ACM Communications in Computer Algebra, Vol.xx, No.xx, Issue xxx, month 201x

Computing generic bivariate Gröbner bases with Mathemagix

Robin Larrieu

Laboratoire d'informatique de l'École polytechnique
LIX, UMR 7161 CNRS
1, rue Honoré d'Estienne d'Orves
Bâtiment Alan Turing, CS35003
91120 Palaiseau, France
`larrieu@lix.polytechnique.fr`

Abstract

Let $A, B \in \mathbb{K}[X, Y]$ be two bivariate polynomials over an effective field $\mathbb{K}$, and let G be the reduced Gröbner basis of the ideal $I := \langle A, B\rangle$ generated by A and B with respect to the usual degree lexicographic order. Assuming A and B sufficiently generic, G admits a so-called *concise representation* that helps computing normal forms more efficiently [7]. Actually, given this concise representation, a polynomial $P \in \mathbb{K}[X, Y]$ can be reduced modulo G with quasi-optimal complexity (in terms of the size of the input A, B, P). Moreover, the concise representation can be computed from the input A, B with quasi-optimal complexity as well. The present paper reports on an efficient implementation for these two tasks in the free software Mathemagix [10]. This implementation is included in Mathemagix as a library called larrix.

1 Introduction

In the past year, some attention was drawn to generic bivariate polynomials, whose structure led to several new complexity results:

- on polynomial reduction, by van der Hoeven and Larrieu [8, 7].
- on the resultant, by Villard [12], and later by van der Hoeven and Lecerf [9];

A natural question with such theoretical results is whether they can be used in practical implementations. In this paper, we focus on polynomial reduction, and more specifically on the results from [7]: here input polynomials are generic and dense for the total degree order. In this very particular setting, we show that the new ideas can be turned into competitive software.

Let $A, B \in \mathbb{K}[X, Y]$ be two bivariate polynomials over an effective field $\mathbb{K}$, and $I := \langle A, B\rangle$ be the ideal they generate. If one wants to compute in the quotient algebra $\mathbb{A} := \mathbb{K}[X, Y]/I$, a solution is to first compute a Gröbner basis of I, then one can reduce any polynomial to some normal form (which is unique by definition of a Gröbner basis). References on the computation of such bases are Faugère's algorithms F4 and F5 [4, 3], and his FGb software [5].

A first remark is about the size of these objects. If A and B have degree n, they have $\Theta(n^2)$ coefficients, and the algebra $\mathbb{A}$ has dimension n^2 generically. However, the Gröbner basis G has generically $n+1$ elements $G_0, \ldots, G_n$ with $\Theta(n^2)$ coefficients each. *A priori*, computing Gröbner bases and normal forms requires then at least $\Theta(n^3)$ operations, simply because of the size of equation

$$P = Q_0G_0 + Q_1G_1 + \cdots + Q_nG_n + R. \tag{1}$$

In fact, using a more compact representation of G, a quasi-optimal complexity $\tilde{O}(n^2)$ can be achieved [7].

2 Idea of the algorithm

The purpose of this section is to briefly recall the results from [7] (refer to this for details).

2.1 Key ingredients

The first idea is to use a *dichotomic selection strategy* to control the degrees of the quotients. A selection strategy describes how one chooses against which basis element a given term is reduced. In the dichotomic selection strategy, each monomial is reduced preferably against one end of the Gröbner basis (G_0 or G_n), or the G_i where i has the highest 2-adic valuation. This way, most quotients have a very small degree: roughly speaking, there are $n/2$ quotients of degree d, plus $n/4$ quotients of degree $2d$, plus $n/8$ quotients of degree $4d$, and so on, where d is a constant independent of n.

This allows for a second ingredient that is to keep only sufficiently many leading terms of each G_i. The definition of "sufficiently many" depends on the degree of the quotient Q_i. For example, G_0 and G_n have to be known entirely as Q_0 and Q_n can have very large degree; but on the other hand, for the $n/2$ indices i such that $\deg(Q_i) = d$, it suffices to know the (approximately) nd leading terms of G_i.

Knowing G_i with this precision is sufficient to compute the quotient Q_i, but not the remainder R. The third idea is to keep track of the relations that exist between $G_0, \ldots, G_n$. Using these relations, equation (1) can be symbolically rewritten to use fewer terms, so that the remainder can be evaluated with the expected complexity.

The later two ingredients lead to a so-called *concise representation* for the Gröbner basis. This representation consists of the basis elements truncated to the appropriate precision (ingredient 2), and the collection of some well-chosen relations (ingredient 3). The whole representation requires only $\tilde{O}(n^2)$ space.

2.2 Algorithms

Concise representation. Let us run Buchberger's algorithm [1] from input A, B. Starting from $G_0 := A$ and $G_1 := B \text{ rem } A$, we set $G_{i+2} := \text{Spol}(G_i, G_{i+1}) \text{ rem } (G_0, \ldots, G_{i+1})$. For generic A, B, we have $\text{lm}(G_0) = Y^n$ and $\text{lm}(G_i) = X^{2i-1}Y^{n-i}$ for $i \geqslant 1$. In particular, $\text{lm}(G_n) = X^{2n-1}$, and the Gröbner basis is complete once G_n is reached because of the Bezout bound (there are n^2 monomials under the Gröbner stairs at this point). This procedure can be improved by setting simply $G_{i+2} := \text{Spol}(G_i, G_{i+1}) \text{ rem } (G_i, G_{i+1})$, or in matrix form

$$\begin{pmatrix} G_{i+1} \\ G_{i+2} \end{pmatrix} = M_i \begin{pmatrix} G_i \\ G_{i+1} \end{pmatrix},$$

indeed we notice that the leading monomials are the same with this new formula. It turns out that the matrix M_i depends only on the terms of degree $\deg(G_i)$ of G_i, and on the terms of degree $\deg(G_{i+1})$ of G_{i+1}. More precisely, obtaining $M_0, M_1, \ldots, M_{n-2}$ essentially boils down to a univariate GCD computation for the dominant diagonals of A and B (with $\text{diag}(A) := \sum_i a_{n-i,i} Z^i \in \mathbb{K}[Z]$). Elementary linear algebra then gives matrices $M_{i,k}$ such that

$$\begin{pmatrix} G_{i+k} \\ G_{i+k+1} \end{pmatrix} = M_{i,k} \begin{pmatrix} G_i \\ G_{i+1} \end{pmatrix}.$$

These matrices allow to compute the truncated basis elements $G_i^{\#}$ by decreasing precision: for example if $n = 8$, start with $G_0^{\#}, G_1^{\#}$, then compute $G_8^{\#}$, then $G_4^{\#}, G_5^{\#}$ and finally $G_2^{\#}, G_3^{\#}$ and $G_6^{\#}, G_7^{\#}$. With this algorithm, the concise representation is computed in $\tilde{O}(n^2)$ operations.

Normal forms. Consider first the naive reduction algorithm: start with $Q_0 = \cdots = Q_n = R = 0$; then reduce the leading term of $P - \sum_i Q_i G_i - R$ and update the quotients or remainder accordingly at each step, until $P - \sum_i Q_i G_i - R = 0$. A first improvement is to evaluate the formula $P - \sum_i Q_i G_i - R$ using fast series arithmetic as in [6]; this ensures quasi-linear complexity with respect to the size of equation (1).

To reduce the size of this formula, we use the truncated elements $G_i^{\#}$ from the previous subsection, and we rewrite progressively the equation to maintain sufficient precision. Roughly speaking, as soon as a quotient Q_i is known, the term Q_iG_i is replaced by an equivalent $S_kG_k + S_{k+1}G_{k+1}$ for a well-chosen k. With this algorithm, the normal form of P is computed in $\tilde{O}(n^2 + d^2)$ operations, where $d := \deg(P)$.

3 Experimental results

The above algorithms were implemented in the MATHEMAGIX software [10], whose source code can be downloaded from `svn://scm.gforge.inria.fr/svnroot/mmx/`. The implementation of the bivariate Gröbner basis and normal form algorithms is gathered in the package LARRIX. All timings were measured on a platform equipped with an INTEL(R) CORE(TM) i7-6700 CPU at 3.40 GHz and 32 GB of 2133 MHz DDR4 memory.

We compare the MATHEMAGIX implementation of our algorithms with the equivalent functionalities in FGB [5] (Gröbner basis) and SAGEMATH [11] (Gröbner basis and normal form)[1]. The benchmarks are done using the svn revision 10718 of MATHEMAGIX, FGB/modp version 14538 and SAGEMATH version 8.0; in each case the program uses a single thread.

We run the following experiment (file `bench/ggg_bench.cpp`, or `bench/ggg_FGb_bench.cpp` for FGB, or `bench/ggg_sage_bench.sage` for SAGE):

- pick random bivariate polynomials A, B of degree n with coefficients in the prime field $\mathbb{Z}/65521\mathbb{Z}$. This field is sufficiently large so that random elements satisfy the genericity assumptions with very high probability.
- compute a Gröbner basis G in concise representation with our algorithm and measure the time needed for this task. If the genericity hypothesis is not satisfied, this can be detected at this point and the computation fails (but it does not happen in practice).
- pick a random polynomial P of degree $2n$, compute its normal form with respect to G and measure the time needed for this task.

The results are given in Figure 1 (a solid line represents the Gröbner basis computation, and a dashed line represents the reduction in normal form). We observe that our implementation becomes faster than the others for degrees as low as 20. For larger degrees, the speedup becomes really significant: for $n = 200$, the MATHEMAGIX implementation computes the concise representation in 188ms and the normal form in 1.4s, while FGB and SAGE need around 30s for each task.

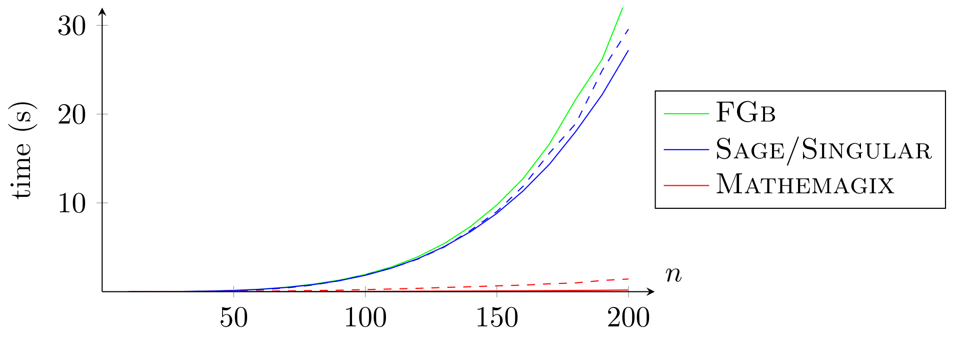

Figure 1: Comparison of the Mathemagix implementation with other software

To find out the most computationally expensive subtasks in our algorithm, we run the experiment for n the order of a few thousand and we measure the time needed for each part. (Let us mention that

[1] Let us mention that SAGEMATH relies on SINGULAR [2] as a backend for multivariate polynomials.

for $n = 1000$, FGB ran for 28 hours before running out of memory.) For the concise representation, we see that computing the truncated basis (matrix-vector products of bivariate polynomials) represent more than 99% of the time. For the normal form, evaluating the formula $P - \sum_i Q_i G_i - R$ using series arithmetic takes about 80-85% of the time, and the substitutions represents 14-18%.

degree n	1000	2000	4000
Truncated basis (s)	10.2	55.6	310
Total concise repr. (s)	10.3	56.0	312
Relaxed products (s)	74.4	447	2603
Substitutions (s)	16.3	83	422
Total normal form (s)	91.9	535	3046

References

[1] Bruno Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenrings nach einem nulldimensionalen Polynomideal.* PhD thesis, Universitat Innsbruck, Austria, 1965.

[2] Wolfram Decker, Gert-Martin Greuel, Gerhard Pfister, and Hans Schönemann. SINGULAR 4-1-0 — A computer algebra system for polynomial computations. `http://www.singular.uni-kl.de`, 2017.

[3] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In *Proceedings of the 2002 international symposium on Symbolic and algebraic computation*, ISSAC '02, pages 75–83, New York, NY, USA, 2002. ACM.

[4] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139(1–3):61 – 88, 1999.

[5] Jean-Charles Faugère. FGb: A Library for Computing Gröbner Bases. In K. Fukuda, J. van der Hoeven, M. Joswig, and N. Takayama, editors, *Mathematical Software - ICMS 2010*, volume 6327 of *Lecture Notes in Computer Science*, pages 84–87, Berlin, Heidelberg, September 2010. Springer Berlin / Heidelberg.

[6] Joris van der Hoeven. On the complexity of polynomial reduction. In I. Kotsireas and E. Martínez-Moro, editors, *Proceedings of Applications of Computer Algebra 2015*, volume 198 of *Springer Proceedings in Mathematics and Statistics*, pages 447–458, Cham, 2015. Springer.

[7] Joris van der Hoeven and Robin Larrieu. Fast Gröbner basis computation and polynomial reduction for generic bivariate ideals. Technical report, HAL, 2018. `http://hal.archives-ouvertes.fr/hal-01770408`, accepted for publication in AAECC.

[8] Joris van der Hoeven and Robin Larrieu. Fast reduction of bivariate polynomials with respect to sufficiently regular gröbner bases. In *Proceedings of the 2018 ACM International Symposium on Symbolic and Algebraic Computation*, ISSAC '18, pages 199–206, New York, NY, USA, 2018. ACM.

[9] Joris van der Hoeven and Grégoire Lecerf. Fast computation of generic bivariate resultants. Technical report, HAL, 2019. `http://hal.archives-ouvertes.fr/hal-02080426`.

[10] Joris van der Hoeven, Grégoire Lecerf, Bernard Mourrain, et al. Mathemagix, from 2002. `http://www.mathemagix.org`.

[11] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 8.0)*, 2017. `https://www.sagemath.org`.

[12] Gilles Villard. On computing the resultant of generic bivariate polynomials. In *Proceedings of the 2018 ACM International Symposium on Symbolic and Algebraic Computation*, ISSAC '18, pages 391–398, New York, NY, USA, 2018. ACM.