and the gradient evaluation code (in reverse AD).

```fortran
! Generated by Tapenade 3.6
! (r4227M) – 17 Nov 2011 08:35

subroutine factorable_b(x, xb, a, b, c, y, yb, n)
  real :: xb(n), yb
  real :: phi1b, phi2b, phi3b, phi4b, phi5b, phi6b
  phi1 = SUM(a(:) \ast x(:)) ; phi2 = SUM(b(:) \ast x(:)) ; phi3 = SUM(c(:) \ast x(:)) ;
  phi4 = SIN(phi2) ; phi5 = EXP(phi3) ; phi6 = phi1 \ast phi4 ;
  phi6b = phi5 \ast yb ; phi5b = phi6 \ast yb ; phi4b = phi1 \ast phi6b ; phi3b = EXP(phi3) \ast phi5b ; phi2b = COS(phi2) \ast phi4b
  xb = 0.0 ; xb(:) = b(:) \ast phi2b + a(:) \ast phi1b + c(:) \ast phi3b ; yb = 0.0
end subroutine factorable_b
```

The variable names appended with letter b are introduced by Tapenade and represent the local adjoints. It can be checked that the number of arithmetic operations in evaluating the hand-coded and the source-transformed gradient is approximately the same.

The number of operations for the derivatives is bounded by a factor of four times the number of operations of the original code, both for tangent and for adjoint, which is a classical result/belief in the AD community. It turns out that the classical formulae for separable or factorable functions resulted in programs with the same operations count. So, in our view, AD gives the same operations count as existing formulae on separable functions. In a forthcoming paper we provide a more detailed illustration of structure exploiting computation of gradients and Hessians of factorable functions using source transformation AD.

References

Finding $2F_1$ Type Solutions of Differential Equations with 5 Singularities

Mark van Hoeij and Vijay Jung Kunwar

Department of Mathematics, Florida State University

May 23, 2012

1 Introduction

Differential equations with $2F_1$ type solutions are very common in Mathematics and they occur quite frequently in Combinatorics and Physics. We are interested in solving differential equations with $n = 5$ non removable regular singularities. ($n = 3$ is easy, and $n = 4$ is done by M. van Hoeij and R. Vidunas (paper in progress)).

2 An Example

Consider the following differential equation:

$$-\frac{x(x-1)y}{(x+1)^2(3x-1)^2(3x+1)} + \frac{27x^3 + 19x^2 + x + 1}{2x(x+1)(3x-1)(3x+1)} \frac{dy}{dx} + \frac{d^2y}{dx^2} = 0.$$  \hspace{1cm} (1)

This equation has 5 regular singularities $\{-1, -\frac{1}{3}, 0, \frac{1}{3}, \infty\}$; among which the singularities $\{-1, \frac{1}{3}\}$ are logarithmic. Current Computer Algebra systems do not solve it. $y = 2F_1(\frac{1}{12}, \frac{1}{12}; \frac{2}{3} \mid f)$, with $f = \frac{(x-1)^3(3x+1)}{(5x-1)(x+1)^2}$ is a solution of (1).

Our goal is to build a complete table of all rational functions $f$ that can occur in this context, and then to develop a differential solver from it.

3 Problem Statement

If a second order differential equation $L_{inp}$ has:

(i) 5 non removable regular singularities.

(ii) At least one of the singularities is logarithmic.

Then we want to find its solution if it can be expressed in terms of $2F_1$ Hypergeometric function. More precisely, we want to find a solution of $L_{inp}$ in the form:

$$y = e^{\int r r_0 y_1 + r_1 y'_1}$$ \hspace{1cm} (2)

where $y_1 = 2F_1(a, b; c \mid f)$ and $f, r, r_0, r_1 \in \mathbb{C}(x)$.

4 Why logarithmic singularities ?

In the above example, the degree of $f$ was 4. For arbitrary $a, b$ and $c$ (without restriction (ii) above), the degree bound for $f$ is 60 when $n = 4$, and 96 when $n = 5$. For $n = 4$, there are 926 Belyi maps (up to Möbius equivalence) and a small number of near Belyi maps that can occur as $f$. 

96
For \( n = 5 \), we decided to restrict to differential equations \( L_{\text{inp}} \) that have at least one logarithmic singularity, for two reasons:

1. That lowers the degree bound for \( f \) from 96 to a more manageable 18.
2. Logarithmic singularities are very common in practice.

Among the differential equations with 5 non removable regular singularities, most of those which are \( _2F_1 \) solvable, arise from the Gauss Hypergeometric Equation (GHE) with exponent differences \((1-c, c-a-b, b-a) = (1/k, 1/2, 0)\) where \( k \in \{3, 4, 6\} \). We want to treat \((1/3, 1/2, 0)\) first, as that covers the majority of such cases. Denote the GHE with exponent differences \((1/3, 1/2, 0)\) at \((0, 1, \infty)\) as \( L_{320} \).

5 Our Results

Let the amount of ramification of \( f \) be \( R_{01\infty} \) (above \( \{0, 1, \infty\} \)) and \( R_{\text{out}} \) (above \( \mathbb{P}^1 \setminus \{0, 1, \infty\} \)). As in [1], using the Riemann Hurwitz formula, we can find the bound on the degree of \( f \) and \( R_{\text{out}} \). For \((1/3, 1/2, 0)\), we find:

\[
\text{deg}(f) \leq 18 \quad \text{and} \quad R_{\text{out}} \leq 2.
\]

We computed all rational functions (up to Möbius equivalence) that can occur as \( f \) in the solution (2) of \( L_{\text{inp}} \). For \((1/3, 1/2, 0)\), we computed a table with the following numbers of entries:

<table>
<thead>
<tr>
<th>( R_{\text{out}} )</th>
<th>Name</th>
<th>Degrees</th>
<th>Number of ( f )'s (up to Möbius equivalence)</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( Belyi )</td>
<td>3 – 16, 18</td>
<td>260</td>
<td>0 dimensional families</td>
</tr>
<tr>
<td>1</td>
<td>( Belyi_{-1} )</td>
<td>2 – 10, 12</td>
<td>68</td>
<td>1 dimensional families</td>
</tr>
<tr>
<td>2</td>
<td>( Belyi_{-2} )</td>
<td>4, 6</td>
<td>2</td>
<td>2 dimensional families</td>
</tr>
</tbody>
</table>

Our solver for \( L_{\text{inp}} \) will be complete if our table is complete. To prove the completeness, we do a combinatorial search to find all dessins and near dessins that are compatible with conditions (i) and (ii). If every dessin and near dessin in this search corresponds to a member of our table of Belyi maps and near Belyi maps, then the table is complete.

6 Main Algorithm

**Step 1**: Compute the singularity structure and a 5 point invariant (a function for sets of 5 points that is invariant under Möbius transformation) of \( L_{\text{inp}} \).

**Step 2.a**: Compare the 5 point invariant of \( L_{\text{inp}} \) with the ones in our table of Belyi maps. If they match, then compute Möbius transformation from singularities of the Belyi map to the singularities of \( L_{\text{inp}} \). The Belyi map composed with the Möbius transformation gives Candidate(s) for \( f \).

**Step 2.b**: For \( Belyi_{-1} \) maps \( f(x, t) \), compare 5 point invariants between singularity structures with matching exponent differences. That gives the value of \( t \) and thus, gives Candidate(s) for \( f \).

**Step 2.c**: For \( Belyi_{-2} \) maps, we have programs that compute Candidate(s) for \( f \) from the singularity structure of \( L_{\text{inp}} \).

**Step 3**: For each Candidate \( f \), we compute \( L_f \) (apply \( x \mapsto f \) on \( L_{320} \)) and finally, use [2] to compute \( r, r_0, r_1 \) in (2) if they exist.

References


Symbolic Analysis of Dynamic Systems

V.D. Irtegov, T.N. Titorenko
Institute for System Dynamics and Control Theory SB RAS
Irkutsk, Russia
e-irteg@icc.ru

1 Introduction

Some problems of qualitative analysis of differential equations can be reduced to algebraic ones that allows one to apply efficiently computer algebra methods for their solving. Examples of such problems are: finding equilibrium points, periodic solutions, stationary sets, etc., as well as investigation of stability, attraction and bifurcations for such types of solutions.

The paper presents an approach to investigation of dynamic systems, the motion equations of which possess first integrals. This approach is based on the Routh-Lyapunov method [1] and computer algebra methods. We demonstrate our technique by an example of symbolic analysis of dynamic systems which are described by Euler’s equations with polynomial first integrals. The differential equations of this problem write [2]:

\begin{align*}
\dot{s}_1 &= \alpha(r_1s_2 - \alpha r_2 r_3) - (\beta r_3 - s_2)(\beta r_2 + s_3), \\
\dot{s}_2 &= (\alpha^2 + \beta^2)r_1 r_3 - (\alpha r_1 + \beta r_2)s_1 + (\alpha r_3 - s_1)s_3, \\
\dot{s}_3 &= (\beta r_1 - \alpha r_2)s_3, \\
\dot{r}_1 &= r_2(\alpha r_1 + \beta r_2 + 2s_3) - r_3s_2 - x\left((\alpha^2 + \beta^2)r_3 s_2 + \beta s_3^2\right), \\
\dot{r}_2 &= r_3s_1 - r_1(\alpha r_1 + \beta r_2 + 2s_3) + x\left((\alpha^2 + \beta^2)r_3 s_1 + \alpha s_3^2\right), \\
\dot{r}_3 &= r_1s_2 - r_2s_1 + x(\beta s_1 - \alpha s_2)s_3.
\end{align*}

(1)

Here \(r_i, s_i\) are the phase variables, \(\alpha, \beta, x\) are some constants.

When \(x = 1\), equations (1) are interpreted as the Poincaré-Zhukovsky equations, which describe the motion of a rigid body having an ellipsoidal cavity filled with an ideal fluid, and when \(x = 0\), these equations are interpreted as Kirchhoff’s equations of motion of a rigid body in an ideal fluid. In the cases when \(x > 0\) and \(x < 0\), equations (1) are considered as Euler’s equations on Lie algebras \(so(4)\) and \(so(3, 1)\), respectively.

Equations (1) have the following first integrals:

\begin{align*}
2H &= s_1^2 + s_2^2 + 2(\alpha r_1 + \beta r_2)s_3 + 2s_3^2 - (\alpha^2 + \beta^2)r_3^2 = 2h, \\
V_1 &= r_1s_1 + r_2s_2 + r_3s_3 = c_1, \\
V_2 &= r_1^2 + r_2^2 + r_3^2 + x(s_1^2 + s_2^2 + s_3^2) = c_2, \\
V_3 &= x(\beta s_1 - \alpha s_2)^2s_3^2 + (r_1s_1 + r_2s_2)((\alpha^2 + \beta^2)(r_1s_1 + r_2s_2) + 2(\alpha s_1 + \beta s_2)s_3) + s_3^2(s_1^2 - s_2^2 + (\alpha r_1 + \beta r_2 + s_3)^2) = c_3.
\end{align*}

(2)

For the above equations, we state the problem of finding the stationary sets of non-zero dimension, which are called invariant manifolds (IM), and investigation of their stability and bifurcations.

The method of obtaining the IM of equations (1), which is used in this work, reduces the problem to solving a system of polynomial equations with parameters. In order to find the desired solutions we construct some linear combinations

\begin{align*}
2K = 2\lambda_0 H - 2\lambda_1 V_1 - \lambda_2 V_2 - \lambda_3 V_3 \quad (\lambda = \text{const}).
\end{align*}

(3)
(these combinations may also be nonlinear) from first integrals (2).

Next, we write down the conditions of stationarity for the integral $K$ (3) with respect to the phase variables $r_i, s_i$. As a result, we have:

$$\partial K/\partial s_1 = \lambda_0 s_1 - \lambda_1 r_1 - \lambda_3[(\alpha^2 + \beta^2)r_1(r_1s_1 + r_2s_2) + s_1s_3(\alpha r_2 + \beta r_1) + s_2s_3(2\alpha r_1 + s_3)] = 0,$$
$$\partial K/\partial s_2 = \lambda_0 s_2 - \lambda_1 r_2 - \lambda_3[(\alpha^2 + \beta^2)r_2(r_1s_1 + r_2s_2) + s_1s_3(\alpha r_2 + \beta r_1) + s_2s_3(2\beta r_2 + s_3)] = 0,$$
$$\partial K/\partial s_3 = \lambda_0(\alpha r_1 + \beta r_2 + 2s_3) - \lambda_1 r_3 - \lambda_3[(r_1s_1^2\alpha s_1 + \beta s_2)(s_1r_1 + s_2r_2) + s_3(\alpha r_1 + \beta r_2)^2$$
$$+ s_3(s_1^2 + s_2^2 + 2s_3^2) + 3s_3^2(\alpha r_1 + \beta r_2)] = 0,$$
$$\partial K/\partial r_1 = \lambda_0\alpha s_3 - \lambda_1 s_1 - \lambda_2 r_1 - \lambda_3[(\alpha^2 + \beta^2)s_1(r_1s_1 + r_2s_2) + \alpha s_3^2(\beta r_2 + \alpha r_1)$$
$$+ \alpha s_3(s_1^2 + s_2^2) + \beta s_3 s_1 s_2 s_3] = 0,$$
$$\partial K/\partial r_2 = -\lambda_0\beta s_3 + \lambda_1 s_2 + \lambda_2 r_2 + \lambda_3[(\alpha^2 + \beta^2)s_2(s_1r_1 + r_2s_2) + s_2s_3(\alpha s_1 + \beta s_2)$$
$$+ \beta s_3^2(a r_1 + \beta r_2) + \beta s_3^2] = 0,$$
$$\partial K/\partial r_3 = -(\alpha^2 + \beta^2)\lambda_0 + \lambda_2 r_3 + \lambda_1 s_3 = 0. \quad (4)$$

The solutions of the above equations allow us to obtain families of IM of equations (1) which correspond to the family of the first integrals $K$.

Equations (4) represent a system of polynomial equations of third degree with parameters $\lambda_i, \alpha, \beta$. For these equations, we tried to find both solutions existing without any restrictions imposed on the parameters and solutions which exist under some conditions imposed on the parameters. To this end, the Gröbner bases method has been applied. We have constructed a Gröbner basis of system (4) with respect to elimination ordering (the phase variables were used in capacity of variables). The latter has allowed us to decompose the system into several more simple subsystems, to conduct analysis of the solution set of each subsystems and to find the desired solutions, i.e. the families of one-dimensional sets of equations (1). We have also constructed a lexicographical Gröbner basis of equations (4). In this case, some part of parameters and some part of the phase variables have been used in capacity of variables. As a result, we have obtained not only IM, but and the first integrals of the differential equations, which are defined on these IM. The latter allows us to state the problem of finding and investigation of the stationary sets for the differential equations on the IM.

The parametric analysis for the solutions found has been conducted: the conditions of existence of real solutions in the form of restrictions imposed on the parameters $\lambda_i, \alpha, \beta, x$ have been obtained; a class of special IM has been selected from the found families of IM by constructing the envelope for the family of first integrals $K$ (3): stability in the Lyapunov sense and bifurcations have been investigated.

In order to solve the above problems, both Mathematica build-in tools and a special Mathematica software package have been employed. This package has been developed by the authors (in collaboration with their colleagues) to study the phase space of dynamic systems with a large number of symmetries (first integrals). The package allows one to investigate stability in the Lyapunov sense for the dynamic systems of above type.

The work was supported by the 17th Program Fundamental Researches of Presidium RAS.

References


Improved Polynomial Remainder Sequences for Ore Polynomials

Maximilian Jaroschek
Research Institute for Symbolic Computation (RISC)
Johannes Kepler Universität Linz
Altenbergerstraße 69
Linz, A-4040, Austria
maximilian.jaroschek@risc.jku.at

We generalize known improvements of the subresultant algorithm to speed up the computation of greatest common right divisors (GCRDs) of Ore polynomials. In applications of our implementation of the described method to medium-sized examples from combinatorics, we observed speed-ups of up to 2400% compared to GCRD computation by subresultants. We do not yet provide an analysis of the theoretical complexity.

Carrying out Euclid’s Algorithm applied to two polynomials \(a, b \in U[X]\), where \(U\) is a unique factorization domain, usually requires pseudo-division to bypass costly computations in the fraction field of \(U\). In order to avoid exponential coefficient growth during the process without much computational overhead, i.e. without computing the GCD of the coefficients of every remainder, different ways have been extensively studied to find factors of the content of the intermediate results, most notably reduced polynomial remainder sequences (PRSs) and subresultant PRSs, where each remainder in the sequence is a polynomial subresultant of \(a\) and \(b\) [2, 3, 6]. In [5], Li generalized these results to GCRD computations of Ore polynomials.

For commutative polynomials, further improvements are known. In [1], Brown provides formulas for dividing out known factors of the content of the polynomial subresultants of \(a\) and \(b\) and he shows that the GCD of the leading coefficients of \(a\) and \(b\) is such a factor. Another result can be found in [4]: If \(a\) and \(b\) are multiples of \(g \in U[X]\) and if any (and thus every) PRS of \(a\) and \(b\) is normal – i.e. the degrees in the remainder sequence decrease by 1 in every but possibly the last step – the content of the \(i\)-th remainder in the subresultant PRS is divisible by \(\text{lc}(g)^{2(i-1)}\).

We generalize these two results to Ore polynomials. Let \(K\) be a field, \(\sigma : K[X] \to K[X]\) an endomorphism and \(\delta : K[X] \to K[X]\) a pseudo-derivation with respect to \(\sigma\). Suppose \(a, b, m, g\) are elements of the Ore algebra \(K[X][D; \sigma, \delta]\) such that \(a, b\) are left multiples of \(g\) and \(m\) is a left multiple of \(g\) with minimal leading coefficient degree. Denote by \(d_a, d_b, d_m\) the degrees of \(a, b, m\) resp. The subresultant PRS of \(a\) and \(b\), \((r_i)_{i \in \{0, \ldots, \ell+1\}}\), can be obtained by pseudo-division:

\[
\beta_i r_{i+1} = \alpha_i r_{i-1} - q_i r_i, \quad q_i \in K(X)[D; \sigma, \delta], \quad \deg r_{i+1} < \deg r_i,
\]

with \(\alpha_i, \beta_i \in K[X]\) as described in [5]. The degree of the \(i\)-th remainder is denoted by \(d_i\). We show how to modify the \(\beta_i\) to compute the PRS \((\frac{1}{\gamma_i} r_i)_{i \in \{0, \ldots, \ell+1\}}\) in case factors \(\gamma_i \in K[X]\) of the content of \(r_i\) are known. To obtain such factors, we show that we can set

\[
\gamma_2 = \gcd(\text{lc}(a), \sigma^{d_a - d}(\text{lc}(b))), \quad \gamma_i = \sigma^{d_i-2-d_{i-1}}(\gamma_{i-1}), \quad 2 < i \leq \ell.
\]

∗supported by the Austrian FWF grant Y464-N18.
The generalization of the second improvement to the noncommutative case is less obvious due to the fact that even though we want $g$ to have polynomial coefficients, it usually is a GCRD in $K(X)[D;\sigma,\delta]$, i.e. we allow quotients with rational coefficients. (In general, Gauss’ Lemma does not hold in $K[X][D;\sigma,\delta]$.) It turns out that instead of $\text{lc}(g)$, the crucial quantity to be considered is the leading coefficient of $m$. If we assume that $\text{lc}(m)$ and $d_m$ are known and the subresultant PRS of $a$ and $b$ is normal, then we can set

$$
\gamma = \sigma^{d_b-d_m}(\text{lc}(m)),
$$

$$
\gamma_i = \prod_{j=0}^{2i-3} (\sigma^{2i+j}(\gamma)), \quad 2 \leq i \leq \ell.
$$

Of course, $\gamma$ is not known in general. A plausible guess, which in our applications was always correct, is

$$
\gamma = \gcd(\sigma^{d_b-d_a}(\text{lc}(a)), \text{lc}(b)).
$$

A typical application of GCRD computation of Ore polynomials is to find an operator of minimal order that annihilates a function for which some nonminimal annihilating operators can be computed efficiently by other means. Tests with operators in $\mathbb{Q}[X][S; s_x; 0]$ (with $s_x(X) = X + 1$) and $\mathbb{Q}[X][D; \text{id}_{\mathbb{Q}[X]}; \frac{d}{dx}]$ show that our implementation of the described improvements in Sage [7] give a significant speed-up compared to computations with subresultants or primitive remainders.

References


Accurate Computing Elementary Symmetric Functions

Hao Jiang\textsuperscript{a,b,c}, Stef Graillat\textsuperscript{c}, Roberto Barrio\textsuperscript{d}

\textsuperscript{a} School of Science, National University of Defense Technology, Changsha, 410073, China
\textsuperscript{b} The State Key Laboratory for High Performance Computation, NUDT, Changsha, 410073, China
\textsuperscript{c} PEQUAN, LIP6, Université Pierre et Marie Curie, CNRS, Paris, 75005, France
\textsuperscript{d} Dpto. de Matemática Aplicada and IUMA, Universidad de Zaragoza, E-50009 Zaragoza, Spain

Hao.Jiang@lip6.fr, Stef.Graillat@lip6.fr, rbarrio@unizar.es

This work concerns with the numerical computation of the $k$-th elementary symmetric function (ESF) with floating-point inputs $X = (x_1, \ldots, x_n)$, which is defined as
\[ S_k^{(n)}(X) = \sum_{1 \leq \pi_1 < \cdots < \pi_k \leq n} x_{\pi_1} x_{\pi_2} \cdots x_{\pi_k}, \quad 1 \leq k \leq n. \tag{1} \]

We focus mainly on the case $2 \leq k \leq n - 1$. For $k = 1$, the problem simplifies to the computation of the sum of floating-point numbers, and for $k = n$, to the computation of floating-point product. The classic and widely-used method is the so-called Summation Algorithm, denoted by $\text{SumESF}$, which is essentially the algorithm used by MATLAB’s $\text{poly}$. The error analysis has been considered in \cite{1}, and the result implies that the algorithm is forward stable. We present the relative forward error bound as follows,
\[ \left| \frac{\text{SumESF}(X,k) - S_k^{(n)}(X)}{S_k^{(n)}(X)} \right| \leq \frac{1}{k} \gamma_2(n-1) \text{cond}(S_k^{(n)}(X)), \quad \text{with } \text{cond}(S_k^{(n)}(X)) = \frac{kS_k^{(n)}(|X|)}{|S_k^{(n)}(X)|}, \tag{2} \]

where $\gamma_n = nu/(1-nu)$ with $u$ be the rounding error unit (in double precision $u = 2^{-53}$) and absolute value is to be understood componentwise. However, when performed in floating-point arithmetic, the computed result by $\text{SumESF}$ may still be less accurate than expected due to cancelations. This is why a more accurate algorithm is required.

By introducing error-free transformation (EFT) to the traditional Summation Algorithm, we propose a fast and accurate compensated algorithm, which is denoted by $\text{CompSumESF}$ and presented in Figure 1. For a pair of floating-point numbers $a, b \in \mathbb{F}$, when no underflow occurs, there exists a floating-point number $y$ satisfying $a \circ b = x + y$ with $\circ \in \{+,-,\times\}$, where $x = \text{fl}(a \circ b)$ is the usual floating-point approximation and $y$ represents the exact rounding error. The transformation $(a, b) \rightarrow (x, y)$ is regarded as an EFT. The EFT algorithms for the addition and product of two floating-point numbers used in $\text{CompSumESF}$ are $\text{TwoSum}$ and $\text{TwoProd}$ algorithms, respectively. One can see the details about their properties in \cite{2}. Then, the forward error bound of our method is
\[ \left| \frac{\text{CompSumESF}(X,k) - S_k^{(n)}(X)}{S_k^{(n)}(X)} \right| \leq u + \frac{1}{k} \gamma_2(n-1) \text{cond}(S_k^{(n)}(X)), \tag{3} \]

It is interesting to compare our method with the approach using Bailey’s double-double arithmetic denoted by $\text{DDSumESF}$. All the results about accuracy measurements are reported on Figure 2, which imply that the result computed by our method is as accurate as if computed in twice the working precision. When the problem is not too ill-conditioned it yields nearly full accuracy. We perform numerical tests
About measured running time, using compiler VC+++9.0, on a laptop with an Intel(R) Core(TM) i5-2520M processor, with two cores each at 2.50Ghz. The results show that CompSumESF is as accurate as DDDSumESF but only requires on the average 57% of its measured running time. Moreover, our method only requires addition and multiplication of floating-point numbers in the same working precision as the given data. As a consequence, it seems that our method is a simple, fast and accurate algorithm to compute elementary symmetric functions.

As an application, the ESFs appear when expanding a linear factorization of a polynomial

$$\prod_{i=1}^{n} (x - x_i) = \sum_{i=0}^{n} c_i x^i = \sum_{i=0}^{n} (-1)^{n-i} s_i^{(n)}(X) x^i. \quad (4)$$

It is an option to use our method to accurately evaluate polynomial’s coefficients from zeros, specially to compute characteristic polynomials from eigenvalues. The computation of ESFs is also an important part of conditional maximum likelihood estimation of item parameters under the Rasch model in psychological measurement [3]. It is promising that our method, improving the numerical accuracy, can allow much more items to be calibrated.

*This work was supported by the Foundation for Innovative Research Groups of the National Natural Science Foundation of China (Grant No.60626003), by the Spanish Research project MTM2009-10767 and by CSC 2011611057.

References


Solving Non-Linear Arithmetic

Dejan Jovanović
New York University
New York, USA, 10012
dejan@nyu.edu

Leonardo de Moura
Microsoft Research
Redmond, USA, 98074
leonardo@microsoft.com

Abstract

We propose a new decision procedure for the existential theory of the reals. It performs a backtracking search for a model in \( \mathbb{R} \), where the backtracking is powered by a novel conflict resolution procedure based on cylindrical algebraic decomposition. The initial experimental results are very encouraging. The full article has been accepted at the 6th International Joint Conference on Automated Reasoning (IJCAR 2012).

Summary. Solving polynomial constraints is one of the classic problems in computer algebra. In 1951, Tarski [8] showed that the theory of real closed fields admits elimination of quantifiers, and hence that a general decision procedure for solving polynomial constraints was indeed possible. Tarski’s original procedure was unfortunately totally impractical but, as one would expect, it has consequently been much improved. Most notably, Collins [2] gave the first relatively effective method of quantifier elimination by cylindrical algebraic decomposition (CAD). The CAD procedure itself has gone through many revisions and improvements [5, 3, 1]. However, even with the improvements and various heuristics, its doubly-exponential worst-case behavior has remained as a serious impediment.

The CAD algorithm works by decomposing \( \mathbb{R}^k \) into connected components such that, in each cell, all of the polynomials from the problem are sign-invariant. To be able to perform such a particular decomposition, CAD first performs a projection of the polynomials from the initial problem. This projection includes many new polynomials, derived from the initial ones, and these polynomials carry enough information to ensure that the decomposition is indeed possible. Unfortunately, the size of these projection sets grows exponentially in the number of variables, causing the projection phase, and its consequent impact on the search space, to be a key hurdle to CAD scalability.

We propose a new decision procedure for the existential theory of the reals that tries to alleviate the above problem. The new procedure performs a backtracking search for a model in \( \mathbb{R} \), where the backtracking is powered by a novel conflict resolution procedure. Our approach takes advantage of the fact that each conflict encountered during the search is based on the current assignment and generally involves only a few constraints, a conflicting core. When in conflict, we project only the polynomials from the conflicting core and explain the conflict in terms of the current model. This means that we use projection conservatively, only for the subsets of polynomials that are involved in the conflict, and even then we reduce it further. As another advantage, the conflict resolution provides the usual benefits of a Conflict-Driven Clause Learning style [7] search engine, such as non-chronological backtracking and the ability to ignore irrelevant parts of the search space.

The website http://cs.nyu.edu/~dejan/nonlinear/ contains a technical report, our prototype nlsat, and experimental results. The technical report contains a detailed description of our procedure, examples, additional references, and implementation details.
Table 1: Experimental results.

<table>
<thead>
<tr>
<th>solver</th>
<th>solver</th>
<th>time (s)</th>
<th>solved</th>
<th>time (s)</th>
<th>solved</th>
<th>time (s)</th>
<th>solved</th>
<th>time (s)</th>
<th>solved</th>
<th>time (s)</th>
<th>solved</th>
<th>time (s)</th>
<th>solved</th>
<th>time (s)</th>
<th>solved</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlsat</td>
<td>1002</td>
<td>343.48</td>
<td>420</td>
<td>5.73</td>
<td>89</td>
<td>234.57</td>
<td>10</td>
<td>170.33</td>
<td>13</td>
<td>95.62</td>
<td>1534</td>
<td>849.73</td>
<td>10</td>
<td>49.01</td>
<td>1491</td>
</tr>
<tr>
<td>Mathematica</td>
<td>1006</td>
<td>796.90</td>
<td>420</td>
<td>17.19</td>
<td>50</td>
<td>366.10</td>
<td>9</td>
<td>208.04</td>
<td>6</td>
<td>29.01</td>
<td>1491</td>
<td>1572.01</td>
<td>13</td>
<td>275.44</td>
<td>1324</td>
</tr>
<tr>
<td>QEPCAD</td>
<td>991</td>
<td>2616.94</td>
<td>368</td>
<td>1331.67</td>
<td>21</td>
<td>38.85</td>
<td>6</td>
<td>43.56</td>
<td>4</td>
<td>5.80</td>
<td>1390</td>
<td>4036.82</td>
<td>10</td>
<td>2948.13</td>
<td>1324</td>
</tr>
<tr>
<td>Redlog-VTS</td>
<td>847</td>
<td>2864.02</td>
<td>419</td>
<td>78.58</td>
<td>42</td>
<td>490.54</td>
<td>6</td>
<td>3.31</td>
<td>10</td>
<td>275.44</td>
<td>1324</td>
<td>2948.13</td>
<td>13</td>
<td>275.44</td>
<td>1324</td>
</tr>
<tr>
<td>Redlog-CAD</td>
<td>848</td>
<td>21706.75</td>
<td>363</td>
<td>730.25</td>
<td>21</td>
<td>173.15</td>
<td>6</td>
<td>2.53</td>
<td>4</td>
<td>0.81</td>
<td>1242</td>
<td>22613.49</td>
<td>13</td>
<td>275.44</td>
<td>1324</td>
</tr>
<tr>
<td>iSAT</td>
<td>203</td>
<td>122.93</td>
<td>291</td>
<td>16.95</td>
<td>21</td>
<td>24.52</td>
<td>20</td>
<td>822.01</td>
<td>0</td>
<td>0.00</td>
<td>535</td>
<td>986.41</td>
<td>13</td>
<td>275.44</td>
<td>1324</td>
</tr>
</tbody>
</table>

**Experimental Results.** In order to evaluate the new decision procedure we have implemented a new solver nlsat, the implementation being a clean translation of the decision procedure described in our technical report. We compare the new solver to the following solvers: Mathematica 8.0, QEPCAD 1.65, Redlog-CAD and Redlog-VTS; and the interval based iSAT. The nlsat main procedure and the polynomial and real algebraic number libraries were implemented from scratch in C++. We have reused parsers, simplifiers and basic data-structures from the z3 theorem prover [4]. The experimental results are summarized in Table 1.

The meti-tarski benchmarks are proof obligations extracted from the MetiTarski project, where the constraints are of high degree and the polynomials represent approximations of the elementary real functions being analyzed. The keymaera benchmark set contains verification conditions for hybrid systems. The zankl set of problems originating from attempts to prove termination of term-rewrite systems. We also have two crafted sets of benchmarks, the hong benchmarks, which are a parametrized generalization of the problem from [6], and the kissing problems that describe some classic kissing number problems, both sets containing instances of increasing dimensions.

**References**


Gröbner walk for computing matrix normal forms over Ore polynomials

George Labahn
Symbolic Computation Group
School of Computer Science
University of Waterloo, Waterloo
Ontario, Canada, N2L 3G1
glabahn@uwaterloo.ca

Johannes Middeke
Research Institute for Symbolic Computation (RISC)
Johannes Kepler Universität Linz
Altenbergerstraße 69
Linz, A-4040, Austria
johannes.middeke@risc.jku.at

In this poster we explore the links between the Hermite normal form and the Popov normal form of matrices over univariate Ore polynomials on the one side and the theory of Gröbner bases for modules on the other side. We focus on how the Gröbner walk algorithm which converts Gröbner bases into Gröbner bases for a different term order can be applied for converting matrices in Hermite normal form to Popov normal form. Our purpose is to see if existing methods can be described by Gröbner walks and, conversely, to see what type of algorithm is described in terms of matrix normal forms using a Gröbner walk.

Matrix normal forms play an important role in the analysis and solving of systems of linear equations. We are interested in systems of linear operators which can be modelled by Ore polynomials. These are a non-commutative generalisation of the usual polynomials with which they share many properties including a Gröbner basis theory. Examples of Ore polynomials are differential operators, difference operators and also the commutative polynomials themselves.

We will consider two popular normal forms w.r.t. elementary row operations. As an upper triangular form, the Hermite normal form is particularly well suited for solving linear systems—see, e.g., [4]. The Popov normal form on the other hand has the advantage of having rows of minimal degree—see, e.g., [3]. This makes it well suited if a description of the row space or its factor module is needed.

Both, the Hermite normal form and the Popov normal form are Gröbner bases for their row space in the sense of Gröbner bases for modules. This has first been noted for the commutative case in [5] and recently generalised to Ore polynomials in [6]. More precisely, the term order needed to obtain a Hermite normal form is the position over term order while a Popov normal form can be computed using the term over position order.

In general, the Popov normal form is easier to compute than the Hermite normal form. Thus, in order to compute a Hermite normal form for a given matrix, one could possibly attempt to first convert it into Popov normal form and try to compute the Hermite normal form from there. In the case of commutative polynomial ideals there are two famous algorithms which solve exactly this latter task of converting a Gröbner basis from one monomial ordering to another. These are the Gröbner walk and the FGLM algorithm.

It has already been proved in [6] that the FGLM algorithm can be converted to the case of matrix normal forms over Ore polynomials and allows an efficient conversion of Popov normal forms into Hermite normal forms. In the present work we investigate the translation of the Gröbner walk. We are basing our method on [1]. This source describes a Gröbner walk algorithm for modules of commutative polynomials in several variables which we translate to Ore polynomials in a single variable. Since this reduces the number of different monomial orderings, the Gröbner fans become more simple than in the original algorithm. We investigate the similarities and differences of this approach for normal form conversion with existing methods like that in [7] for the case of commutative polynomials. As another example for the connection
of Gröbner walk with existing normal form theory, in [2], so-called *shifted Popov normal forms* are defined. It is easy to prove that they are Gröbner bases as well. We have found that they correspond to certain intermediate Gröbner bases which are found during a walk from a Popov normal form to a Hermite normal form.

**References**


On the complexity of computing certain resultants

Romain Lebreton  
LIX, École polytechnique, France  
lebreton@lix.polytechnique.fr

Esmaeil Mehrabi and Éric Schost  
CS Department, Western University, Canada  
mehrabi@uwo.ca, eschost@uwo.ca

Computing resultants is a fundamental algorithmic question, at the heart of higher-level algorithms for solving systems of equations, computational topology, etc. However, in many situations, the best known algorithms are still sub-optimal. The following table summarizes the best results known to us (from [3]), using soft-Oh notation to omit logarithmic factors. In all cases, we assume that \( f, g \) have coefficients in a field \( k \), and that their partial degrees in all variables is at most \( d \). The partial degree in all remaining variables of their resultant \( r = \text{res}(f, g, x_1) \) is then at most \( 2d^2 \). In this note, the cost of an algorithm is the number of arithmetic operations in \( k \) it performs.

<table>
<thead>
<tr>
<th>( f, g ) are in . . .</th>
<th>( k[x_1] )</th>
<th>( k[t, x_1] )</th>
<th>( k[t, x_0, x_1] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of terms in ( f, g )</td>
<td>( \Theta(d) )</td>
<td>( \Theta(d^2) )</td>
<td>( \Theta(d^3) )</td>
</tr>
<tr>
<td>number of terms in ( r = \text{res}(f, g, x_1) )</td>
<td>1</td>
<td>( \Theta(d^2) )</td>
<td>( \Theta(d^3) )</td>
</tr>
<tr>
<td>cost of computing ( r ) (best known bound)</td>
<td>( O^*(d) )</td>
<td>( O^*(d^3) )</td>
<td>( O^*(d^3) )</td>
</tr>
<tr>
<td>optimal?</td>
<td>yes, up to log factors</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

In the last cases, one can replace the ring \( k[t] \) by \( \mathbb{Z} \) and consider the bit-complexity of computing resultants of polynomials in \( \mathbb{Z}[x_1] \) or \( \mathbb{Z}[x_0, x_1] \). We do not give details, but most results carry over, mutatis mutandis. One can also consider polynomials in \( \mathbb{Z}[t, x_0, x_1] \), etc; we do not discuss this.

**Main result.** Our contribution is on the third case, with \( f, g \) in \( k[t, x_0, x_1] \). We make the following assumptions (which hold for generic \( f, g \)):

**A**\(_1\). The reduced Groebner basis of the ideal \( \langle f, g \rangle \) in \( k(t)[x_0, x_1] \) for the lexicographic order \( x_1 > x_0 \) has the form \( \langle R(x_0), x_1 - S(x_0) \rangle \), with \( R, S \) in \( k(t)[x_0] \) and \( R \) monic.

**A**\(_2\). All solutions of the system \( f = g = 0 \) in \( k(t) \) have multiplicity 1.

Our algorithm uses matrix multiplication; we let \( 2 < \omega < 3 \) be such that one can multiply \( n \times n \) matrices over \( k \) in \( n^\omega \) operations. The best known result [5] is \( \omega \simeq 2.37 \).

**Theorem 1.** Let \( f, g \) be in \( k[t, x_0, x_1] \), with degree at most \( d \) in all variables and that satisfy **A**\(_1\), **A**\(_2\). Suppose that \( k \) has cardinality at least \( 12d^4 \). There exists a probabilistic algorithm that computes \( r = \text{res}(f, g, x_1) \) using \( O(d^{\frac{5\omega}{\omega-3}}) \) operations in \( k \) and success probability at least 1/2.

Since we have \( 2 < \omega \leq 3 \), the exponent \( \rho \) in our running time satisfies \( 4.5 < \rho \leq 5 \). This improves on the best previous results, getting us closer to an optimal \( O^*(d^4) \). Even under assumptions **A**\(_1\), **A**\(_2\), and allowing probabilistic algorithms, we are not aware of any previous improvements over \( O^*(d^4) \).

**Sketch of our algorithm.** The polynomial \( R \) introduced in **A**\(_1\) and the resultant \( r \) of \( f \) and \( g \) are related by the equality \( R = r/\text{LeadingCoefficient}(r, x_0) \). We describe how to compute \( R \), since finding the proportionality factor that gives \( r \) is straightforward.

The algorithm uses Newton / Hensel lifting techniques. We choose a random expansion point \( \tau \) for \( t \) in \( k \). This is the source of the probabilistic aspect of the algorithm: we expect that no denominator in \( R \) or \( S \) vanishes at \( \tau \), and that the solutions of the system \( f(\tau, x_0, x_1) = g(\tau, x_0, x_1) = 0 \) in \( \mathbb{F} \) still have multiplicity 1; the analysis in [4, Prop. 3] and our assumption on the cardinality of \( k \) show that at least half the points...
in $k$ are “lucky” for this random choice. Below, we take $\tau = 0$ for simplicity; then, by assumption, for $\kappa \geq 1$, $R_\kappa = R \mod t^\kappa$ and $S_\kappa = S \mod t^\kappa$ are well-defined; they lie in $A_\kappa[x_0]$, with $A_\kappa = k[t]/t^\kappa$.

We first compute $R_1 = R \mod t$ and $S_1 = S \mod t$, using Reischert’s algorithm in $k[x_0, x_1]$; this costs $O^*(d^3)$. Then, we compute $R_\kappa$ and $S_\kappa$ for some $\kappa \geq 4d^2$ using lifting techniques: the successive lifting steps compute $(R_2, S_2), (R_1, S_1), \ldots, (R_4, S_4), \ldots$. The assumption that the system $f(\tau, x_0, x_1) = g(\tau, x_0, x_1) = 0$ has simple roots makes this step well-defined; we analyze its cost below. Finally, we get $R$ by applying rational reconstruction to all coefficients of $R_\kappa$ in time $O^*(d^4)$.

The key subroutine. The above process is hardly new: the references [2, 4] give details on such lifting algorithms, in more general contexts; however, as explained now, a direct application of these results performs poorly in our context.

Given $(R_\kappa, S_\kappa)$, the algorithm of [2, 4] computes $(R_{2\kappa}, S_{2\kappa})$ as follows. Let $B_\kappa = A_{2\kappa}[x_0, x_1]\langle R_\kappa, x_1 - S_\kappa \rangle = k[t, x_0, x_1]/(t^{2\kappa}, R_\kappa, x_1 - S_\kappa)$. First, compute the normal form of $(f, g)$, and of their Jacobian matrix $J$, in $B_\kappa$; then, deduce the vector

$$\begin{bmatrix} \delta_R \\ \delta_S \end{bmatrix} = \begin{bmatrix} R'_\kappa \\ -S'_\kappa \end{bmatrix} J^{-1} \begin{bmatrix} f \\ g \end{bmatrix} \in B_k^{2 \times 1}.$$ 

Taking canonical preimages of $\delta_R$ and $\delta_S$ in $A_{2\kappa}[x_0]$, we have $R_{2\kappa} = R_\kappa + \delta_R$ and $S_{2\kappa} = S_\kappa + \delta_S$. The bottleneck is the computation of the normal form of $f, g$ and $J$: the algorithm in [2, 4] does $O(d)$ operations in $B_\kappa$, for a total of $O^*(\kappa d^3)$ operations in $k$. Summing over all lifting steps, with $\kappa = 1, 2, 4, 8, \ldots$ up to about $\kappa \approx d^2$ leads to the bound $O^*(d^5)$, which is no better than Reischert’s algorithm.

We now sketch how to compute e.g. the normal form of $f$ in $B_k$ more efficiently, using a baby-steps / giant-steps approach inspired by Brent and Kung’s algorithm [1].

1. Seeing $f$ as a polynomial of degree $d$ in $x_1$, with coefficients $f_i \in A_{2\kappa}[x_0]$ of degree $d$ in $x_0$, build the $\sqrt{d+1} \times \sqrt{d+1}$ matrix $M_1 = (f(\sqrt{d+1} t^{-j}))$ with entries in $A_{2\kappa}[x_0]$.

2. Compute $\sigma_0 = S_\kappa^0, \sigma_1 = S_\kappa^1, \ldots, \sigma_{\sqrt{d+1} - 1} = S_\kappa^{\sqrt{d+1} - 1}$ in $B_k$ (baby steps).

3. Cut all $\sigma_i$ into slices, writing $\sigma_i = \sum_{j=0}^{d-1} \sigma_{i,j} x_0^d$, with $\sigma_{i,j} \in A_{2\kappa}[x_0]$ of degree less than $d$ in $x_0$. Build the $\sqrt{d+1} \times d$ matrix $M_2 = (\sigma_{i,j})$ and compute $M = (m_{i,j}) = M_1 M_2$.

4. Using the $m_{i,j}$, reconstruct $f \mod (R_\kappa, x_1 - S_\kappa)$ using Horner’s scheme (giant steps).

As in Brent and Kung’s algorithm, the dominant cost is matrix multiplication (Step 3). We do matrix multiplication in sizes $\sqrt{d+1} \times \sqrt{d+1}$ and $\sqrt{d+1} \times d$, with entries in $k[t, x_0]$, of degrees at most $2\kappa$ in $t$ and $d$ in $x_0$. The cost is thus $O^*(\kappa d^{\frac{\sqrt{d+1}+3}{2}})$ operations in $k$.

Summing over all lifting steps, using the fact that we take $\kappa = 1, 2, 4, 8, \ldots$ (powers of 2) until approximately $d^2$, the total cost is $O^*(d^{\frac{3\sqrt{d+1}+7}{2}})$, as claimed.

References
Sparse Differential Resultant for Laurent Differential Polynomials

Wei Li, Chun-Ming Yuan, Xiao-Shan Gao
KLMM, AMSS, Chinese Academy of Sciences
Beijing, China, 100190
{liwei,cmyuan,xgao}@mmrc.iss.ac.cn

Abstract

In this poster, we first introduce the concept of Laurent differentially essential systems and give a criterion for them in terms of their supports. Then the sparse differential resultant for a Laurent differentially essential system is defined and its basic properties are given. In particular, order and degree bounds for the sparse differential resultant, as well as a BKK-type degree bound for the differential resultant, are given. Based on these bounds, an algorithm to compute the sparse differential resultant is proposed, which is single exponential.

1 Definition of Sparse Differential Resultant

Let \( E \) be an ordinary differential field contained in a universal differential field \( F \) and \( Y = \{ y_1, \ldots, y_n \} \) a set of differential indeterminates over \( E \). Throughout this poster, we shall use the prefix “\( \delta \)-” as a synonym of “differential” or “differentially” and use \( a ^{(k)} \) to denote \( \delta^k a \).

**Definition 1** A Laurent \( \delta \)-monomial of order \( s \) is of the form \( \prod_{i=1}^{n} \prod_{k=0}^{s} (y_i ^{(k)})^{d_{ik}} \) where \( d_{ik} \) are integers which can be negative. A Laurent \( \delta \)-polynomial is a finite linear combination of Laurent \( \delta \)-monomials with coefficients from \( E \). All Laurent \( \delta \)-polynomials form the Laurent \( \delta \)-polynomial ring, denoted by \( E \{ Y, Y^{-1} \} \). For \( f \in E \{ Y, Y^{-1} \} \), \( (a_1, \ldots, a_n) \in E^n \) is called a non-polynomial solution of \( f \), if \( f(a_1, \ldots, a_n) = 0 \) and for each \( i \) and \( k \in \mathbb{N} \), \( a_i ^{(k)} \neq 0 \).

Suppose \( A_i = \{ M_{i0}, M_{i1}, \ldots, M_{il_i} \} \) \( (i = 0, 1, \ldots, n) \) are finite sets of Laurent \( \delta \)-monomials. Consider \( n + 1 \) generic Laurent \( \delta \)-polynomials defined over \( A_i \) \( (i = 0, 1, \ldots, n) \):

\[
\mathbb{P}_i = \sum_{k=0}^{l_i} u_{ik} M_{ik} \ (i = 0, \ldots, n),
\]

where all the \( u_{ik} \in E \) are \( \delta \)-independent over \( \mathbb{Q} \). Denote \( u_i = (u_{i0}, u_{i1}, \ldots, u_{in}) \).

**Definition 2** A set of Laurent \( \delta \)-polynomials of form (1) is called a Laurent differentially essential system if there exist \( k_i \ (1 \leq k_i \leq l_i) \) such that \( \text{d.tr.deg} \mathbb{Q} \langle \frac{M_{i0}}{M_{i0}^{k_0}}, \frac{M_{i1}}{M_{i0}^{k_1}}, \ldots, \frac{M_{il_i}}{M_{i0}^{k_{l_i}}} \rangle / \mathbb{Q} = n \).
Theorem 3 For $\mathbb{P}_i$ given in (1), let $q_j = \max_{i=0}^n \text{ord}(\mathbb{P}_i, y_j)$ and $M_{ik}/M_{0i} = \prod_{j=1}^n \prod_{l=0}^{q_j} (y_j^l)^{tikjl}$. Denote $d_{ij} = \sum_{k=0}^{tikj} u_{ikj} \sum_{l=0}^{q_j} t_{ikjl}$ where $x_j$ are algebraic indeterminates. Let $M_{\mathbb{P}} = (d_{ij})_{n \times n}$. Then $\mathbb{P}_0, \ldots, \mathbb{P}_n$ form a Laurent $\delta$-essential system if and only if $\text{rk}(M_{\mathbb{P}}) = n$.

Suppose $\mathbb{P}_i$ form a Laurent $\delta$-essential system. Let $[\mathbb{P}_0, \ldots, \mathbb{P}_n] \subset \mathbb{Q}\{\mathbb{Y}, \mathbb{Y}^{-1}, u_0, \ldots, u_n\}$. Then there exists an irreducible $\delta$-polynomial $R \in \mathbb{Q}\{u_0, \ldots, u_n\}$ such that $[\mathbb{P}_0, \ldots, \mathbb{P}_n] \cap \mathbb{Q}\{u_0, \ldots, u_n\} = \text{sat}(R)$.

This $R$ is defined to be the sparse differential resultant of $\mathbb{P}_i$.

2 Basic Properties of Sparse Differential Resultant

Theorem 4 Let $\mathbb{P}_i (i = 0, \ldots, n)$ be a Laurent $\delta$-essential system of form (1). The sparse differential resultant $R(\mathbb{u}_0, \ldots, \mathbb{u}_n)$ of $\mathbb{P}_i$ has the following properties.

1) $R(\mathbb{u}_0, \ldots, \mathbb{u}_n)$ is $\delta$-homogenous in each $u_i (i = 0, \ldots, n)$.

2) Let $Z_0(\mathbb{P}_0, \ldots, \mathbb{P}_n)$ be the set of all $\delta$-specializations of $u_{ik}$ under which the $\mathbb{P}_i$ have a common non-polynomial solution. Then its Kolchin $\delta$-closure $Z_0(\mathbb{P}_0, \ldots, \mathbb{P}_n) = \mathbb{V}(\text{sat}(R))$.

3) Assume that $\mathbb{P}_i (i = 0, \ldots, n)$ have the same support set $A$. The differential toric variety $X_A$ associated with $A$ is defined and $R$ is shown to be the projective differential Chow form of $X_A$.

4) Suppose $t_0 = \text{deg}(R, u_{00}) \geq 0$. Then there exist $\xi_{rk}$ in certain $\delta$-field $\mathcal{F}_r$ such that $R = A \prod_{r=1}^{t_0} (u_{00} + \sum_{k=1}^{t_0} w_{0k} \xi_{rk})^{(h_0)}$, where $A$ is a polynomial in $\mathbb{Q}\{u_1, \ldots, u_n\}[u_{00} \setminus u_{00}]$.

5) $h_i = \text{ord}(R, u_i) \leq s - s_i$, where $s = \sum_{i=0}^n s_i$ and $s_i = \text{ord}(\mathbb{P}_i, \mathbb{Y})$. Moreover, $\text{deg}(R) \leq \prod_{i=0}^n (m_i + 1)^{h_i+1} \leq (m+1)^{ns+n+1}$, where $m_i = \text{deg}(\mathbb{P}_i, \mathbb{Y})$ and $m = \max\{m_i\}$.

6) $R$ has a representation $\prod_{i=0}^n M_{0i}^{(h_i+1)} \text{deg}(R) \cdot R = \sum_{i=0}^n h_i G_{ij}(\mathbb{P}_i)^{(j)}$ where $\text{deg}(G_{ij}(\mathbb{P}_i)^{(j)}) \leq [m+1 + \sum_{i=0}^n (h_i+1) \text{deg}(M_{0i})] \text{deg}(R)$.

Let $\mathbb{P}_i (i = 0, \ldots, n)$ be generic $\delta$-polynomials of order $s_i$ and degree $m_i$. Then the differential resultant $R(\mathbb{u}_0, \ldots, \mathbb{u}_n)$ of $\mathbb{P}_i$ has a BKK-style degree bound:

Theorem 5 For each $i \in \{0, 1, \ldots, n\}$,

$$\text{deg}(R, u_i) \leq \sum_{j=0}^{s-s_i} \text{M}((Q_{ij})_{j \neq i, 0 \leq t \leq s-s_i}, Q_0, \ldots, Q_{i,k-1}, Q_{i,k+1}, \ldots, Q_{i,s-s_i})$$

where $Q_{ij}$ is the Newton polytope of $(\mathbb{P}_i)^{(l)}$ in $\mathbb{Y}[s]$ and $\text{M}(\cdot)$ means taking the mixed volume.

Based on the order and degree bounds given in 5) and 6) of Theorem 4, a single exponential algorithm to compute the sparse resultant $R$ is proposed. Precisely,

Theorem 6 The sparse differential resultant of $\mathbb{P}_0, \ldots, \mathbb{P}_n$ can be computed with at most $O((m+1)^{O(nds)}((n+1)(s+1))^{O(ls)})$ $\mathbb{Q}$-arithmetic operations, where $l = \sum_{i=0}^n (l_i + 1)$.

References

Numeric-symbolic methods for computing the Liouvillian solutions of
differential equations and systems

Alberto Llorente and Jorge Mozo-Fernández
University of Valladolid, 47005 Valladolid, Spain
llorente@agt.uva.es jmozo@maf.uva.es

The objective of this work is giving a numeric-symbolic algorithm for finding the Liouvillian solutions of a linear system of differential equations

$$A_0(x) y + A_1(x) y' + \cdots + A_r(x) y^{(r)} = 0$$  \hfill (1)

with $A_0(x), A_1(x), \ldots, A_1(x) \in \mathbb{C}(x)^{n \times n}$ and $\det A_r(x) \neq 0$. Here numeric-symbolic means a numerical stage followed by a symbolic stage for a symbolic correct output.

Cyclic Vector Lemma allows to extend a theorem of Singer [2] to systems. (He proved the case $n=1$.)

**Theorem 1** There exists a function $I : \mathbb{N} \to \mathbb{N}$ such that, if (1) has a non-zero Liouvillian solution, then there exist an algebraic extension $F/\mathbb{C}(x)$ of degree $I(rn)$ at most and a non-zero solution $(y_1, y_2, \ldots, y_n)^\top$ of (1) and, for each $i$ and $j$ with $y_i \neq 0$, $y_i'/y_i \in F$ and $y_j/y_i \in F$.

Recent group-theoretical results allow us to compute $I$, finding $I(k) = (k + 1)!$ for $k \geq 14$.

Following the ideas of van der Hoeven [5], we use differential Galois theory and effective complex numerics. Differential Galois theory associates to (1) an algebraic group of matrices $G$, with Lie algebra $\mathfrak{g}$, such that a solution of Theorem 1 is a common eigenvector of $\mathfrak{g}$ whose images by $G$ fall in $I(rn)$ lines at most. The differential Galois group $G$ is the Zariski closure of the group generated by certain groups: the monodromy, the exponential tori and the Stokes matrices. These groups are computable in effective complex numerics. The computation of the monodromy uses [3]; the exponential tori and the Stokes matrices required [6] in [5], but not in this work.

The effective complex numerics are introduced (in the real case) in [4]. These numbers are a black box with input $\varepsilon > 0$ and output an approximation within $\varepsilon$ of the complex number represented. Effective complex numbers can be added and multiplied, and divided if the denominator is bounded from below. Also roots of polynomials with effective complex coefficients are effective, so these numbers form an algebraically closed field. What cannot be computed in finite time is whether an effective complex number is zero, so we substitute $|a| < \text{tol}$ for $a = 0$ when needed, for certain global variable $\text{tol}$. This introduces certain controlled error in linear algebra: the rank of a set of vectors may be undercomputed, but never overcomputed, and, for $\text{tol}$ small enough, the computation is exact.

Derksen [1] and van der Hoeven [5] gave an algorithm for computing the Zariski closure of a finitely generated group, but this algorithm requires computing the multiplicative syzygies of the eigenvalues, a problem harder than checking if an eigenvalue is a root of unity, which is insolvable in finite time because the roots of unity are dense and codense in the unit circle. In order to make the Derksen–van der Hoeven algorithm effective, we propose waiving the objective of the Zariski closure and to compute a greater closure, the eurymeric closure, to be defined.

A eurymeric group is a Zariski-closed group of invertible matrices whose Lie algebra is closed under the product of matrices and contains the identity. The eurymeric closure of $G$ is the smallest eurymeric
group $H$ containing $G$. Eurymeric groups have good properties that linearize some non-linear steps in the Derksen–van der Hoeven algorithm, and reduce the computation of all the multiplicative syzygies of the eigenvalues to testing if any quotient of the eigenvalues is a root of unity. This is still insolvable in effective complex numerics, but we may truncate the process (essentially, truncate the Euclidean algorithm) when we get that, in case of being a root of unity, its order would be greater than $I(rn)$. Thus we compute an augmentation $H_0$ of $H$, with Lie algebra $h_0$, and we may prove that it keeps the solutions of Theorem 1: if $g$ has nonzero common eigenvectors, then $h_0$ has nonzero common eigenvectors and they are common eigenvectors of $g$.

The errors coming from linear algebra computations, for $\texttt{tol}$ too large, might undercompute $H_0$, and hence overcompute the common eigenvectors of $h_0$, but for $\texttt{tol}$ small enough the computation is exact. Let $H_{\texttt{tol}}$ be the computed version of $H_0$, and $h_{\texttt{tol}}$ its Lie algebra. We pick a common eigenvector of $h_{\texttt{tol}}$ and compute its orbit by $H_{\texttt{tol}}$; we continue with a complementary space to the linear span of the orbit. Then we reconstruct symbolically the solutions and check if there are solutions of (1) in the way explained below. A solution of Theorem 1 is represented by a Darboux polynomial, described in [7], easy to check without computing symmetric powers. The symbolic reconstruction of the rational functions in the Darboux polynomial is done by the methods exposed in [5]. If we find solutions, we have succeeded. If we find that there is no common eigenvector of $h$, we say that zero is the only Liouvillian solution of (1). If there are common eigenvectors but our candidate solutions are wrong, then we have computed with $\texttt{tol}$ too large and then we restart the algorithm with smaller $\texttt{tol}$. If the successive values of $\texttt{tol}$ converge to zero, the computations will be eventually exact and the algorithm terminates with a correct output.

The algorithm avoids computing the symmetric powers and it may end early when the only Liouvillian solution is zero, an advantage over the usual methods. The algorithm also works directly with systems, without converting them into scalar equations.

References


The matrix based representations of the intersection curves

Thang Luu Ba\textsuperscript{1,2}

\textsuperscript{1}Department of Informatics & Telecommunications, University of Athens, Greece

\textsuperscript{2}Department of Mathematics, Hanoi National University of Education, Hanoi, Vietnam

thanglb@di.uoa.gr

Evaluating the intersection of two rational parameterized algebraic surfaces is an important problem in solid modeling. In this work, we make use of some generalized matrix based representations of parameterized surfaces in order to represent the intersection curve of two such surfaces as the zero set of a matrix determinant. As a consequence, we extend to a dramatically larger class of rational parameterized surfaces, the applicability of a general approach to the surface/surface intersection problem due to J. Canny and D. Manocha \cite{4}. Notice that in \cite{4} only some particular surface parameterizations, namely the parameterizations without base points (i.e. that are well defined over all their parameter space) can be represented by such matrices. This absence of base points is definitely a very strong constraint in the context of applications in solid modeling. It is hence the main limitation of applicability of the Canny-Manocha matrix based approach to the surface/surface intersection problem that is nevertheless an interesting compromise to provide an efficient, robust and accurate solution to this problem. The goal of our work is to overcome this limitation. This abstract is part of the work in \cite{1, 3}.

For instance, suppose given a parameterization

\[ \mathbb{P}^2_C \to \mathbb{P}^3_C : (s : t : u) \mapsto (f_1(s,t,u) : f_2(s,t,u) : f_3(s,t,u) : f_4(s,t,u)) \]

of a surface \( S \) such that \( \gcd(f_1,\ldots,f_4) \in \mathbb{C}\setminus\{0\} \). In \cite{1}, there exists a matrix \( M(\phi)(x,y,z,w) \) which is said to be a representation matrix of \( \phi \) because it satisfies the following properties:

- When specializing \( M(\phi) \) at a given point \( P \in \mathbb{P}^3_C \), its rank drops if and only if \( P \) belongs to \( S \).

- The greatest common divisor of the maximal minors of \( M(\phi) \) is equal to the implicit equation of \( S \) raised to the power \( \deg(\phi) \).

Moreover, the entries of \( M(\phi)(x,y,z,w) \) are linear forms in \( \mathbb{C}[x,y,z,w] \). From a computational point of view, the matrix \( M(\phi) \) with the smallest possible value of sizes has to be chosen. It is rarely a square matrix. Also, notice that the second property given above is never used for computations.

Suppose given two distinct parameterized surfaces \( S_1 \) and \( S_2 \). A standard problem in non linear computational geometry is to determine the set \( S_1 \cap S_2 \) which is a curve in \( \mathbb{P}^3_C \). As we explained above, one can build a representation matrix of \( S_1 \) that we will denote by \( M(x,y,z,w) \). Let

\[ \Psi : \mathbb{P}^2_C \to \mathbb{P}^3_C : (s : t : u) \mapsto (a(s,t,u) : b(s,t,u) : c(s,t,u) : d(s,t,u)) \]

be a parameterization of \( S_2 \) where \( a(s,t,u), b(s,t,u), c(s,t,u), d(s,t,u) \) are homogeneous polynomials of the same degree and without common factor in \( \mathbb{C}[s,t,u] \). By substituting in the matrix \( M(x,y,z,w) \) the variables \( x,y,z,w \) by the homogeneous polynomials \( a(s,t,u), b(s,t,u), c(s,t,u), d(s,t,u) \) respectively, we get the matrix

\[ M(s,t,u) := M(\Psi(s : t : u)) = M(a(s,t,u), b(s,t,u), c(s,t,u), d(s,t,u)). \]

114
From the properties of the representation matrix $M(x, y, z, w)$, we known that $M(s, t, u)$ has maximal rank $\rho$ ($\rho$ is the number of rows of $M$). Moreover, for all point $(s_0 : t_0 : u_0) \in \mathbb{P}^2\mathbb{C}$ we have

\[
\text{rank}(M(s_0, t_0, u_0)) < \rho \text{ if and only if } \begin{cases} 
\Psi(s_0 : t_0 : u_0) \in S_1 \cap S_2 \text{ or } \\
(s_0 : t_0 : u_0) \text{ is a base point of } \Psi.
\end{cases}
\] (1)

The equivalence (1) shows that the spectrum of the matrix $M(s, t, u)$, that is to say the set

\[
\{(s_0 : t_0 : u_0) \in \mathbb{P}^2\mathbb{C} \text{ such that rank } M(s_0, t_0, u_0) < \rho\},
\]
yields the intersection locus $S_1 \cap S_2$ plus the base points of the parameterization $\Psi$ of $S_2$.

**Theorem 1.** The spectrum of the matrix $M(s, t, u)$ is an algebraic curve in $\mathbb{P}^2\mathbb{C}$, that is to say is equal to the zero locus of a homogeneous polynomial in $\mathbb{C}[s, t, u]$. In particular, there is no isolated points in the spectrum of $M(s, t, u)$.

As a consequence of Theorem 1, if we use matrix representations to deal with the surface/surface intersection problem, we will end up at some point with a pencil of bivariate and non-square matrices that represents the intersection curve (after dehomogenization). Therefore, in order to be able to handle this intersection curve, for instance to determine its exact topology, it is necessary to extract a pencil of bivariate and square matrices that yields a matrix representation of the intersection curve as a matrix determinant. For that purpose, we develop an algorithm (called $\Delta W$-Decomposition) based on the remarkable work of V. N. Kublanovskaya [2].

We build two companion matrices $A(t)$ and $B(t)$ which allows to linearize the polynomial matrix $M(s, t, 1)$ such that the spectrum of the matrix $M(s, t, 1)$ coincides the spectrum of the matrix $A(t) - sB(t)$. Then, we provide an algorithm that extracts a square matrix whose determinant represents the intersection locus curve $S_1 \cap S_2$.

**Theorem 2.** A pencil of polynomial matrices $A(t) - sB(t)$ is equivalent to a pencil of the following form

\[
\begin{pmatrix}
M_{1,1}(s, t) & 0 & 0 \\
M_{2,1}(s, t) & M_{2,2}(s, t) & 0 \\
M_{3,1}(s, t) & M_{3,2}(s, t) & M_{3,3}(s, t)
\end{pmatrix}
\]

where the pencil $M_{2,2}(s, t)$ is a regular pencil that corresponds to the intersection locus curve $S_1 \cap S_2$.

An implemented Maple is freely available at the URL: [http://cgi.di.uoa.gr/~thanglb/](http://cgi.di.uoa.gr/~thanglb/)

**Acknowledgements.** This work is partially supported by Marie-Curie Initial Training Network “SAGA” (Shapes, Geometry, Algebra), FP7-PEOPLE contract PITN-GA-2008-214584.

**References**


Backward error analysis of approximate Gröbner basis

Kosaku Nagasaka
Kobe University, Kobe, JAPAN, 657-8501
nagasaka@main.h.kobe-u.ac.jp

1 Introduction

There are many algorithms for computing approximate Gröbner basis and they can be thought as just one of symbolic-numeric computations for polynomials (see [1] for some note from the methological point of view). However, there is a big difference between approximate Gröbner basis and others that the backward error analyses are naturally given or not (i.e. easy or not). For example, we consider an approximate factorization of the following irreducible polynomial.

\[ f(x, y, z) = 81x^4 + 72x^2y^2 + \frac{3}{1292}x^2z^2 - 648x^2 + 16y^4 + \frac{1}{969}y^2z^2 - 288y^2 - \frac{837227}{1292}z^4 - \frac{3}{323}z^2 + 1296. \]

We can have the following factorization \( \hat{f}_1(x, y, z)\hat{f}_2(x, y, z) \) with tolerance \( \varepsilon = 4.54478 \times 10^{-6} \).

\[ \hat{f}_1(x, y, z) = 9.000x^2 + 4.000y^2 - 25.46z^2 - 36.00, \quad \hat{f}_2(x, y, z) = 9.000x^2 + 4.000y^2 + 25.46z^2 - 36.00. \]

The resulting approximate factorization \( \hat{f}_1(x, y, z)\hat{f}_2(x, y, z) \) can be characterized as the factorization of the following polynomial in the exact sense, by rationalizing the coefficients and multiplying them.

\[ f(x, y, z) = 81x^4 + 72x^2y^2 - 648x^2 + 16y^4 - 288y^2 - \frac{78400}{121}z^4 + 1296. \]

In contrast, approximate Gröbner basis does not have this behavior in general. For example, the following \( \tilde{G}_{app} \) is the approximate Gröbner basis of the input \( \tilde{F}_{app} \) w.r.t. the graded lexicographic order \((x > y)\), computed by Mathematica (we note that other algorithms also have similar behaviors).

\[
\tilde{F}_{app} = \{ \hat{f}_1(\vec{x}) = 0.01084x^3y + 0.891x^3, \hat{f}_2(\vec{x}) = 0.503xy^3 + 0.1129x + 0.02201 \}, \n\tilde{G}_{app} = \{ 1.0xy^3 + 0.224453x + 0.0437575, 1.0x^3 - 7.87965 \times 10^{-8}x^2 \}.
\]

However, we have the following result if we compute a Gröbner basis of \( \tilde{G}_{app} \) with the rationalized coefficients in the exact sense (we show it in floating-point numbers due to the narrow paper width).

\[ G_{app} \approx \{ 1.23120 \times 10^{30}y^3 + 6.83710 \times 10^{35}, 3.12500 \times 10^{21}x - 2.46239 \times 10^{14} \}. \]

Moreover, the following \( G_{ex} \) is a Gröbner basis of the ideal generated by \( \tilde{F}_{app} \) with the rationalized coefficients, which is different from \( G_{app} \) and \( \tilde{G}_{app} \). We note that the resulting \( G_{ex} \) may not be the basis we want since the input system may have a priori errors on their coefficients and supports.

\[ G_{ex} \approx \{ 5.55931 \times 10^{17}x - 4.38054 \times 10^{10}, 271.000y + 22275.0 \}. \]

Therefore, the resulting approximate Gröbner basis is not a Gröbner basis and does not generate the given ideal in the exact sense. This behavior is common for algorithms computing approximate Gröbner basis hence we have a very natural question: “What is that we computed?” In this poster, we introduce a proof of concept method to find an exact result from those approximate Gröbner bases over the set of floating-point numbers \( \mathbb{F} \), so we can have a backward error analysis. We note that this poster does not introduce any newly defined approximate Gröbner basis. Our aim is just only finding an answer for the above question across several definitions and methods from the exact point of view.
2 Nearby Gröbner basis and system

Problem 1 For the given $\tilde{F}_{app}, \tilde{G}_{app} \subset \mathbb{F}[\vec{x}]$, compute $F_{cl}, G_{cl} \subset \mathbb{R}[\vec{x}]$ such that $G_{cl}$ is a Gröbner basis of $\text{ideal}(F_{cl})$ in the exact sense and $F_{cl}$ and $G_{cl}$ are close to $F_{app}$ and $G_{app}$, respectively.

For this problem we propose the following method: 1) find constraints w.r.t. a close enough exact Gröbner basis $G_{cl}$ of itself to the given approximate Gröbner basis $G_{app}$, 2) solve least squares w.r.t. a close enough system $F_{cl}$ that is a subset of the ideal generated by the resulting exact Gröbner basis $G_{cl}$, and 3) solve the minimization problem w.r.t. a close enough system $F_{cl}$.

We show an example of this method for the given $\tilde{F}_{app}$ and $\tilde{G}_{app}$ in the previous section. At first, we construct a set of parametric polynomials:

$$G_{par} = \{ g_1(\vec{x}) = xy^3 + p_{12}x + p_{13}, g_2(\vec{x}) = x^3 + p_{22}x^2 \} \subset \mathbb{R}[\vec{p}][\vec{x}].$$

The corresponding constraints ($p_{13} = 0$, $p_{13}p_{22} = 0$) are computed by the monomial reduction:

$$\text{Spoly}(g_1, g_2) = -p_{22}x^2y^3 + p_{12}x^3 + p_{13}x^2 \implies \text{Spoly}(g_1, g_2)^{G_{par}} = p_{13}x^2 + p_{13}p_{22}x = 0.$$

If we assume that we have $\text{supp}(f_1(\vec{x})) = \{ x^3y, x^3, x^2y, x^2 \}$ and $\text{supp}(f_2(\vec{x})) = \{ xy^3, x, 1 \}$ for $F_{cl} = \{ f_1(\vec{x}) = \sum_i h_{1i}(\vec{x})g_i(\vec{x}), f_2(\vec{x}) = \sum_i h_{2i}(\vec{x})g_i(\vec{x}) \}$. The problem becomes the least squares:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ p_{22} & 0 \end{pmatrix} \vec{h}_1 = \begin{pmatrix} \frac{271}{5000} \\ \frac{891}{1000} \\ 0 \end{pmatrix}, \quad \begin{pmatrix} 1 & p_{12} \\ p_{13} \end{pmatrix} \vec{h}_2 = \begin{pmatrix} \frac{503}{1000} \\ \frac{1129}{1000} \\ \frac{2201}{10000} \end{pmatrix}, \quad \vec{h}_1 \in \mathbb{R}(\vec{p})^2, \vec{h}_2 \in \mathbb{R}(\vec{p})^1.$$

This can be solved by the exact arithmetic (e.g. LUP, LSP or LQUP matrix decompositions) and we have the following $F_{cl}$.

$$F_{cl} = \{ \frac{271x^3y+22275x^3+271p_{22}x^2y+22275p_{22}x^2}{25000(p_{13}+1)}, \frac{11290p_{12}+2201p_{13}+50300}{100000(p_{13}+1)}x^3y+(11290p_{12}+2201p_{13}+50300)p_{13}x+(11290p_{12}p_{13}+2201p_{13}^2+50300p_{13}) \} \subset \mathbb{R}(\vec{p})[\vec{x}].$$

To get the values of parameters ($p_{12}, p_{13}, p_{22}$), we solve the following minimization problem.

$$\min_{\vec{p}} \sum_{i=1}^{m} ||f_i(\vec{x}) - \tilde{f}_i(\vec{x})|| \quad \text{subject to } p_{13} = 0 \text{ and } p_{13}p_{22} = 0.$$

Finally, we get the following result which makes us to be able to evaluate the backward error.

$$G_{cl} = \{ xy^3 + \frac{1129}{5030}x, x^3 \}, \quad F_{cl} = \{ \frac{271}{5000}x^3y + \frac{891}{1000}x^3, \frac{503}{1000}x^3y + \frac{1129}{1000}x \}.$$

We note that the author wishes to thank Prof. Kaltofen and Lichtblau for their comments at Hybrid2011 which led me to the problem and this work was supported in part by Japanese MEXT KAKENHI (22700011). The analyses for other algorithms will be given in the poster presentation at ISSAC 2012, computed by our preliminary implementation on Mathematica.

(URL: http://wwwmain.h.kobe-u.ac.jp/~nagasaka/research/snap/issac12.nb)

References

Polynomial Root-finding via Structured Matrices

Esteban Segura Ugalde
XLIM-DMI UMR 7252 University of Limoges - CNRS
Limoges, France
estebansu@gmail.com

Abstract
We study the problem of approximating the zeros of an univariate polynomial (up to machine precision). Some popular iterative root-finding methods construct companion matrices (Frobenius, Lagrange) associated with the given polynomial and use eigensolvers to find the eigenvalues of such matrices. Our goal is to study this root-finding technique, exploiting the structure (e.g., diagonal plus rank one) of companion matrices to obtain a decrease of computational cost and memory requirements.

1 Introduction and Preliminaries
We study the problem of approximating (up to machine precision) the zeros of an univariate polynomial of degree $n$:

$$P(x) = p_n x^n + p_{n-1} x^{n-1} + \ldots + p_1 x + p_0$$

This is a classic problem in computational mathematics and still an active research topic. Our starting point is Fortune’s Eigensolve algorithm ([2], [3]). Eigensolve is an iterative method that approximates the roots of univariate polynomials (up to machine precision) and it is based on Weierstrass (Durand-Kerner) method. Each iteration involves a double precision computation of the eigenvalues of a Lagrange companion matrix (using the standard QR algorithm) and high precision for the computation of Lagrange coefficients.

The QR algorithm, however, requires $O(n^2)$ flops per iteration (resulting in $O(n^3)$ overall complexity) and quadratic memory space. Therefore this method may become too expensive for large matrices. But if the matrix whose eigenvalues are sought has a special structure (e.g., diagonal plus rank one (DPR1)), then one may replace classical QR-based eigensolvers with structured algorithms requiring $O(n^2)$ complexity and $O(n)$ storage.

Motivated by a remark in [1], we explore the feasibility and practical stability properties of structured methods in this context: in particular, we apply a structured version of the inverse power method ([1]) and show that the resulting algorithm retains the favorable properties (practical stability, convergence) of the original one, while decreasing computational cost and memory requirements.

2 Main Contribution
We implemented in Matlab an Eigensolve-like algorithm (called $\text{eigensolveB}$), where the structured shifted inverse power method is used (instead of QR) for computing the eigenvalues of the
Lagrange matrix. Our convergence test employs convergence disks and the criterion proposed in [4]. Implicit deflation was used (once a root had been found) to approximate the remaining roots and reduce the computational cost in the next iterations. This implementation can be downloaded from: http://www.unilim.fr/pages_perso/paola.boito/stageM2.html

For comparison purposes, we also implemented a “toy version” of Fortune’s algorithm (called eigen-solveA), where we did not take into account the structure of the matrix and used the Matlab command eig for eigenvalue computation.

Note that (since we are working in Matlab) we do not expect to see gains in running time, as we are competing against built-in routines.

3 Numerical Results and Conclusions

<table>
<thead>
<tr>
<th>Polynomial</th>
<th>eigen-solveA</th>
<th>eigen-solveB</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_{30}$</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$W_{100}$</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>$W_{150}$</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>$T_{64}$</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$T_{128}$</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>$M^{(4)}$</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$M^{(6)}$</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

This table compares the number of iterations required by the nonstructured and structured versions, applied to Wilkinson (degree 30, 100, 150), Toeplitz (degree 64) and Mandelbrot polynomials. This is a very preliminary work, seeking to explore the role of structured computations within well-established root-finding methods. We noticed that, in most cases, the introduction of structured computations does not significantly worsen the stability and convergence properties of the eigen-solver; in fact, it even improves them in some cases.

Ideas for further work include writing an implementation in C or C++, and experimenting with more structured algorithms for computing the eigenvalues of generalized companion matrices.

References


Maximal Perturbation for Preserving the Number of Solutions of a Polynomial System

Hiroshi Sekigawa
Department of Mathematics
Tokai University
4-1-1 Kitakaname, Hiratsuka-shi, Kanagawa 259-1292, Japan
sekigawa@tokai-u.jp

Kiyoshi Shirayanagi
Department of Information Science
Toho University
2-2-1 Miyama, Funabashi-shi, Chiba 274-8510, Japan
kiyoshi.shirayanagi@is.sci.toho-u.ac.jp

Given $f_1, \ldots, f_n \in \mathbb{C}[x_1, \ldots, x_n]$, we investigate the maximal perturbation that preserves the number of solutions in $(\mathbb{C}^n)^n$ of the polynomial system $f_1 = \cdots = f_n = 0$, where $\mathbb{C}^n = \mathbb{C} \setminus \{0\}$, by using the equality condition of the Bernstein, Kushnirenko, and Khovanskii (BKK) bound.

First, to describe our problem, we summarize some properties of the nearest polynomial with a given zero and the BKK bound. For simplicity, we write a monomial $x_1^{s_1} \cdots x_n^{s_n}$ in $n$ variables $x_1, \ldots, x_n$ as $x^s$ and the $n$-tuple $(0, \ldots, 0)$ as $\mathbf{0}$. For a complex Laurent polynomial $f = \sum c_s x^s$, we define $\text{supp}(f)$, the support of $f$, as $\{s \in \mathbb{Z}^n | c_s \neq 0\}$. Throughout this extended abstract, we fix a norm $\| \cdot \|$ of complex vectors, write its dual norm as $\| \cdot \|^*$, and for $f, g \in \mathbb{C}[x_1, \ldots, x_n]$, define the distance between $f$ and $g$ as $\|f - g\|$, the norm of the vector of coefficients of $f - g$.

For a finite set $S \subset \mathbb{N}_0^n$ and $z \in \mathbb{C}^n$, we define $Z(S; z) \subset \mathbb{C}[x_1, \ldots, x_n]$ by $\{p \in \mathbb{C}[x_1, \ldots, x_n] | \text{supp}(p) \subset S, p(z) = 0\}$. Let $f \in \mathbb{C}[x_1, \ldots, x_n]$ and $S = \text{supp}(f)$. From the results of Stetter’s paper [2], for $z \in \mathbb{C}^n$ and the nearest polynomial $\tilde{f} \in Z(S; z)$ to $f$, $\|\tilde{f} - f\| = |f(z)/\|z^*\|_{S \in S}||^*$ holds if $f$ has a nonzero constant term or $z \in (\mathbb{C}^n)^n$. Thus, $N(f; z) = \|\tilde{f} - f\|$ is a continuous function of $z \in \mathbb{C}^n$ if $f$ has a nonzero constant term.

Let $\alpha \in \mathbb{Q}^n \setminus \{\mathbf{0}\}$. For a compact subset $S$ of $\mathbb{R}^n$, let $m(\alpha, S) = \min_{s \in S} \langle \alpha, s \rangle$, where $\langle \cdot, \cdot \rangle$ is the standard inner product on the vector space $\mathbb{R}^n$. We write the set $\{s \in S | \langle \alpha, s \rangle = m(\alpha, S)\}$ as $S_\alpha$. If $\text{supp}(f) = S$, define $f_\alpha = \sum_{s \in S_\alpha} c_s x^s$ for $f = \sum_{s \in S} c_s x^s$. Let $F = (f_\alpha, \ldots, f_n)$ be a Laurent polynomial system and define $F_\alpha = (f_1, \ldots, f_n)$. Let $V(P)$ be the standard $n$-dimensional Lebesgue measure of the Newton polytope $P \in \mathbb{R}^n$ of $F$. Then, Bernstein’s theorem can be described as follows.

Theorem 1 (Bernstein [1]) Counting multiplicities, $F$ has no more than $V(P)$ isolated roots in $(\mathbb{C}^n)^n$. Also, if $F_\alpha$ has no roots in $(\mathbb{C}^n)^n$ for any $\alpha \in \mathbb{Q}^n \setminus \{\mathbf{0}\}$, then $F$ has exactly $V(P)$ roots in $(\mathbb{C}^n)^n$, and if $V(P) > 0$, the converse holds as well.

The bound in Theorem 1 is called the BKK bound. Note that $\{F_\alpha | \alpha \in \mathbb{Q}^n \setminus \{\mathbf{0}\}\}$ is a finite set.

We define $D(F; z) = \max_{1 \leq i \leq n} \{N(f_i; z)\}$. The following is our problem.

\*This work was supported by the Japan Society for the Promotion of Science through Grant-in-Aids for Scientific Research (KAKENHI) 21500026 and 24500022.
Problem 1 (Computation for the Maximal Perturbation) Let $F$ be a given polynomial system $(f_1, \ldots, f_n)$ satisfying the equality condition in Theorem 1, where $f_i \in \mathbb{C}[x_1, \ldots, x_n]$ for $i = 1, \ldots, n$. Compute

$$\min_{z \in \mathbb{C}^n \setminus \{0\}} \{D(F_0; z)\}.$$  

We consider the case where $n \geq 2$ (we can solve Problem 1 easily when $n = 1$). Let $A$ be a given finite subset of $\mathbb{Z}^n$ such that $\{F_\alpha \mid \alpha \in A\} = \{F_\alpha \mid \alpha \in \mathbb{Q}^n \setminus \{0\}\}$. To compute the maximal perturbation, it is necessary to compute

$$\inf_{z \in (\mathbb{C}^*)^n} \{D(F_\alpha; z)\} = \inf_{z \in (\mathbb{C}^*)^n} \{D((f_1, \ldots, f_n); z)\}. \tag{1}$$

If every polynomial in $F_\alpha$ has a nonzero constant term, $D(F_\alpha; z)$ is a continuous function of $z \in \mathbb{C}^n$ and we can prove that $\inf_{z \in (\mathbb{C}^*)^n} \{D(F_\alpha; z)\} = \inf_{z \in \mathbb{C}^n} \{D(F_\alpha; z)\}$. This situation might be useful for numerical computation. Furthermore, non-existence of solutions in $\mathbb{C}^n$ is equivalent to that the Gröbner basis contains 1. Along this line, Problem 1 could be solved by something like a “perturbed” Gröbner basis or an “approximate” Gröbner basis, that is, finding the nearest polynomial system whose Gröbner basis does not contain 1 for a given polynomial system whose Gröbner basis contains 1.

When there exists a polynomial in $F_\alpha$ without a constant term, $D(F_\alpha; z)$ is discontinuous in general so that we cannot take the above approach. Instead of (1), we could consider

$$\inf_{z \in \mathbb{C}^{n+1}} \{D((f_1, \ldots, f_n, x_1 \ldots x_n y - c); z)\}, \tag{2}$$

where $y$ is a new variable and $c \in \mathbb{R}$ with a sufficiently large absolute value that can be computed easily because the values of (1) and (2) are equal. However, in this case, $D((f_1, \ldots, f_n, x_1 \ldots x_n y - c); z)$ is discontinuous in general so that it might not be useful for numerical computation.

Thus, we will take another approach. Instead of adding a new variable, by dividing each polynomial in $F_\alpha$ by an appropriate monomial and by changing variables, we have a new polynomial system $G$ such that every polynomial in $G$ has a nonzero constant term and the number of solutions in $(\mathbb{C}^*)^n$ and the maximal perturbation are unchanged.

Example 1 Let $F$ be $(x_1+x_2^2+x_3^2, x_1 x_2+x_1 x_3+x_2 x_3, x_1 x_2 x_3+1)$. First, divide the first, second, and third polynomials by $x_3^3$, $x_2 x_3$, and 1 (the least monomials of each polynomial with respect to the lexicographic order with $x_1 > x_2 > x_3$), respectively and we obtain $(x_1 x_3^{-3} + x_2^2 x_3^{-3} + 1, x_1 x_3^{-1} + x_1 x_2^{-1} + 1, x_1 x_2 x_3 + 1)$. Second, by taking $y_1 = x_1 x_2^{-e_{1,2} x_3^{-e_{1,3}}}, y_2 = x_2 x_3^{-e_{2,3}}$, and $y_3 = x_3 (e_{i,j} \in \mathbb{N}_0)$, we want to obtain polynomials in $y_1$ with nonzero constant terms. In this example, by taking $y_1 = x_1 x_2^{-1} x_3^{-3}, y_2 = x_2 x_3^{-2},$ and $y_3 = x_3$, we obtain a desired new system $(y_1 y_2 x_3 + y_2^2 y_3 + 1, y_1 y_2 y_3^4 + y_1 y_3^3 + 1, y_1 y_2 y_3^8 + 1)$.

Finally, we mention a remark on a bivariate system. To compute the maximal perturbation, it is sufficient to compute $\inf_{z \in \mathbb{C}^2} \{D((g_1, g_2); z)\}$ for a finite number of pairs $(g_1, g_2)$, where $g_1, g_2 \in \mathbb{C}[x_1, x_2]$ with nonzero constant terms. Since $\text{supp}(g_1) \cup \text{supp}(g_2)$ lie on a line in $\mathbb{R}^2$, we can reduce the computation for $\inf_{z \in \mathbb{C}^2} \{D((g_1, g_2); z)\}$ to that for $\inf_{z \in \mathbb{C}} \{D((h_1, h_2); z)\}$, where $h_1, h_2 \in \mathbb{C}[y_1]$, by changing variables. For $p_1, p_2 \in \mathbb{C}[y_1]$, $p_1$ and $p_2$ has a common zero if and only if $\text{gcd}(p_1, p_2)$ is nontrivial. Thus, we can use the results on “approximate GCD” that has been extensively studied to find the nearest $q_1, q_2 \in \mathbb{C}[y_1]$ for given $p_1, p_2 \in \mathbb{C}[y_1]$ such that $\text{gcd}(q_1, q_2)$ is nontrivial.

References


Variable projection methods for approximate GCD computations

Konstantin Usevich and Ivan Markovsky
School of Electronics and Computer Science
University of Southampton
Southampton, SO17 1BJ, UK
{im, kdu}@ecs.soton.ac.uk

Abstract

This paper presents optimization methods and software for the approximate GCD problem of multiple univariate polynomials in the weighted 2-norm. Backward error minimization and Sylvester low-rank approximation formulations of the problem are solved by the variable projection method. Optimization methods are implemented in publicly available C++ software package with an interface to MATLAB. Results on computational complexity are presented.

1 Introduction

Approximate GCD computations is a rapidly developing area of symbolic-numeric computations [5]. The basic approximate common divisor (ACD) problem is to find the smallest perturbation of given polynomials such that the perturbed polynomials have common divisor of fixed degree $d$. If tentative GCD degree $d$ is not known ($\varepsilon$-GCD problem [2]), then the approximate GCD can be found by successively solving ACD problems for different degrees.

The ACD problem is a nonconvex optimization problem and is usually solved in two steps: (a) obtain initial approximation by some heuristic and (b) perform local optimization. A lot of methods for computing initial approximation have been developed. In addition, step (a) is usually combined with model order selection (selection of ACD degree $d$). In this paper we present efficient methods for local optimization step that have $O(n)$ complexity if the GCD degree is small, hence they are preferable to $O(n^2)$ methods based on structured total least norm [2].

Let $\mathbb{P}_n$ denote the space of homogeneous polynomials in $\mathbb{F}[x, y]$ (where $\mathbb{F} = \mathbb{R}, \mathbb{C}$) of degree $n$ (including 0). Given $N$ polynomials $p^{(1)} \in \mathbb{P}_{n_1}, \ldots, p^{(N)} \in \mathbb{P}_{n_N}$ and $d \geq 1$ the following equivalent formulations are available.

Problem 1 (Backward error minimization)

$$\minimize \sum_{k=1}^{N} \|p^{(k)} - \hat{g}^{(k)}\hat{h}\|_w^2 \quad \text{over} \quad \hat{g}^{(1)} \in \mathbb{P}_{n_1-d}, \ldots, \hat{g}^{(N)} \in \mathbb{P}_{n_N-d}, \hat{h} \in \mathbb{P}_d \setminus \{0\}$$
Problem 2 (Sylvester-structured low-rank approximation)

\[
\text{minimize } \sum_{k=1}^{N} \|p^{(k)} - \hat{p}^{(k)}\|_w^2 \quad \text{over} \quad \hat{p}^{(1)} \in \mathbb{P}_{n_1}, \ldots, \hat{p}^{(N)} \in \mathbb{P}_{n_N}
\]

subject to \( S_d(\hat{p}^{(1)}, \ldots, \hat{p}^{(N)}) \) is rank deficient,

where \( S_d \) is the multipolynomial generalized Sylvester matrix [7].

For initial approximation mainly Problem 2 is used, solved suboptimally by unstructured low-rank approximation.

### 1.1 Variable projection methods

Problem 1 for fixed \( \hat{h} \) is a linear least squares problem with a closed-form solution

\[
f(\hat{h}) = \sum_{k=1}^{N} \|p^{(k)}\|_w^2 - s^\top(\hat{h}) \Gamma^{-1}(\hat{h}) s(\hat{h}),
\]

where \( s^\top(\hat{h}) \Gamma^{-1}(\hat{h}) s(\hat{h}) \) is the sum of squared norms of projections of \( p^{(k)} \) on the space of polynomials with common divisor \( \hat{h} \). Hence Problem 1 can be solved by minimizing \( f(\hat{h}) \) over \( \hat{h} \in \mathbb{P}_d \setminus \{0\} \) (a variable projection principle [3, Ch. 9] applied to a nonlinear least squares Problem 1).

Minimization of \( f(\hat{h}) \) is usually solved by global optimization methods: subdivision methods [4], SDP relaxations [6]. For local optimization the original formulation (Problem 1) or Sylvester low-rank approximation formulation are mainly used. Although evaluation of \( f(\hat{h}) \) is characterised as “expensive” in [5], we develop an efficient method of \( f(\hat{h}) \) evaluation for ACD of \( N \) polynomials in \( \|\cdot\|_w \)-norm, which is an extension of the method presented in [9] for two polynomials and 2-norm.

The term \( s^\top(\hat{h}) \Gamma^{-1}(\hat{h}) s(\hat{h}) \) is the cost function for the weighted structured Hankel low-rank approximation, and therefore the related results on complexity and structure of the problem [8, 9] can be applied. If \( d \ll n_k \), then the \( \Gamma \) matrix is a \( d \)-banded matrix, and therefore its Cholesky factorization can be computed in \( O(d^2 n) \) flops, where \( n := \sum_{k=1}^{N} n_k \). If the weights are uniform (the 2-norm case), then \( \Gamma \) is block-diagonal with Toeplitz banded blocks and the complexity of Cholesky factorization is \( O(dn) \). It can be shown [8] that the cost function and gradient evaluation has the same computational complexity as the Cholesky factorization, both for weighted and unweighted cases.

Problem 2 can be solved by applying a similar principle [8] and turning Problem 2 into optimization problem over the set of linear relations on rows of \( S_d \), i.e. over the set of polynomials \( u^{(k)} \in \mathbb{P}_{n_k - d} \) such that

\[
u^{(1)} p^{(k)} + u^{(k)} p^{(1)} = 0, \quad k = 2, \ldots, N.
\]

The complexity of the cost function and gradient evaluation in this case is \( O((\sum_{k=1}^{N} (n_k - d))^2 n) \) (preferable if \( n_k \approx d \)).
1.2 Software implementation and future work

Variable projection algorithms for Problems 1 and 2 are implemented efficiently in C++ package for structured low-rank approximation [1]. To convert between SLRA problem parameters and ACD problem parameters, a MATLAB interface is available.

Variable projection can be applied to complex and vector polynomials, and it would be interesting to investigate these cases. Another possible direction of future work would be to design efficient methods for “medium” $d$.

References


An Efficient Algorithm to Factorize Sparse Bivariate Polynomials over the Rationals *

Wenyuan Wu¹, Jingwei Chen², Yong Feng¹
¹Chongqing Institute of Green and Intelligent Technology, CAS, China
²Chengdu Institute of Computer Applications, CAS, China
wuwenyuan@cigit.ac.cn, velen.chan163.com, yongfeng@cigit.ac.cn

Polynomial factorization is one of the central problems and also a successful story in computer algebra. In this poster, we give a summary of an algorithm, recently presented in the authors’ manuscript [5], that uses both symbolic and numeric methods to exactly compute the irreducible factorization in \( \mathbb{Z}[x,y] \), and therefore in \( \mathbb{Q}[x,y] \), of any bivariate polynomial satisfying

**Hypothesis.** On \( f \in \mathbb{Z}[x,y] \) and its initial factors, we assume that (i) \( f \) is squarefree, non-constant and monic in \( x \); (ii) \( f \) has no univariate factors; (iii) the initial factors of \( f \) are mutually coprime.

There are a large number of algorithms to factorize sparse polynomials, such as probabilistic algorithm [6], supersparse (lacunary) algorithm [3], polytope method [1, 4, 2], etc. Our algorithm can be seen as a polytope method and is based on the generalized Hensel lifting, which preserves the sparsity, and a numerical combination before lifting.

We now sketch the algorithm. To factorize \( f \in \mathbb{Z}[x,y] \) satisfying the hypothesis, firstly we factorize its Newton polynomial which essentially is a univariate polynomial; secondly we combine these factors to the right initial factors using a numerical method; lastly we lift the initial factors to the irreducible factors of \( f \) over \( \mathbb{Z} \) without usual expression swell.

Figure 1: The polytope and the Newton line

Figure 2: Lifting for \( G_2 \)

Note that it is different from classical methods that we perform the combination step before the lifting step. We give a brief example here. Let \( f \) have two irreducible factors \( G_1 = x^4 + 2x^2y + y^2 + 2y^3 \) and \( G_2 = x^4 - 4y^2 + 5y^5 \). We first factorize its Newton polynomial \( f^{(0)} = -4y^4 - 8x^2y^3 + 2x^6y - 3x^4y^2 + x^8 \) (Each term of \( f^{(0)} \) lies on the Newton line, i.e. \( LR \) in Fig. 1.) in \( \mathbb{Z}[x,y] \) as \( g_1 \cdot g_2 \cdot g_3 \triangleq (x^2 + y^2)^2 \cdot (x^2 + 2y^2) \cdot (x^2 - 2\hat{y}^2) \), where \( \hat{y} = y^{1/2} \). However, the initial

---

*This work was partially supported by NKBRPC (2011CB302400) and NSFC (11001040, 11171053).
factors here are $G_1^{(0)} = x^4 + 2x^2y + y^2$ and $G_2^{(0)} = x^4 - 4y^2$, respectively. Thus there exists a unique vector $\mu_i = (\mu_{j,i}) \in \{0,1\}^3$ such that $G_i^{(0)} = \prod_{j=1}^3 g_j^{\mu_{j,i}}$ for $i = 1, 2$, for instance $G_2^{(0)} = g_0^0 \cdot g_1^1 \cdot g_3^3$.

After linearization by taking natural logarithm, we have $\ln G_i^{(0)} = \sum_{j=1}^3 \mu_{j,i} \ln g_j$. By uniformly random choosing some evaluation points, we can construct a numerical matrix equation $AU = B$, from which we get all $\mu_{j,i}$’s by rounding the solution $U$. Then we use the generalized Hensel lifting to lift each initial factor by $y^{1/2}$ in each step until the factorization completes (see Fig. 2).

We prove that the combination is correct under some error control conditions and that the generalized Hensel lifting always returns the irreducible factors of $f$ in $\mathbb{Z}[x, y]$ provided that its initial factors are given. Moreover, the sparsity of the input polynomial is preserved during lifting.

From the complexity point of view, given a bivariate polynomial $f$ in $\mathbb{Q}[x, y]$ satisfying the hypothesis, our algorithm reduces the computation of the irreducible factors of $f$ over $\mathbb{Q}$ to the computation of univariate polynomials factorization with degree at most $d_x$ over $\mathbb{Q}$ in $O(Td_xd_y + s\omega)$ arithmetic operations, where $T$ is the number of non-zero terms of $f$, $d_x$ and $d_y$ are the degrees in $x$ and $y$ respectively, $s$ is the number of the irreducible factors in $\mathbb{Q}[x, \hat{y}]$ of the Newton polynomial of $f$, and $\omega$ is the exponent of matrix multiplication. Note that our algorithm is probabilistic.

In Table 1, we compare some experimental results for the Maple’s built-in function factor and our implementation BiFactor of our algorithm in Maple 14, where all univariate factorizations are directly computed by factor. All tests were run on Athlon 7750 processor (2.70 GHz) with 2GB memory. Time is shown in seconds. All test polynomials are of total degree $d$ and constructed by multiplying two random polynomials with total degree $d/2$.

<table>
<thead>
<tr>
<th>$d$</th>
<th>$d_x$</th>
<th>$d_y$</th>
<th>$T$</th>
<th>Time Maple</th>
<th>Time BiFactor</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>50</td>
<td>50</td>
<td>130</td>
<td>0.281</td>
<td>0.157</td>
</tr>
<tr>
<td>100</td>
<td>95</td>
<td>89</td>
<td>143</td>
<td>2.125</td>
<td>0.109</td>
</tr>
<tr>
<td>200</td>
<td>156</td>
<td>152</td>
<td>144</td>
<td>7.578</td>
<td>0.235</td>
</tr>
<tr>
<td>400</td>
<td>335</td>
<td>305</td>
<td>144</td>
<td>87.297</td>
<td>0.218</td>
</tr>
<tr>
<td>800</td>
<td>580</td>
<td>565</td>
<td>144</td>
<td>941.994</td>
<td>0.422</td>
</tr>
<tr>
<td>90</td>
<td>39</td>
<td>50</td>
<td>705</td>
<td>0.532</td>
<td>0.437</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>65</td>
<td>827</td>
<td>4.282</td>
<td>1.468</td>
</tr>
<tr>
<td>200</td>
<td>193</td>
<td>179</td>
<td>890</td>
<td>50.75</td>
<td>2.156</td>
</tr>
<tr>
<td>400</td>
<td>306</td>
<td>302</td>
<td>746</td>
<td>208.515</td>
<td>2.110</td>
</tr>
<tr>
<td>800</td>
<td>784</td>
<td>613</td>
<td>912</td>
<td>10301.844</td>
<td>4.578</td>
</tr>
</tbody>
</table>

Table 1: Experimental results

Our future work is to weaken the hypothesis and to generalize the method to polynomials with variables more than two.

References


