# Faster bivariate lexicographic Gröbner bases modulo $x^k$

Xavier DAHAN[1]

[1] *Tohoku university, IEHE, Sendai, Japan*

**Abstract**

Given $t$ bivariate polynomials $f_1, \ldots, f_t \in \mathbb{K}[x, y]$, and an integer $k$ we report a work-in-progress to compute a minimal, not reduced, lexicographic Gröbner basis of the ideal $\langle f_1, \ldots, f_t, x^k \rangle$ in $O^\sim(td^2k)$, where $d$ is an upper bound on the $y$-degree of the $f_i$'s. Using the fast normal form algorithm of Schost & St-Pierre [1], this implies that we can compute its reduced Gröbner basis in $O^\sim(td^2k + s^2dk)$ where $s$ is the number of polynomials contained in the output Gröbner basis. In many instances this improves the algorithm of Schost & St-Pierre [2] based on the Howell matrix normal form that runs in time $O^\sim(td^\omega k)$.

**Keywords**

Gröbner bases, Lexicographic order, Bivariate, Euclidean division

## 1. Introduction

**Background** Lexicographic Gröbner bases (lexGb for short) play a fundamental role when manipulating polynomial systems due to the elimination property that they are endowed with. But the lexicographic order often does not behave well with standard Gröbner bases algorithms [3], whereas the degree reverse lexicographic order has been often observed to behave the best among monomial orders when computing a Gröbner basis. This is grounded in strong theoretical evidences [4, 5]. As a result, a standard strategy to compute a zero-dimensional lexGb consists first in computing a Gröbner basis for the degree reverse lexicographic order with efficient modern algorithms like F4, F5 [6, 7] and then proceed to a change of order algorithm [8] to compute the reduced lexGb.

In the case of two variables only, the situation is different. For $t$ polynomials $f_1, \ldots, f_t \in \mathbb{K}[x, y]$, of maximal degree $d$ in $y$, and total degree $d_{tot}$, Schost and St-Pierre in [2], obtains a running time in $O^\sim(t^\omega d^\omega d_{tot})$ when at least one $f_i$ has for leading monomial $y^d$. As usual $\omega$ is the exponent of matrix multiplication. This algorithm computes the Hermite normal form of a generalized Sylvester matrix of the input polynomials, extending the case $t = 2$ treated by Lazard [9, Section 5].

The same article [2] also considers computing the reduced lexGb modulo $x^k$, that is of the ideal $\langle f_1, \ldots, f_t, x^k \rangle$. The idea is to work in the ring $R = \mathbb{K}[x]/x^k$ and to compute the Howell normal form of a generalized Sylvester matrix of the $f_i \in R[y]$. This Howell normal form is the adaptation of the Hermite normal form for matrices of polynomials with coefficients in $R$. The cost can be made of $O^\sim(td^\omega k)$.

**Result** We consider here the more general moduli $P^k$, for $P$ an irreducible polynomial in $\mathbb{K}[x]$ of degree $d_P$. In this paragraph we let $d_P = 1$ to simplify comparisons. Our algorithm based on Euclidean division works in $O^\sim(td^2k)$ and computes a minimal but not reduced Gröbner basis.

*Schost & St-Pierre's normal form algorithm.* The cost of reducing this minimal lexGb is due to another article of Schost & St-Pierre [1, Section 4] and is better stated in term of the output lexGb $\mathcal{H} = (h_s, \ldots, h_0)$. Assume that $\text{LM}(h_s) \prec \cdots \prec \text{LM}(h_0)$, where $\prec$ stands for the lexicographic

*SCSS 2024: 10th International Symposium on Symbolic Computation in Software Science, August 28–30, 2024, Tokyo, Japan*

EMAIL: xdahan@gmail.com (X. DAHAN)

ORCID: 0000-0001-6042-6132 (X. DAHAN)

order with $x \prec y$. Under our setting we have $h_s = P^k$, and if we let $\deg_x(P) = 1$ like when $P = x$, then $\deg_x(h_s) = k$. Let $\deg_y(h_0) = n_0$ be the largest $y$-degree of the polynomials in $\mathcal{H}$, and let $s + 1 = |\mathcal{H}|$ be the number of polynomials in the output. Then reducing $\mathcal{H}$ costs $O^{\sim}(s^2 n_0 k)$ (see [1, Prop. 4.4 & Prop. 5.1]).

Note that $s \leq \min\{k, d\}$ and $n_0 \leq d$. So we obtain an algorithm that computes the reduced lexGb in $O^{\sim}(td^2k + s^2 n_0 k)$, always within $O^{\sim}(td^2k + s^2 dk)$. This is comparable or better than $O^{\sim}(td^\omega k)$ as soon as $s^2 n_0 = O(td^\omega)$. In the case $t = O(1)$ and $s \approx d = k$, we obtain $O^{\sim}(d^4)$ which is worth than $O^{\sim}(d^{\omega+1})$. But in many cases, the asymptotic complexity is better.

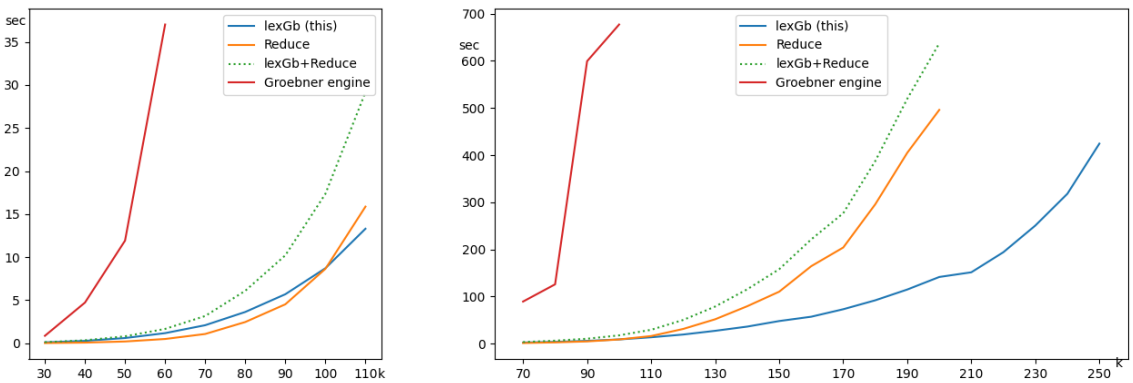
**Implementation** The articles [2, 1] do not mention any implementation. Indeed, implementations of the Howell normal form are apparently seldom and not available publicly — at least in major computer algebra systems. We only found Chapter 4 of the PhD thesis of Storjohann which gives the complexity of $O^{\sim}(d^\omega k)$, see [10, Theorem 4.6], and from which originates the result of [2].

On the other hand, since the presented algorithm that computes a minimal non-reduced lexGb in $O^{\sim}(td^2k)$ can be compared to the quadratic-time standard Euclidean algorithm, it is efficient up to fast univariate arithmetic (polynomial operations involving the $x$-variable only). Our implementation in Magma (available at http://xdahan.sakura.ne.jp/lexgb24.html) supports this claim. As for the normal form algorithm modulo a reduced Gröbner basis of Schost & St-Pierre [1, Section 4], making an implementation efficient is an interesting challenge. For now we resorted here to the internal Magma command "`Reduce`" (in orange).

**Some timings** We tested the algorithm for two input polynomials modulo $x^k$:

$$a_k \equiv \left(\prod_{i=1}^{k}(y + i + x + \cdots + x^i)\right), \quad b_k \equiv (y + 1 + 2x)\left(\prod_{i=2}^{k}(y + i + x + \cdots + x^{i-1} + 2x^i)\right)$$

The reduced lexGb of $\langle a_k, b_k, x^k\rangle$ has $s + 1 = k + 1$ polynomials (the maximum possible) and is dense (it has $O(k^3)$ coefficients). Therefore, this family of examples is suitable for benchmarking as it involves worst case situations: the number of recursive calls is maximal, the cost of the normal form is maximal. Note also that taking only $t = 2$ polynomials as input is not restrictive since recursive calls involve more polynomials in the input. We report below on timings for $k = 30, 40, \ldots, 120$ (left) and $k = 70, 140, \ldots, 250$ (right) over a prime finite field of 64bits. With $t = O(1)$, $k \approx d$, the theoretical cost to obtain a minimal lexGb is $O^{\sim}(k^3)$. The internal command `Reduce` of Magma becomes quickly the bottleneck (in orange. Time $> 500$s for $k > 200$, timings are not displayed). Without surprise, its timing grows faster than the cost of the fast version of Schost & St-Pierre, which is here $O^{\sim}(k^4)$ (it seems to be closer to something in $O^{\sim}(k^5)$). Although we could not compare with the Howell form approach of [2], we could compare with



the internal Gröbner engine of Magma by calling `GroebnerBasis`$([a, b, x^k])$ (red, until $k = 100$). As already reported in [11], the timings are incomparably slow.

**Scope** The motivation behind working modulo $P^k$ is twofold. Firstly, this serves as a skeleton for a similar algorithm that tackles the more general input $\langle f_1, \ldots, f_t, T\rangle$ for an arbitrary polynomial $T \in \mathbb{K}[x]$, not necessarily the power of an irreducible one. See [11], which utilizes dynamic evaluation, for a detailed account when $t = 2$. Secondly, we would like to target the reduced lexGb $\mathcal{H}$ of the general input $\langle f_1, \ldots, f_t\rangle$, not modulo a univariate polynomial, with our Euclidean-division based algorithm. After the work [11], a natural question asks how can we compute the lexGb of two polynomials $f_1, f_2$ from their subresultant sequence? To this end, it is enlightening to access the lexGbs modulo $P_i^{k_i}$, where $P_i^{k_i}$ runs over the primary factors of the elimination polynomial of the $(f_j)_j$'s. The work [12] then permits to understand how these lexGbs can reconstruct $\mathcal{H}$ via Chinese remainders. These remarks lead to a reasonable hope to compute a minimal lexGb faster than the $O^{\sim}(t^{\omega} d^{\omega} d_{tot})$ of [2].

Treating only two variables is clearly limited. Yet, all aspects shall be mastered as there is a cliff in difficulty when considering more than two variables: no general form of Lazard's structural theorem [9] which is key in this work and in [2, 1]. Let us mention though the radical case where some sort of generalizations of Lazard's theorem have been shown [13, 14, 15]. The Euclidean algorithm based approach certainly helps to understand where this difficulty stems from. One aspect of it can be related to the absence of a MonicForm routine (see Eq. (1)) that would transform a nilpotent polynomial, say in $\mathbb{K}[x, y, z]$ modulo a primary ideal in $\mathbb{K}[x, y]$. Think of $f = xz^2 + yz + x + y$, nilpotent modulo the primary $\langle x^2, y^2\rangle \subset \mathbb{K}[x, y]$. It appears that the reduced lexGb of the ideal $\langle f,\ y^2,\ x^2\rangle$ is $[f,\ y^2,\ xy,\ x^2]$. Observe the new polynomial $xy$ introduced with the smaller variables $x$ and $y$. This phenomenon does not appear for two variables only. Therefore, the Euclidean division approach helps to understand better the case of three variables or more.

**Related works** Recently, articles dealing with bivariate Gröbner bases have flourished. A number of them address the question of quasi-optimal asymptotic complexity estimates, with adequate genericity assumptions, and the relation with the resultant [16, 17, 18, 19, 20]. Focusing on non-generic lexGbs, the work [11] from which the present work is inspired, generalizes dynamic evaluation to a non-squarefree modulus. We have already cited [2, 1]. Besides the fast normal form in Section 4, the article [1] introduces a fast Newton iteration for general bivariate lexGbs.

## 2. The algorithm

**Overview** It is based on ideas introduced in [11], which is constrained to two input polynomials $a$ and $b$. Let us summarize the content of the first part of [11] which focuses on working modulo $P^k$ (the second part focuses on working modulo an arbitrary monic univariate polynomial). The divisions occurring in the Euclidean algorithm of $a$ and $b$ modulo $P^k$ require invertible leading coefficients. In the ring $R = \mathbb{K}[x]/\langle P^k\rangle$ elements are either invertible or nilpotent. Weierstrass preparation theorem realized by Hensel lifting permits to circumvent this difficulty, by calculating a "monic form": Given $\tilde{f} \in \mathbb{K}[x, y]$ reduced modulo $P^k$, we denote $f, C_f \leftarrow \text{MonicForm}(\tilde{f}, P^k)$ where:

$$C_f = \gcd(\text{content}(\tilde{f}),\ P^k) \in \mathbb{K}[x], \qquad f \text{ is monic in } y, \qquad \langle \tilde{f},\ P^k\rangle = \langle C_f\, f,\ P^k\rangle. \tag{1}$$

The Euclidean algorithm can be pursued with the monic $f$, and $C_f f$ will be part of the lexGb.

We adapt this strategy to design the main algorithm $\mathcal{H} \leftarrow \text{Add}(f,\ \mathcal{G})$ where $\mathcal{G}$ is a minimal lexGb such that $\mathcal{G} \cap \mathbb{K}[x] = \langle P^{k'}\rangle$ with $k' \leq k$, $f \in \mathbb{K}[x, y]$ and $\mathcal{H}$ is a minimal lexGb of $\langle f\rangle + \langle \mathcal{G}\rangle$. Assuming for the moment this algorithm correct and running in time $O^{\sim}(d^2 k')$, the general algorithm 1 "lexGb" has the following worst-case complexity:

**Algorithm 1:** $\mathcal{G} \leftarrow \text{LEXGB}(f_1, \ldots, f_t; P^k)$

**Input:** Bivariate polynomials $f_1, \ldots, f_t$. Power of an irreducible polynomial $P^k \in \mathbb{K}[x]$.
**Output:** reduced lexGb of $\langle f_1, \ldots, f_t, P^k \rangle$

1 $f_1, \ldots, f_t \leftarrow f_1 \mod P^k, \ldots, f_t \mod P^k$ // $O^\sim(tdd_x)$ or free
2 $\mathcal{G} \leftarrow [P^k]$
3 **for** $i$=1,...,$t$ **do**
4 $\quad \mathcal{G} \leftarrow \text{ADD}(f_i, \mathcal{G})$ // $O^\sim(d^2kd_P)$
5 **return** $\textit{REDUCE}(\mathcal{G})$ // REDUCE based on the normal form of [1, Section 4]. $O^\sim(s^2n_0k)$

**Theorem 1.** *Let $d$ be the maximal degree in $y$ of the polynomials $f_1, \ldots, f_t$. Let $d_x$ their maximal degree in $x$. Let $d_P = \deg_x(P)$. Algorithm 1 computes a minimal lexGb of $\langle f_1, \ldots, f_t, P^k \rangle$ in $O^\sim(td^2kd_P + tdd_x)$.*

*If the input polynomials are reduced modulo $P^k$, or if $P = x$ then the cost is $O^\sim(td^2kd_P)$.*

*The reduced lexGb requires additionally $O^\sim(s^2n_0k)$ operations in $\mathbb{K}$, where $s = |\mathcal{G}|$, $n_0 = \deg_y(g_0)$ is the largest $y$-degree of the polynomials in the output.*

**Algorithm 2:** $\text{ADD}(g, \mathcal{G})$

**Input:** $g \in \mathbb{K}[x, y]$, $\mathcal{G} = [g_0, \ldots, g_s]$ minimal lexGb modulo $P^k = g_s$
**Output:** minimal lexGb of $\langle g \rangle + \langle \mathcal{G} \rangle$

6 **if** $\mathcal{G} ==$ [constant] **then**
7 $\quad$ **return** [1]
8 $f, C_f \leftarrow \text{MONICFORM}(g, P^k)$ // $\langle C_f f, P^k \rangle = \langle g, P^k \rangle$, $f$ monic, $C_f \in \mathbb{K}[x]$
9 **if** $f == 1$ **then**
10 $\quad$ **return** $\text{ADDUNIVARIATE}(C_f, \mathcal{G})$ //Special "easy" case where input polynomial $\in \mathbb{K}[x]$
11 **if** $|\mathcal{G}| == 1$ **then**
12 $\quad$ **return** $[C_f f, P^k]$
13 **return** $\text{ADDGENERIC}(f, C_f, \mathcal{G})$ // Output generates $\langle C_f f \rangle + \langle \mathcal{G} \rangle$

**The main algorithm 2 "Add"** The purpose is given a minimal lexGb $\mathcal{G}$ as above, not necessarily zero-dimensional, and a polynomial $f \in \mathbb{K}[x, y]$ to construct a minimal lexGb of the ideal $\langle f \rangle + \langle \mathcal{G} \rangle$. Thus, it is interesting for its own. It builds upon Euclidean divisions, the key point consists in obtaining a degree (in $y$) decrease through a Euclidean division (see Lines 24 and 34), and then to proceed to adequate recursive calls, with smaller input data (Lines 21 and 23). The algorithm 2 "ADD" actually only treats base cases, and then calls Algorithm 3 "ADDGENERIC", whose input are amenable to recursive calls. One base case is when $f \in \mathbb{K}[x]$ (Line 10) treated apart in the "easy" ADDUNIVARIATE. We omit this short algorithm in this work-in-progress report. Otherwise Algorithm 3 ADDGENERIC, called at Line 13, treats "generic" input: $f$ monic, reduced modulo $P^k$, and $d_f := \deg_y(f) \geq 1$. Its role essentially boils down to managing four cases. Write $\mathcal{G} = [g_0, \ldots, g_s]$ ($\text{LM}(g_s) \prec \cdots \prec \text{LM}(g_0)$, so that $g_s = P^k$ and $\deg_y(g_0) = n_0$).

1. Case distinction: $\ell > k$ or $\ell \leq k$ (equivalently $C_f \nmid g_s = P^k$ or $C_f \mid g_s$)
2. Subcases distinction: $d_f \leq n_0$ or $d_f > n_0$

The first case distinction is treated by renaming variables (if-test at Line 16). The subcase distinction (if-test at Line 20) leads to call two subroutines ADDTWOA and ADDTWOB which looks very similar, but with key differences.

**Algorithms AddTwoA and AddTwoB** The input are monic bivariate polynomials $a, b$, monic univariate polynomials $C_a, C_b$ which are powers of $P$, and a minimal lexGb $\mathcal{G}$ modulo $P^k$.

**Algorithm 3:** AddGeneric($f,\ C_f,\ \mathcal{G}$)

**Input:** $f \in \mathbb{K}[x,y]$ monic $\deg_y(f) \geq 1$, $C_f = P^t \in \mathbb{K}[x]$, $\mathcal{G}$ minimal lexGb modulo $P^k$
**Output:** minimal lexGb of $\langle C_f\, f\rangle + \langle \mathcal{G}\rangle$

14 Let $\mathcal{G} = [g_0, \ldots,\ g_{s-1},\ P^k]$, and write $g_0 = M_0\, g_{0,y}$
15 $C_a \leftarrow \gcd(M_0,\ C_f)$ // $C_a = C_f$ or $C_a = M_0$
16 **if** $C_a == C_f$ **then**
17 $\quad a \leftarrow f,\ b \leftarrow g_{0,y},\ C_b \leftarrow M_0$ // $C_b b = g_0$
18 **else**
19 $\quad b \leftarrow f,\ a \leftarrow g_{0,y},\ C_b \leftarrow C_f$ // $C_a a = g_0$
20 **if** $\deg_y(a) > \deg_y(b)$ **then** // Always holds $\langle C_a\, a,\ C_b\, b,\ P^k\rangle = \langle C_f\, f,\ g_0,\ P^k\rangle$
21 $\quad$ **return** AddTwoA($a,\ b,\ C_a,\ C_b,\ \mathcal{G}_{\geq 1}$) // lexGb of $\langle C_a\, a,\ C_b\, b\rangle + \langle g_1, \ldots, g_s\rangle$
22 **else**
23 $\quad$ **return** AddTwoB($a,\ b,\ C_a,\ C_b,\ \mathcal{G}_{\geq 1}$) // lexGb of $\langle C_a\, a,\ C_b\, b\rangle + \langle g_1, \ldots, g_s\rangle$

Additional degree constraints on $a, b, C_a, C_b$ depend on one or the other algorithm. The output is a minimal lexGb of $\langle C_a\, a,\ C_b\, b\rangle + \langle \mathcal{G}\rangle$ (whence the name "AddTwo"). The key point is the degree decrease obtained by the Euclidean division at Lines 24 and 34. Then they undertake recursive calls. These divisions henceforth imply the complexity stated in Theorem 1.

**Algorithm 4:** AddTwoA($a,\ b,\ C_a,\ C_b,\ \mathcal{G}$)

**Input:** 1. $a, b \in \mathbb{K}[x,y]$ monic, $\deg_y(a) > \deg_y(b)$
2. $C_a, C_b \in \mathbb{K}[x]$ powers of $P$, $C_a \mid C_b \mid P^k$,
3. $\mathcal{G}$ minimal lexGb modulo $P^k$, and $\deg_y(a) > \deg_y(\mathcal{G})$
**Output:** minimal lexGb of $\langle C_a\, a,\ C_b\, b\rangle + \langle \mathcal{G}\rangle$

24 $r \leftarrow a \mod b$ // $b$ monic. Over $R = \mathbb{K}[x]/\langle \frac{P^k}{C_b}\rangle$. It holds $\langle C_b\, a,\ C_b\, b,\ P^k\rangle = \langle C_b\, b,\ C_b\, r,\ P^k\rangle$
25 **if** $r \equiv 0 \mod \frac{g_1}{C_b}$ **then** // Here $\langle C_b\, a,\ C_b\, b,\ P^k\rangle = \langle C_b\, b,\ P^k\rangle$
26 $\quad \mathcal{G}'' \leftarrow$ Add($b,\ \frac{1}{C_b}\mathcal{G}$) // Here $\langle C_b\, \mathcal{G}''\rangle = \langle C_b\, b\rangle + \langle \mathcal{G}\rangle$
27 **else**
28 $\quad \mathcal{G}' \leftarrow$ Add($r,\ \frac{1}{C_b}\mathcal{G}$) // $\langle C_b \mathcal{G}'\rangle = \langle C_b\, r\rangle + \langle \mathcal{G}\rangle$
29 $\quad \mathcal{G}'' \leftarrow$ Add($b,\ \mathcal{G}'$) // $\langle C_b \mathcal{G}''\rangle = \langle C_b\, b\rangle + \langle C_b \mathcal{G}'\rangle = \langle C_b\, b,\ C_b\, r\rangle + \langle \mathcal{G}\rangle = \langle C_b\, b,\ C_b\, a\rangle + \langle \mathcal{G}\rangle$
30 **if** $C_a == C_b$ **then**
31 $\quad$ **return** $C_b \cdot \mathcal{G}''$ // Here $\langle C_b \mathcal{G}''\rangle = \langle C_b\, b,\ C_a\, a\rangle + \langle \mathcal{G}\rangle$
32 **else**
33 $\quad$ **return** $[C_a\, a]$ cat $C_b \cdot \mathcal{G}''$ // output generates $\langle C_a\, a\rangle + \langle C_b \mathcal{G}''\rangle = \langle C_a\, a,\ C_b\, b\rangle + \langle \mathcal{G}\rangle$

**Algorithm 5:** AddTwoB($a,\ b,\ C_a,\ C_b,\ \mathcal{G}$)

**Input:** 1. $a, b \in \mathbb{K}[x,y]$ monic, $\deg_y(a) \leq \deg_y(b)$,
2. $C_a,\ C_b \in \mathbb{K}[x]$ powers of $P$, $C_a \mid C_b \mid P^k$,
3. $\mathcal{G}$ minimal lexGb modulo $P^k$, $\deg_y(b) > \deg_y(\mathcal{G})$
**Output:** minimal lexGb of $\langle C_a\, a,\ C_b\, b\rangle + \langle \mathcal{G}\rangle$

34 $r \leftarrow \frac{C_b}{C_a} b \mod a$ // $a$ monic. Over $R = \mathbb{K}[x]/\langle \frac{P^k}{C_a}\rangle$. It holds $\langle C_a\, r,\ C_a\, a,\ P^k\rangle = \langle C_b\, b,\ C_a\, a,\ P^k\rangle$.
35 **if** $r \equiv 0 \bmod \frac{P^k}{C_a}$ **then** // Here $\langle C_a\, a,\ P^k\rangle = \langle C_a\, a, C_b\, b,\ P^k\rangle$
36 $\quad$ **return** $C_a \cdot$ Add($a, \frac{1}{C_a}\mathcal{G}$) // Output generates $\langle C_a\, a\rangle + \langle \mathcal{G}\rangle$
37 **else**
38 $\quad \mathcal{G}' \leftarrow$ Add($r,\ \frac{1}{C_a}\mathcal{G}$) // $\langle C_a \mathcal{G}'\rangle = \langle C_a\, r\rangle + \langle \mathcal{G}\rangle$ **return** $C_a \cdot$ Add($a,\ \mathcal{G}'$) // Output generates $\langle C_a\, a\rangle + \langle C_a \mathcal{G}'\rangle = \langle C_a\, a,\ C_b\, b\rangle + \langle \mathcal{G}\rangle$

## References


[1] É. Schost, C. St-Pierre, Newton iteration for lexicographic Gröbner bases in two variables, Journal of Algebra 653 (2024) 325–377.

[2] É. Schost, C. St-Pierre, $p$-adic algorithms for bivariate Gröbner bases, in: Proceedings of the 2023 International Symposium on Symbolic and Algebraic Computation, ACM, New York, NY, USA, 2023.

[3] K. Kalorkoti, Counting and Gröbner bases, J. Symbolic Computation 31 (2001) 307–313.

[4] D. Bayer, M. Stillman, A criterion for detecting m-regularity, Inventiones mathematicae 87 (1987) 1–11.

[5] H. Loh, The converse of a theorem by Bayer and Stillman, Advances in Applied Mathematics 80 (2016) 62–69.

[6] J.-C. Faugère, A new efficient algorithm for computing Gröbner bases (F4), Journal of pure and applied algebra 139 (1999) 61–88.

[7] J.-C. 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, 2002, pp. 75–83.

[8] J.-C. Faugère, P. Gianni, D. Lazard, T. Mora, Efficient computation of zero-dimensional Gröbner bases by change of ordering, Journal of Symbolic Computation 16 (1993) 329–344.

[9] D. Lazard, Ideal bases and primary decomposition: case of two variables, Journal Symbolic Computation 1 (1985) 261–270.

[10] A. Storjohann, Algorithms for matrix canonical forms, Ph.D. thesis, ETH Zürich, 2000.

[11] X. Dahan, Lexicographic Gröbner bases of bivariate polynomials modulo a univariate one, Journal of Symbolic Computation 110 (2022) 24–65.

[12] X. Dahan, Chinese remainder theorem for bivariate lexicographic Gröbner bases, in: Proceedings of the 2023 ACM International Symposium on Symbolic and Algebraic Computation, ISSAC '23, ACM press, New York, NY, USA, 2023.

[13] M. Lederer, The vanishing ideal of a finite set of closed points in affine space, J. of Pure and Applied Algebra 212 (2008) 1116–1133.

[14] M. Marinari, T. Mora, A remark on a remark by Macaulay or enhancing Lazard structural theorem, Bull. Iranian Math. Soc. 29 (2003) 1–45, 85.

[15] B. Felszeghy, B. Ráth, L. Rónyai, The lex game and some applications, Journal of Symbolic Computation 41 (2006) 663 – 681.

[16] G. Villard, On computing the resultant of generic bivariate polynomials, in: Proceedings of the 2018 ACM International Symposium on Symbolic and Algebraic Computation, ISSAC '18, ACM, 2018, p. 391–398.

[17] G. Villard, Elimination ideal and bivariate resultant over finite fields, in: Proceedings of the 2023 ACM International Symposium on Symbolic and Algebraic Computation, ISSAC '23, ACM press, New York, NY, USA, 2023.

[18] J. van der Hoeven, R. 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, ACM, New York, NY, USA, 2018, pp. 199–206.

[19] J. van der Hoeven, R. Larrieu, Fast Gröbner basis and polynomial reduction for generic bivariate ideal, Applicable Algebra in Engineering, Communication and Computing 30 (2019) 509–539.

[20] J. van der Hoeven, G. Lecerf, Fast computation of generic bivariate resultants, Journal of Complexity 62 (2021) 101499.