msolve: A Library for Solving Polynomial Systems

Jérémy Berthomieu, Christian Eder, Mohab Safey El Din

msolve: A Library for Solving Polynomial Systems

Jérémy Berthomieu
Sorbonne Université, CNRS, LIP6
F-75005 Paris, France
jeremy.berthomieu@lip6.fr

Christian Eder
Technische Universität
Kaiserslautern
Kaiserslautern, Germany
ederc@mathematik.uni-kl.de

Mohab Safey El Din
Sorbonne Université, CNRS, LIP6
F-75005 Paris, France
mohab.safey@lip6.fr

ABSTRACT

We present a new open source C library msolve dedicated to solving multivariate polynomial systems of dimension zero through computer algebra methods. The core algorithmic framework of msolve relies on Gröbner bases and linear algebra based algorithms for polynomial system solving. It relies on Gröbner basis computation w.r.t. the degree reverse lexicographical order, Gröbner conversion to a lexicographical Gröbner basis and real solving of univariate polynomials. We explain in detail how these three main steps of the solving process are implemented, how we exploit AVX2 instruction processors and the more general implementation ideas we put into practice to better exploit the computational capabilities of this algorithmic framework. We compare the practical performances of msolve with leading computer algebra systems such as Magma, Maple, Singular on a wide range of systems with finitely many complex solutions, showing that msolve can tackle systems which were out of reach by the computer algebra software state-of-the-art.

ACM Reference Format:
Jérémy Berthomieu, Christian Eder, and Mohab Safey El Din. 2021. msolve: A Library for Solving Polynomial Systems. In *Proceedings of the 2021 International Symposium on Symbolic and Algebraic Computation (ISSAC '21), July 18–23, 2021, Virtual Event, Russian Federation.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3452143.3465545

1 INTRODUCTION

Problem statements and motivation. Polynomial systems arise in a wide range of areas of scientific engineering and computing sciences. Classical problems are to decide if the solution set is finite (over an algebraic closure of the ground field), compute its dimension when it is not, else count the solutions and isolate them over the real or complex numbers when the ground field is infinite.

We design a software library, for solving multivariate polynomial systems, with a focus on those which have dimension at most 0, i.e. finitely many solutions in an algebraic closure of the ground field. We rely on computer algebra methods yielding algebraic parametrizations of their solutions. This allows us to bypass the commonly encountered issues related to accuracy and exhaustivity met by numerical methods because of the non-linearity of the input.

Prior works and state-of-the-art. In this context, one can mention *regular chains* whose base operation is computing gcd of polynomials with coefficients encoded by an algebraic tower of extensions combined with splitting polynomial ideal techniques [5], *geometric resolutions* which is based on an incremental procedure intersecting a lifted curve (obtained by Hensel lifting generic solutions to the first i polynomials) with the hypersurface defined by the $(i+1)$st polynomial [24] and *Gröbner bases* which consist in a set of polynomials in the ideal generated by the input such that for a given monomial order one can use them to define an intrinsic multivariate division and thus decide the ideal membership problem.

In msolve, we focus on Gröbner bases because of their importance in computer algebra systems and their use in many higher-level algorithms. Note that when the input system generates a radical ideal, of dimension at most 0, and in generic coordinates, a Gröbner basis for a lexicographical order on the monomials is in a so-called *shape position*, i.e. it has the following shape:

$$w(x_n), x_{n-1} + u_{n-1}(x_n), \ldots, x_1 + u_1(x_n). \qquad (1)$$

One can then recover the coordinates of all solutions by evaluating univariate polynomials at the roots of a univariate polynomial. Up to normalization, this is very close to a rational parametrization

$$w(x_n), w'(x_n)x_{n-1} + v_{n-1}(x_n), \ldots, w'(x_n)x_1 + v_1(x_n) \qquad (2)$$

where w' is the derivative of w. Such representations of the solution set go back to Kronecker and appear in many works (see e.g. [3, 30]) and, under the above assumptions, are computed by regular chains and geometric resolution algorithms.

Further, we mean by solving a polynomial system of dimension at most 0 the computation of such a rational parametrization of its solution set. Note that such a parametrization exists only when all distinct solutions have distinct x_n-coordinate, which one can always ensure through some linear change of coordinates. Also, when the input coefficients are rational numbers, one includes in the requested output the isolation of the real solutions to w.

Several libraries for computing Gröbner bases can be found, most of them being either tailored for crypto applications (see e.g. [2]) or are designed for algorithmic experimentation (see e.g. [36]). Recently, maple and magma have greatly improved their Gröbner bases engines, the one in maple being based several years ago on FGb [19], which is developed by J.-Ch. Faugère.

The first and third authors are supported by the joint ANR-FWF ANR-19-CE48-0015 ECARP project, the ANR grants ANR-18-CE33-0011 Sesame and ANR-19-CE40-0018 De Rerum Natura projects, the PGMO grant CAMiSAdo and the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement N. 813211 (POEMA). The second author is supported by the Forschungsinitiative Rheinland-Pfalz.

ISSAC '21, July 18–23, 2021, Virtual Event, Russian Federation

ACM ISBN 978-1-4503-8382-0/21/07…$15.00
https://doi.org/10.1145/3452143.3465545

Main results. This is a software paper and hence does not contain any new algorithm or theorem. The main outcome of this work is a software library, written in plain C, *open source*, distributed under the license GPLv2 which includes modern implementations of algorithms for solving multivariate polynomial systems based on Gröbner bases. It supports polynomial systems with coefficients in a prime field of characteristic $< 2^{31}$ or with rational coefficients. It allows one to solve zero-dimensional systems which are out of reach for leading computer algebra systems like magma and maple.

For instance, msolve is able to solve polynomial systems with thousands of complex solutions such as Katsura-14, which has $8,192$ complex solutions, whose solution set is encoded by a rational parametrization of bit size $\simeq 2^{32.37}$, sequentially, within 15 days on an Intel® Xeon® CPU E7-4820 v4 @ 2.00GHz while maple and magma could not solve it after 6 months.

The msolve library is available at: https://msolve.lip6.fr

It includes efficient implementations of the F4 algorithm [18] (reducing Gröbner bases computations to Gaussian elimination), of a change of orders algorithm due to Faugère and Mou [21] (based on computing minimal polynomials of linear endomorphisms) and a dedicated univariate real root solver. Solving systems with rational coefficients is handled through multi-modular computations.

We design and use dedicated data structures to take into account current hardware architectures. Our F4 implementation enjoys several implementations of linear algebra with dedicated storage to handle sparsity structures arising naturally in this algorithm as well as hashing tables and masks for encoding exponent vectors and divisibility checks between monomials.

Our implementation of change of orders is designed for the cases where the *radical* of the ideal generated by the input equations admits a Gröbner basis for a lexicographical order which is in shape position and under an extra assumption which is recovered by replacing the last variable by a generic enough linear combination of the input variables. Hence, msolve loses information on the multiplicities of the solutions but focuses on solving. It includes an efficient routine verifying the correctness of the result when the input ideal is not radical which was missing in the literature.

This implementation uses a dedicated storage of the matrix encoding the linear endomorphism for which one needs to compute the minimal polynomial. It exploits the structure of this matrix to reduce this computation to scalar products of dense vectors.

This allows us to use extensively vectorization instructions such as AVX2 to speed up our computations. A more intricate use of AVX2 instructions is also set for the F4 implementation of msolve.

A special care has been brought to memory consumption which is low compared to the one of maple or magma. This is suitable for a trivial multi-threaded scheme for multi-modular computations, almost dividing the runtime by the number of threads.

Structure of the paper. In Section 2, we fix some notation and recall some background. Section 3 gives an overview of the algorithms. Section 4 describes the design of the library and the implementation ideas. Section 5 reports on the practical performances of msolve.

2 NOTATIONS AND BACKGROUND

We recall below some basic notions on polynomial rings and Gröbner bases. Let $\mathcal{K}$ be a field; we denote by $\mathcal{P} = \mathcal{K}[x_1, \ldots, x_n]$ the polynomial ring with base field $\mathcal{K}$ and variables $x_1, \ldots, x_n$.

Let $<$ denote a monomial order on $\mathcal{P}$. We consider only *global* monomial orders, i.e. orders for which $x_i > 1$ for all $1 \leq i \leq n$. We mainly consider the global monomial orders $<_{\mathrm{DRL}}$, the degree reverse lexicographical order, and $<_{\mathrm{LEX}}$, the lexicographical order (see e.g. [16, Chap. 2, Sec. 2, Def. 3]). Given a monomial order $<$ we can highlight the maximal terms of elements in $\mathcal{P}$ with respect to $<$: For $f \in \mathcal{P} \setminus \{0\}$, $\mathrm{lt}_<(f)$ is the *lead term*, $\mathrm{lm}_<(f)$ the *lead monomial*, and $\mathrm{lc}_<(f)$ the *lead coefficient* of f. For any set $F \subset \mathcal{P}$ we define the *lead ideal* $L_<(F) = \langle \mathrm{lt}_<(f) \mid f \in F \rangle$; for an ideal $I \subset \mathcal{P}$, $L_<(I)$ is defined as the ideal generated by lead terms of all elements of I. Further, we omit the index $<$ when it is clear from the context.

Definition 1. *A finite set $G \subset \mathcal{P}$ is called a* Gröbner basis *for an ideal $I \subset \mathcal{P}$ w.r.t. a monomial order $<$ if $G \subset I$ and $L_<(G) = L_<(I)$. This is equivalent to the condition that for any $f \in I \setminus \{0\}$ there exists a $g \in G$ such that $\mathrm{lm}_<(g) \mid \mathrm{lm}_<(f)$.*

Buchberger gave in 1965 [12] an algorithmic criterion for computing Gröbner bases based on the definition of S-pairs:

Definition 2. *Let $f, g \in \mathcal{P}$ be nonzero, let $G \subset \mathcal{P}$ be finite.*

(1) *Denote by $\lambda := \mathrm{lcm}(\mathrm{lm}_<(f), \mathrm{lm}_<(g))$. The* S-pair *between f and g is given by*
$$\mathrm{sp}(f, g) := \frac{\lambda}{\mathrm{lt}_<(f)} f - \frac{\lambda}{\mathrm{lt}_<(g)} g.$$

(2) *We say that g is a reducer of f if for a term t in f there exists a term $\sigma \in \mathcal{P}$ such that $\mathrm{lt}_<(\sigma g) = t$. The* reduction *of f by g is then given by $f - \sigma g$. We say that* f reduces to h w.r.t. G and $<$ *if there exist finitely many reducers $g_1, \ldots, g_k$ in G such that $h = f - \sum_{i=1}^{k} \sigma_i g_i$ and there exists no term in h for which an element of G is a reducer.*

Theorem 3 (Buchberger's criterion). *A finite set $G \subset \mathcal{P}$ is called a* Gröbner basis *for an ideal $I \subset \mathcal{P}$ w.r.t. a monomial order $<$ if for all $f, g \in G$ $\mathrm{sp}(f, g)$ reduces to zero w.r.t. G.*

Gröbner bases for an ideal I w.r.t. a monomial order $<$ are not unique, but *reduced* Gröbner bases are (these are Gröbner bases where the lead coefficient of each element is 1 and where no monomial of an element $g \in G$ lies in $L_<(G \setminus \{g\})$).

Recall that Gröbner bases allow one to define a *normal form*, i.e. given $f \in \mathcal{P}$, $F \subset \mathcal{P}$ and a Gröbner basis G of $\langle F \rangle$ for a monomial order $<$, one can compute a unique representative of f in the quotient ring $\frac{\mathcal{P}}{\langle F \rangle}$ which is a $\mathcal{K}$-vector space. This property of a Gröbner basis allows one to discover *relations*, i.e. polynomials which lie in $\langle F \rangle$ and the Hilbert series/polynomial associated to $\langle F \rangle$ (see [16, Chap. 10, Sec. 2]) when using the $<_{\mathrm{DRL}}$ order, from which we deduce the Krull dimension and the degree of $\langle F \rangle$. Recall that the Krull dimension of $\langle F \rangle$ coincides with the dimension of its associated algebraic set $V(F)$ in $\overline{\mathcal{K}}^n$, i.e. the largest integer d such that the intersection of $V(F)$ with d hyperplanes in $\overline{\mathcal{K}}^n$ is finite and of maximal cardinality. This cardinality is the degree of $\langle F \rangle$ when it is radical (i.e. $f^k \in \langle F \rangle$ for some k implies that $f \in \langle F \rangle$). We refer

to [17, Chapter 12] for more details on the equivalence of the various definitions of dimension and [17, Section 1.9] for the relations between Hilbert series and degree of ideals and varieties.

Elimination orders such as $<_{\mathrm{LEX}}$ allow one to compute a basis for $\langle F\rangle \cap \mathcal{K}[x_i,\ldots,x_n]$ for $1 \le i \le n$ and then put into practice the Elimination theorem [16, Chapter 3] by yielding an algebraic description of the Zariski closure of the projection of $V(F)$ on the $(x_i,\ldots,x_n)$-subspace. It turns out that Gröbner bases w.r.t. $<_{\mathrm{LEX}}$ order enjoy a triangular structure from which one can extract information on the solution set.

Other geometric operations are encoded with ideal theoretic operations such as set difference whose algebraic counterpart is *saturation* (given an ideal $I \subset \mathcal{P}$ and $g \in \mathcal{P}$, the saturation of I by $\langle g\rangle$, denoted by $I : g^\infty$, is the set of polynomials h such that $hg^k \in I$ for some $k \in \mathbb{N}$).

When the set of solutions of F in $\overline{\mathcal{K}}^n$ is finite, the quotient ring $\frac{\mathcal{P}}{\langle F\rangle}$ is a finite dimensional vector space [16, Chap. 5, Sec. 2, Prop. 7]. The dimension of this vector space coincides with the *degree* of the ideal $\langle F\rangle$: this is the number of solutions counted with multiplicities.

This property is at the foundations of algorithms based on Gröbner bases for solving polynomial systems. It implies that $\langle F\rangle$ has a non-zero intersection with $\mathcal{K}[x_i]$ for $1 \le i \le n$. It is heavily exploited in change of orders algorithms computing a Gröbner basis for a given ideal $\langle F\rangle$, in particular to compute rational parametrizations as they are defined in Section 1. Such algorithms are important since computing Gröbner bases w.r.t. $<_{\mathrm{LEX}}$ order is usually way more expensive than pre-computing a Gröbner basis w.r.t. $<_{\mathrm{DRL}}$ and then applying such a change of orders algorithm [7].

In the following, we assume we are given a finite set of polynomials $F \subset \mathcal{P}$ such that $\langle F\rangle$ is zero-dimensional, that is $F = 0$ has a finite number of solutions in $\overline{\mathcal{K}}^n$. Hence, when F satisfies the Shape position assumption described in Section 1, we apply the following classical solving strategy:

(1) Compute the reduced Gröbner basis G of $\langle F\rangle$ w.r.t. $<_{\mathrm{DRL}}$.
Note that once we have computed G we can decide whether $\langle F\rangle$ is zero-dimensional.
(2) Convert G to the reduced Gröbner basis H of $\sqrt{\langle F\rangle}$ w.r.t. $<_{\mathrm{LEX}}$ and deduce a rational parametrization R encoding its solutions, as in Equation (2).
(3) Apply a univariate solver to the uniquely defined univariate polynomial w in R. Go on by substituting variables already solved.

3 IMPLEMENTED ALGORITHMS

3.1 Faugère's F4 Algorithm

In 1965, Buchberger initiated the theory of Gröbner bases for global monomial orders. Specifically, he introduced some key structural theory, and based on this theory, proposed the first algorithm for computing Gröbner bases [12, 14]. Buchberger's algorithm introduced the concept of critical pairs and S-pairs and repeatedly carries out a certain polynomial operation (called reduction).

msolve includes an implementation of Faugère's F4 algorithm [18] which is variant of Buchberger's. Here we highlight the main differences to Buchberger's algorithm:

• In contrast to Buchberger's algorithm one can choose several S-pairs from the pair set P at a time, for example, all of the same minimal degree. These S-pairs are stored in a subset $L \subset P$.
• Then for all terms of all the generators of the S-pairs in L we search in the current intermediate Gröbner basis G for possible reducers. We add those to L and again search all of their terms for reducers in G.
• Once all available reduction data is collected from the last step, we generate a matrix with columns corresponding to the terms appearing in L and rows corresponding to the coefficients of each polynomial in L. In order to reduce now all chosen S-pairs at once we apply Gaussian Elimination on the matrix and recheck afterwards which rows of the updated matrix give a new leading monomial not already in $L(G)$.

In order to optimize the algorithm one can now apply Buchberger's product and chain criteria, see [13, 29]. With these, useless S-pairs are removed before even added to P thus less zero rows are computed during the linear algebra part of F4. Still, for bigger examples there are many zero reductions.

It is known that Buchberger-like algorithms for computing Gröbner bases, as F4, have a worst-case time complexity doubly-exponential in the number of solutions of the system for $<_{\mathrm{DRL}}$. Still, in practice these algorithms behave in general way better.

3.2 Gröbner conversion Algorithm

We present the variant of the fglm algorithm [20] due to Faugère and Mou [21] which is used in msolve. We assume that the input Gröbner basis G is reduced and that it spans a zero-dimensional ideal I of degree D. We also assume that I satisfies two *generic* assumptions, namely assumptions (P_1) and (P_2) defined below.

Assumption (P_1) means that for every monomial m in the monomial basis B of $\frac{\mathcal{P}}{I}$ for $<_{\mathrm{DRL}}$, either mx_n is another monomial in B or it is the leading monomial of a polynomial in G.

Since $\frac{\mathcal{P}}{I}$ is a finite dimensional $\mathcal{K}$-algebra, the multiplication by x_n is a linear map whose associated matrix $\mathcal{M}$ is called the *multiplication matrix of* x_n. Under assumption (P_1), each column of $\mathcal{M}$ is either a column of the identity matrix or can be read from a polynomial in G whose leading term is divisible by x_n.

Assumption (P_2) means that I is in *shape position*, i.e. the reduced Gröbner basis for I for $<_{\mathrm{LEX}}$ is given by $\{g_n(x_n), x_{n-1} + f_{n-1}(x_n), \ldots, x_1 + f_1(x_n)\}$ where g_n has degree D and polynomials $f_1,\ldots,f_{n-1}$, the *parametrizations* of $x_1,\ldots,x_{n-1}$ have degrees at most $D-1$. Furthermore, the radical of I, $\sqrt{I} = \{h \in \mathcal{P} \mid \exists k \in \mathbb{N}, h^k \in I\}$, also satisfies (P_2) and its reduced Gröbner basis for $<_{\mathrm{LEX}}$ is $\{w_n(x_n), x_{n-1}+u_{n-1}(x_n), \ldots, x_1+u_1(x_n)\}$, with w_n the squarefree part of g_n and $\deg u_i < \deg w_n$ for $1 \le i \le n$.

By construction, the minimal polynomial of x_n in $\frac{\mathcal{P}}{I}$ is the same as the minimal polynomial of $\mathcal{M}$ and is of degree at most D. This polynomial is also called the *eliminating polynomial of* x_n and is g_n. Let $V_0 \in \mathcal{K}^D$ be a column-vector chosen at random and for all $1 \le k < 2D$, $V_k^t = V_0^t \mathcal{M}^k$. Assume that the monomial 1 is the first one in the monomial basis B. Then, using Wiedemann's algorithm, g_n is computed by guessing the minimal recurrence relations of the table $(u_k)_{0\le k<2D}$ defined by $u_k = v_{k,1}$. This guessing step is usually done with the Berlekamp–Massey algorithm and its fast variants [11].

Assuming x_i is the jth monomial in B, its parametrization f_i is computed by solving a Hankel system with matrix $(V_{k+\ell,1})_{0\le k,\ell<D}$ and vector $(V_{k,j})_{0\le k<D}$. Then, $u_i = f_i \bmod w_n$.

Whenever the ideal I does not satisfy assumption (P_2), a parametrization of the solutions might still be possible. This is the case if $\sqrt{I}$ satisfies (P_2). In that case, for $1 \leq i \leq n$, assuming x_i is the jth monomial in B, u_i can be computed in a similar fashion using [27, Algorithm 2]. Let $d = \deg g_n < D$ and $h = g_n \sum_{k=0}^{d-1} V_{k,1} x_n^{d-1-k} \operatorname{quo} x_n^d$. Then,

$$u_i = -\left(g_n \sum_{k=0}^{d-1} V_{k,j} x_n^{d-1-k} \operatorname{quo} x_n^d\right) h^{-1} \bmod w_n. \quad (3)$$

This yields the following algorithm in pseudo-code.

Algorithm 1 Sparse `fglm`

Input: $G \in \mathcal{K}[x_1, \ldots, x_n]$ the reduced Gröbner basis for a zero-dimensional ideal of degree D, satisfying assumption (P_1) w.r.t. $<_{DRL}$, such that for all i $x_i \notin L_{<_{DRL}}(G)$, and whose radical satisfies assumption (P_2)

Output: A parametrization of the roots of $\langle G \rangle$.

```
Build M the multiplication matrix of x_n
Pick V_0 ∈ K^D at random
for k from 0 to 2D − 2 do
    V_{k+1} ← M^t V_k
end for
g_n ← Berlekamp–Massey(V_{0,1}, ..., V_{2D−1,1})
w_n ← squarefree(g_n)
if deg g_n = D then
    for i from 1 to n − 1 do
        Solve the Hankel system to determine f_i(x_n)
        u_i ← f_i mod w_n
    end for
else
    for i from 1 to n − 1 do
        Compute u_i as in Equation (3).
    end for
end if
return {g_n, w_n, x_{n−1} + u_{n−1}(x_n), ..., x_1 + u_1(x_1)}.
```

Note that for solving purpose, assumptions (P_1) and (P_2) can always be retrieved by adding to the input system a generic linear form depending on one more variable which will then stand as the least one.

3.3 Univariate Polynomial Solving

In this subsection, we describe the algorithm used for real root isolation in `msolve`. It takes as input $f \in \mathbb{Q}[x]$ which we assume to be squarefree since the algebraic representation output by `msolve` stands for the *radical* of the ideal generated by the input equations.

Hence, let $f \in \mathbb{Q}[x]$ be squarefree; further, we denote by $\sigma(f)$ the number of sign variations in the sequence of coefficients of f when it is encoded in the standard monomial basis. Note that when $\sigma(f) = 0$, f has no positive real root. By Descartes' rule of signs, the difference between $\sigma(f)$ and the number of positive real roots of f is an even non-negative integer which we denote by $\delta(f)$. Consequently, when $\sigma(f) = 1$, f has a single positive real root.

This can be used efficiently in a subdivision scheme, introduced by Akritas and Collins in [15], as follows. We start by computing an integer B such that all positive real roots of f lie in the interval $]0, B[$ using the bounds given in e.g. [6, Chapter 10]. Note that, up to scaling, one can assume this interval to be $]0, 1[$. Hence, the idea is to apply some transformation $\tilde{f} = (x+1)^{\deg(f)} f\left(\frac{1}{x+1}\right)$, and compute $\sigma(\tilde{f})$. If it is 0 or 1, we are done. Else, one performs recursive calls to the algorithm by splitting the interval $]0, 1[$ to $\left]0, \frac{1}{2}\right[$ and $\left]\frac{1}{2}, 1\right[$. This is done by mapping them to $]0, 1[$, applying the transformations $x \to \frac{x}{2}$ and $x \to \frac{x+1}{2}$ to $\tilde{f}$ and taking the numerator. Termination of this subdivision scheme is ensured by Vincent's theorem [39].

To get all the real roots of f it suffices to apply the transformation $x \to -x$ and call the subdivision scheme on this newly obtained polynomial. Many improvements have been brought during the past years, in particular by integrating Newton's method to accelerate the convergence of the subdivision scheme (see e.g. [35]).

4 IMPLEMENTATIONAL DETAILS

In this section, we are given $F \subset \mathcal{P}$ and we denote by I the ideal generated by F. We assume that the base field $\mathcal{K}$ is either $\mathbb{Q}$ or a prime field of characteristic $< 2^{31}$.

To tackle systems with coefficients in $\mathbb{Q}$, we use multi-modular approaches. Here, we do not discuss details on technical necessities like good or bad primes in detail, but refer to [4, 37]. Our implementations of `F4`, the linear algebra routine on which it relies and `fglm` run over prime fields with characteristic $< 2^{31}$. In the end, we obtain rational parametrizations with polynomials with coefficients in $\mathbb{Z}$. The real root isolator implemented in `msolve` is based on the big num `mpz` arithmetic of GMP [25].

4.1 Efficiency in `F4`

For an efficient implementation of `F4` we use different approaches.

(1) We use hashing tables with linear probing in order to store the exponent vectors corresponding to monomials.

(2) For testing monomial divisibility in the symbolic preprocessing step we use a divisor mask of 32-bits, if there are more than 32 variables we just recognize the first 32.

(3) In general, rows are stored in a sparse format since for most systems `F4` matrices are very sparse. For denser matrices a sparse-dense hybrid format is implemented.

(4) We use the sparsest possible rows as pivot rows when applying Gaussian Elimination.

(5) For computations modulo prime numbers $2^{30} < p < 2^{31}$ we can use CPU intrinsics to make the basic operations, additions and multiplications of `uint32_t` elements more efficient. Using `AVX2` we can store eight 32-bit (unsigned) coefficients in one 256-bit `__m256i` type. We apply four multiplications and subtractions at a time storing intermediate results in 64-bit (signed) integers. Testing if the intermediate values are negative we can add, in that instance, a square of the field characteristic to correct positive coefficients of the usual storage type. Depending on the sparsity of the matrix this approach can lower the time spent for linear algebra in `F4` by more than the half.

4.2 Probabilistic Linear Algebra

For F4 we use the Gebauer–Möller installation from [23] in order to discard useless critical pairs. Since there still might be zero reductions during the run of the algorithm we apply over finite fields an idea that was first publicly stated by Monagan and Pearce in [32] (where it is attributed to A. Steel from the magma team).

After having moved the sparsest row for each pivot into the upper pivot matrix part, we take the remaining k rows into account. These are the rows to be reduced by the upper pivot matrix (i.e. the known leading terms for G). We partition these k rows into blocks of a given size, say ℓ rows form one block. Now we take a random linear combination of these ℓ rows and reduce it w.r.t. the upper pivot matrix. If the outcome is non-zero we have found a new pivot row and add it to the upper pivot row. Then we take another random linear combination of the ℓ rows. We stop with the current block once we have either reduced ℓ linear combinations or once the first reduction to zero happens. The probability of getting zero by chance is roughly $1/p$, for p being the field characteristic. So, if p is big enough we get the correct result with a high probability. Moreover, one can increase the probability of correctness by doing more than one reduction to zero before the block is finished. Once all blocks are handled, we are done with the linear algebra part of F4. Further we call this strategy **probabilistic linear algebra**.

4.3 F4 Tracer

In order to have more efficient modular runs of F4 we can exploit already known meta data from previous runs. We **learn** from the first finite field computation modulo some given prime number p applying F4 with *exact* linear algebra: **Trace** the main steps of the algorithm, i.e. for the first round of F4

(1) store all polynomials and multiples that generate the matrix,
(2) remove from this list all polynomials that are reduced to zero; also remove all reducers that are only needed for these specific polynomials.

In the following calls of F4 for different prime numbers we **apply** the trace from the computation modulo p. For each round we just run the following two steps:
(1) Generate the matrix with the already computed polynomials using the information from the trace.
(2) Use exact linear algebra, add the new polynomials to the basis.

Remark 4. *If we use the tracer to* F4 *we cannot use the probabilistic linear algebra in the first round since then we could not detect which specific rows reduce to zero. In the application phase of the tracer it is then useless to apply the probabilistic linear algebra since the matrices are already optimal in the sense that we do not compute any zero reduction at all. If the first prime number for which we generate the tracer is a good prime number we can be sure that only a finite number of other prime numbers exist such that the Gröbner basis computed modulo these primes via applying the tracer is not correct.*

4.4 Change of orders

Recall that we apply fglm to the *generic* situation where the ideal I satisfies assumption (P_2) and that the monomial basis $B = (m_1, \ldots, m_D)$ of $\frac{\mathcal{P}}{I}$ (D is the degree of I) satisfies assumption (P_1). This is deduced from the reduced $<_{DRL}$ Gröbner basis G of I. We denote by $\mathcal{M}$ the matrix encoding the endomorphism $\varphi : \overline{f} \in \frac{\mathcal{P}}{I} \to \overline{f x_n} \in \frac{\mathcal{P}}{I}$.

The algorithm in [21] relies on computing the Krylov sequence:

$$V_i^t = V_0^t \, \mathcal{M}^i \quad \text{for } 1 \le i < 2D$$

where V_0 is a randomly chosen vector with coefficients in our base field $\mathcal{K}$ (which is prime of characteristic $< 2^{31}$ in our context).

In [21], Faugère and Mou note that, under assumption (P_1), the matrix $\mathcal{M}$ can be read on the $<_{DRL}$ Gröbner basis of I as follows.

(1) if $\varphi(m_i) = m_j \in B$ then the ith column of $\mathcal{M}$ is the vector whose entries are all 0 except the jth which is 1;
(2) if $\varphi(m_i)$ is the lead monomial of the jth element g_j of G then the ith column of $\mathcal{M}$ is the vector of coefficients of the tail of $-g_j$ which is $\mathrm{lt}(g_j) - g_j$.

In the end, observe that the transpose of $\mathcal{M}$ enjoys a structure of generalized companion matrix with "trivial" blocks (corresponding to case (1)) and "dense" lines (corresponding to case (2)) which leads to see this matrix as a "sparse" one.

In [21], the authors analyze the sparsity of $\mathcal{M}$ under some genericity assumptions. In [27], the authors develop block Krylov techniques to accelerate these algorithms in particular through parallelism and make clearer how to apply them in the situation where I is not radical. The implementation developed there is based on the eigen library for sparse matrix multiplication. In our implementation, we treat $\mathcal{M}$, not as a general sparse matrix but as a generalized companion matrix. We encode the transpose of $\mathcal{M}$ as follows:

- we store the position of "trivial" lines and, for these lines, the position of the '1' in these "trivial" lines;
- we store the position of "dense" lines and an array for the list of coefficients.

With such an encoding, computing the V_i's simply boils down to multiplying the "dense" rows of $\mathcal{M}$ with a subvector of V_{i-1} and copying entries of V_{i-1} to the appropriate coordinates of V_i.

This reduction to dense matrix vector multiplication is efficient if most of the "dense" lines are indeed dense which is the case in most of the examples. It also allows us to use in a straightforward way AVX2 intrinsics for computing scalar products of vectors with coefficients in finite fields. As explained in Subsection 4.1, we can then perform four multiplications of the entries of our vectors by storing them in a __m256i register. To delay as much as possible reductions by the prime number defining our base field, we accumulate the highest and lowest 32 bits in separate accumulators. Since we are dealing with dense vectors, this approach allows us to obtain a speed-up close to 3.

Verifying the Parametrizations. If $\deg g_n = D$, then the returned Gröbner basis is the reduced one of I for $<_{LEX}$. Otherwise, if $\sqrt{I}$ satisfies assumption (P_2), the goal is to return a Gröbner basis of this ideal. We describe now how we implemented a new procedure deciding if $\sqrt{I}$ satisfies (P_2).

In Section 3.2, we computed a polynomial $p_i = x_i + u_i(x_n)$ as the parametrization of x_i in $\sqrt{I}$. We now compute a second parametrization q_i for x_i and compare them. Since computing the sequence terms is actually the bottleneck of this variant of the fglm algorithm, the goal is to use the sequence terms at hand.

Let us notice that since $(V_k)_{k\geq 0}$ satisfies the relation given by g_n, it is hopeless to just shift the sequence terms by increasing k. Using this recurrence relation, we can rewrite the computations w.r.t. the first sequence terms making them yield $q_i = p_i$, whether $\sqrt{I}$ satisfies (P_2) or not. Therefore, the idea is to shift the sequence terms in *another direction*. Let us assume that 1 (resp. x_i, resp. x_i^2) is the first (resp. jth, resp. j'th) monomial in B and let pick $\lambda \in \mathcal{K}$ at random. Replacing all instances of $V_{k,1}$ by $V_{k,j} + \lambda V_{k,1}$ and all those of $V_{k,j}$ by $V_{k,j'} + \lambda V_{k,j}$ makes us compute a parametrization $q_i = x_i + \tilde{f}_i(x_n)$ of the radical of the colon ideal $I : (x_i + \lambda)$, see [8, Th. 3.1]. If $\mathcal{K}$ is large enough, then $I : (x_i + \lambda) = I$ and both ideals share the same radical. Now, if $\sqrt{I}$ satisfies assumption (P_2), then so does $\sqrt{I : (x_i + \lambda)}$. Otherwise, q_i actually depends on λ and must be different from p_i, the computed parametrization of x_i for $\sqrt{I}$. Thus, we know that $\sqrt{I}$ does not satisfy assumption (P_2).

4.5 Multi-modular Approach

When $\mathcal{P} = \mathbb{Q}[x_1, \ldots, x_n]$, we have implemented efficient multi-modular algorithms.

One starts by picking randomly a prime number p_0 in the interval $]2^{30}, 2^{31}[$ and next *(i)* run the F4 tracer on the modular image of our input system in $\frac{\mathbb{Z}}{p_0\mathbb{Z}}[x_1, \ldots, x_n]$, *(ii)* run fglm on the computed Gröbner basis and normalize the obtained Gröbner basis for $<_{\text{LEX}}$ (which is in Shape position by assumption) to obtain a rational parametrization. This process is repeated for several primes, applying the tracer we learnt from p_0 until one can perform rational reconstruction (through Chinese remainder lifting) to obtain a solution over $\mathcal{P}$ whose modular image by reduction to some prime p coincides with the output of step *(ii)* when running the computation over $\frac{\mathbb{Z}}{p\mathbb{Z}}[x_1, \ldots, x_n]$. For Chinese remainder lifting and rational reconstruction, we use functions from FLINT [26] (which we have slightly adapted to our context).

Note that in step *(i)*, one can replace the F4 tracer with F4 based on probabilistic linear algebra. Note also that all computations modulo prime numbers are *independent* of each other.

This multi-modular approach is probabilistic: only for homogeneous systems we can apply a final check (over $\mathbb{Q}$) if the computed Gröbner basis is correct. Other than that we get the correct result if the Gröbner basis computed modulo the first chosen prime p_0 coincides with the image modulo p_0 of the Gröbner basis (over the rationals) of the input system. This happens with high probability and the number of such bad primes is finite (see e.g. [9, 37]).

One choice in the current design of msolve, which is inspired by the last release of FGb, is that the multi-modular process is implemented *globally*, i.e. we do not lift the intermediate reduced $<_{\text{DRL}}$ Gröbner basis over $\mathbb{Q}$.

4.6 Univariate real root isolation

Our implementation uses tricks which were previously introduced by Hanrot et al. in https://members.loria.fr/PZimmermann/software/ to implement [34] and also used in the SLV library [38]. These consist in observing that we only need the two basic operations: *(i)* shifting $x \to x + 1$ in the considered polynomial and *(ii)* scaling the coefficients by the transformation $x \to 2^k x$ for $k \in \mathbb{Z}$ which can be handled by specific GMP mpz_ shift operators [25]. The single innovation in msolve is motivated by the large bit sizes of the coefficients (several tens of thousands) and the large degrees (several thousands) of the polynomials output by msolve.

Firstly, observe that one needs to count the number of sign variations of the polynomial obtained after a combination of *(i)* and *(ii)*. In our context the bit size of the coefficients is way larger than the degree of the considered polynomial. Hence, taking appropriate dyadic approximations of these coefficients is sufficient to decide the sign (unless some unexpected cancellations occur). Note that computing such dyadic approximations is free using GMP.

Secondly, to tackle large degrees, we revisit asymptotically fast algorithms for Taylor shift (see [22]) (which we combine with the above dyadic approximation technique) and implement them carefully using the FFT-based multiplication of FLINT for univariate polynomials with integer coefficients. This is a major difference with other implementations because of the (wrong) belief that asymptotically fast algorithms are useless in this context (see [28, Section 3.1]). This allows us to obtain a univariate solver which outperforms the state of the art on examples coming from our computations (usually extracted from applications of polynomial system solving). The cross-over point of our asymptotically fast implementation of the Taylor shift against the classical implementations used in current real root solvers is around degree 512.

Similarly, we implement the quadratic interval real root refinement described in [1] for better practical efficiency which improves upon the naive one implemented in SLV.

5 EXPERIMENTAL RESULTS

We compare msolve with two other computer algebra systems:

- magma -v2.23-6 [10]: using the command Variety().
- maple -v2019 [31]: using the command PolynomialSystem() from the module SolveTools with option engine=groebner.

All compared implementations use Faugère's F4 algorithm and variants of the fglm algorithm and then solve univariate polynomials.

All chosen systems are zero-dimensional with rational coefficients. All computations are done sequentially. Table 1 states various, partly well-known benchmarks, which differ in their specific hardness, like reduction process, pair handling, sparsity of multiplication matrices, etc. Table 2 is dedicated to critical points computations, CP(d,nv,np) describes critical points for a system of np polynomials in nv variables of degree d.

For each system we give its degree and if it is radical (all but one are radical). For msolve we give specific timing information, also on the single modular computations: We apply msolve with the tracer option, giving also the timings for the first modular computation learning and generating the tracer (F4 (learn)) and the timings for the further modular computations applying only the tracer (F4 (apply)). We also use msolve with independent modular computations, applying the probabilistic linear algebra in each modular F4 (F4 (prob.)) In any case, we apply the same fglm implementation. Furthermore, we state the number of primes needed by msolve to solve over $\mathbb{Q}$. For maple and magma we just give the overall timings. Symbol '—' means that the computation was stopped after waiting more than 10 times the runtime of msolve. For all systems, the bottleneck has been the computation of either

Examples	System data		msolve single modular computation				msolve overall			maple single modular		Others overall	
	degree	radical	F4 (prob.)	F4 (learn)	F4 (apply)	fglm	# primes	trace	independent	F4	fglm	maple	magma
Katsura-9	256	yes	0.06	0.17	0.03	0.03	83	4.89	7.49	0.10	0.04	104	2,522
Katsura-10	512	yes	0.24	0.81	0.09	0.11	188	43.7	70.5	0.36	0.15	1,278	82,540
Katsura-11	1,024	yes	1.34	6.26	0.45	0.49	388	424	814	1.82	0.74	7,812	–
Katsura-12	2,048	yes	8.61	56.1	3.10	3.96	835	6,262	11,215	8.50	5.40	120,804	–
Katsura-13	4,096	yes	52.8	425	18.9	30.6	1,772	89,390	148,372	60.9	35.7	–	–
Katsura-14	8,192	yes	318	3,336	128	210	3,847	1,308,602	2,007,170	393	271	–	–
Eco-10	256	yes	0.10	0.28	0.05	0.02	161	12.5	21.2	0.14	0.03	26.3	6,520
Eco-11	512	yes	0.39	1.21	0.17	0.07	327	90.3	161	0.56	0.12	312	214,770
Eco-12	1,024	yes	2.25	11,619	1.07	0.34	530	877	1,619	2.97	0.85	4,287	–
Eco-13	2,048	yes	11.7	67.3	6.61	2.12	1,225	12,137	19,553	15.1	6.70	66,115	–
Eco-14	4,096	yes	67.1	516	34.8	25.9	2,670	167,798	254,389	104.8	69.1	–	–
Henrion-5	100	yes	0.01	0.01	0.004	0.01	83	0.71	0.83	0.01	0.01	2.7	93
Henrion-6	720	yes	0.11	0.22	0.07	0.11	612	138	157	0.17	0.16	1,470	–
Henrion-7	5,040	yes	9.55	27.5	6.51	20.46	4,243	117,803	127,456	12.8	27.1	–	–
Noon-7	2,173	yes	1.66	5.3	0.93	1.95	1,305	4,039	5,045	1.97	3.13	432	–
Noon-8	6,545	yes	26.6	153	17.5	72.3	6,462	598,647	640,177	32.4	76.2	5,997	–
Phuoc-1	1,102	no	4.01	4.65	3.42	2.59	753	4,467	5,056	4.60	5.91	–	–

Table 1: Benchmark timings given in seconds (if not otherwise stated)

a Lexicographical Gröbner basis in shape position or a rational parametrization of the solution set.

First thing to note is that magma is in all instances slower than msolve or maple. Although, for some examples, magma's modular F4 computation is even a bit faster than the other two, magma's bottleneck is both a not optimized fglm combined with the fact that magma seems to lift a lexicographical Gröbner basis instead of a rational parametrization (the latter one having in general coefficients of significantly smaller bit size).

For nearly all systems, msolve is faster, sometimes by an order of magnitude, than maple. We report on modular timings of maple for F4 (which is based on a probabilistic linear algebra) and fglm. It appears that msolve's modular implementations of both F4 and fglm are faster than the ones in maple (with a speed-up sometimes close to 2, sometimes less). It seems that in the multi-modular process, maple uses its probabilistic variant of F4 while msolve takes advantage of its tracer. Also, maple's documentation indicates that on some examples, an algorithm computing a so-called rational univariate representation (preserving multiplicities), instead of fglm, may be used. It is likely that on most examples we tried, fglm is not used. Note also that maple lifts a whole DRL Gröbner basis over $\mathbb{Q}$ while msolve avoids this step. Furthermore, our tracer shares some similarities with [33].

There are, of course, few examples, where msolve is not competitive. In particular, for some systems, msolve may need to introduce a generic linear form as previously explained while a rational parametrization can be obtained without it (but up to computing normal forms). Also some systems admit a triangular representation, and/or can be split. It seems that maple can detect and sometimes take advantage of such situations. This is typically the case for the Noon-n examples.

As for the univariate solver in msolve, we compare with maple[1] and tdescartes (non-open source) and SLV (open source). We use the standard benchmarks provided in Table 1, the solving process leads to polynomials which do not have clusters of real roots. The benefit of implementing asymptotically fast algorithms for real root solvers is now obvious: msolve's runtimes outperforms its competitors' on this class of problems.

Finally, Table 4 compares memory usage of msolve against maple and magma. It illustrates the low memory usage of msolve. However, we emphasize that msolve is a specialized library while maple and magma are general purpose computer algebra systems.

Overall, msolve performs very efficiently on a wide range of input systems, using way less memory than its competitors, allowing its users to solve polynomial systems which are not tractable by maple and magma.

Acknowledgments. We thank J.-Ch. Faugère for his advices and support and for providing us the rational parametrizations of Katsura-n for $15 \leq n \leq 17$, and A. Bostan for his comments on this paper.

REFERENCES

[1] J. Abbott. Quadratic interval refinement for real roots. *ACM Communications in Computer Algebra*, 48(1/2):3–12, 2014.
[2] M. Albrecht and G. Bard. *The M4RI Library – V. 20200125.* The M4RI Team, 2021. http://m4ri.sagemath.org.
[3] M.-E. Alonso, E. Becker, M. F. Roy, and T. Wörmann. Zeros, multiplicities, and idempotents for zero-dimensional systems. In *Algorithms in Algebraic Geometry and Applications*, pages 1–15. Birkhäuser, 1996.
[4] E. A. Arnold. Modular algorithms for computing Gröbner bases. *J. Symbolic Comput.*, 35(4):403–419, 2003.
[5] P. Aubry, D. Lazard, and M. M. Maza. On the theories of triangular sets. *Journal of Symbolic Computation*, 28(1-2):105–124, 1999.
[6] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in real algebraic geometry*, volume 10 of *Algorithms and Computation in Mathematics*. Springer-Verlag, Berlin, 2nd edition, 2006.
[7] D. Bayer and M. Stillman. A criterion for detecting m-regularity. *Inventiones mathematicae*, 87(1):1–11, 1987.

[1] We use maple-v16 as it is faster than the -v2019 for real root isolation on our benchmarks

Examples	System data		msolve single modular computation				msolve overall			maple single modular		Others overall	
	degree	radical	F4 (prob.)	F4 (learn)	F4 (apply)	fglm	# primes	trace	independent	F4	fglm	maple	magma
CP(3,5,2)	288	yes	0.03	0.04	0.01	0.03	326	18.1	19.2	0.06	0.05	249	–
CP(3,6,2)	720	yes	0.22	0.59	0.12	0.16	1,042	390	450	0.31	0.22	23,440	–
CP(3,7,2)	1,728	yes	1.97	8.18	1.23	1.20	3,037	9,643	11,511	2.78	2.54	–	–
CP(3,8,2)	4,032	yes	18.5	111.5	12.2	19.6	8,211	269,766	323,838	24.6	25.3	–	–
CP(4,4,3)	576	yes	0.04	0.86	0.03	0.07	339	40.9	41.8	0.08	0.11	916	–
CP(4,5,3)	3,456	yes	3.24	8.60	2.23	4.83	2,747	21,528	23,559	4.33	9.21	–	–
CP(3,6,6)	729	yes	0.18	0.42	0.11	0.15	779	255	294	0.30	0.23	–	–
CP(4,6,6)	4,096	yes	7.70	25.6	5.44	14.09	3,476	71,472	77,941	10.2	16.71	–	–
CP(3,7,7)	2,187	yes	2.49	8.97	1.58	1.86	2,795	12,412	14,375	3.27	3.75	–	–

Table 2: Critical points timings given in seconds (if not otherwise stated)

Examples	♯ sols	msolve	maple		SLV		tdescartes	
		time	time	ratio	time	ratio	time	ratio
Katsura-10	120	3.1	4.8	1.5	3.8	1.2	20	6.5
Katsura-11	216	27	60	2.2	50.5	1.9	156	5.8
Katsura-12	326	207	656	3.2	555	2.7	2,206	10.6
Katsura-13	582	2,220	16,852	7.6	13,651	6.1	22,945	10.3
Katsura-14	900	20,149	250,094	12.4	252,183	12.5	384,566	19.1
Katsura-15	1,606	197,048	3,588,835	18.2	3,540,480	18.0	5,178,180	26.3
Katsura-16	2,543	1,849,986	–	–	–	–	–	–
Katsura-17	4,428	16,128,000	–	–	–	–	–	–

Table 3: Real root isolation timings given in seconds

Examples	msolve	maple	magma	Examples	msolve	maple	magma
Katsura-9	15	271	71	Henrion-5	11	26	23
Katsura-10	25	276	223	Henrion-6	47	171	–
Katsura-11	66	2,279	–	Henrion-7	3,428	–	–
Katsura-12	229	1,123	–	Noon-7	209	419	–
Katsura-13	1,037	–	–	Noon-8	881	1,227	–
Eco-10	27	210	213	CP(3,5,2)	17	525	–
Eco-11	82	428	354	CP(3,6,2)	55	7,885	–
Eco-12	117	1,027	–	CP(3,7,2)	312	–	–
Eco-13	318	8,654	–	CP(4,4,3)	24	635	–
Eco-14	15,748	–	–	CP(4,5,3)	2,065	–	–
Phuoc-1	176	–	–				

Table 4: Maximal memory usage given in MB

[8] J. Berthomieu, C. Eder, and M. Safey El Din. Computing colon ideals through sequences, 2021. https://www-polsys.lip6.fr/~berthomieu/colon/colon.pdf.

[9] J. Böhm, W. Decker, C. Fieker, S. Laplagne, and G. Pfister. Bad primes in computational algebraic geometry. In G.-M. Greuel, T. Koch, P. Paule, and A. Sommese, editors, *Mathematical Software – ICMS 2016*, pages 93–101, Cham, 2016. Springer International Publishing.

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

[11] R. P. Brent, F. G. Gustavson, and D. Y. Yun. Fast solution of Toeplitz systems of equations and computation of Padé approximants. *Journal of Algorithms*, 1(3):259–295, 1980.

[12] B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal.* PhD thesis, University of Innsbruck, 1965.

[13] B. Buchberger. A Criterion for Detecting Unnecessary Reductions in the Construction of Gröbner Bases. In *EUROSAM '79, An International Symposium on Symbolic and Algebraic Manipulation*, volume 72 of *Lecture Notes in Computer Science*, pages 3–21. Springer, 1979.

[14] B. Buchberger. An Algorithm for Finding the Basis Elements of the Residue Class Ring of Zero Dimensional Polynomial Ideal (English translation of [12]) . *Journal of Symbolic Computation*, 41(3-4):475–511, 2006.

[15] G. E. Collins and A. G. Akritas. Polynomial real root isolation using Descarte's rule of signs. In *Proceedings of the Third Symp. on Symb. and Alg. Comp.*, SYMSAC '76, pages 272–275, New York, NY, USA, 1976. ACM.

[16] D. A. Cox, J. Little, and D. O'Shea. *Ideals, varieties, and algorithms.* Springer, fourth edition, 2015.

[17] D. Eisenbud. *Commutative Algebra*, volume 150 of *Graduate Texts in Mathematics*. Springer-Verlag, 1995.

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

[19] J.-Ch. Faugère. FGb: A Library for Computing Gröbner Bases. In K. Fukuda, J. v. d. Hoeven, M. Joswig, and N. Takayama, editors, *Mathematical Software – ICMS 2010*, pages 84–87, Berlin, Heidelberg, 2010. Springer.

[20] J.-Ch. Faugère, P. Gianni, D. Lazard, and T. Mora. Efficient Computation of Zero-dimensional Gröbner Bases by Change of Ordering. *J. Symb. Comput.*, 16(4):329–344, 1993.

[21] J.-Ch. Faugère and C. Mou. Sparse FGLM algorithms. *Journal of Symbolic Computation*, 80:538–569, 2017.

[22] J. v. z. Gathen and J. Gerhard. Fast algorithms for Taylor shifts and certain difference equations. In *Proceedings of the 1997 Int. Symp. on Symb. and Alg. Comp.*, ISSAC '97, pages 40–47, New York, NY, USA, 1997. ACM.

[23] R. Gebauer and H. M. Möller. On an installation of Buchberger's algorithm. *Journal of Symbolic Computation*, 6(2-3):275–286, 1988.

[24] M. Giusti, G. Lecerf, and B. Salvy. A Gröbner Free Alternative for Polynomial System Solving. *Journal of Complexity*, 17(1):154–211, 2001.

[25] The. GMP group. GMP: The GNU Multiple Precision Arithmetic Library, 2021. https://gmplib.org.

[26] W. B. Hart. Fast library for number theory: An introduction. In *Proc. of the 3rd Int. Cong. on Math. Soft.*, ICMS'10, pages 88–91. Springer-Verlag, 2010. http://flintlib.org.

[27] S. G. Hyun, V. Neiger, H. Rahkooy, and É. Schost. Block-Krylov techniques in the context of sparse-FGLM algorithms. *Journal of Symbolic Computation*, 98:163–191, 2020. Special Issue on Symb. and Alg. Comp.: ISSAC 2017.

[28] A. Kobel, F. Rouillier, and M. Sagraloff. Computing real roots of real polynomials ... and now for real! In *Proceedings of the Int. Symp. on Symb. and Alg. Comp.*, ISSAC '16, pages 303–310, New York, NY, USA, 2016. ACM.
[29] C. Kollreider and B. Buchberger. An improved algorithmic construction of Gröbner-bases for polynomial ideals. *SIGSAM Bull.*, 12:27–36, 1978.
[30] L. Kronecker. Grundzüge einer arithmetischen Theorie der algebraische Grössen. *Journal für die reine und angewandte Mathematik*, 1882(92):1–122, 1882.
[31] Maplesoft. *Maple 2019 – a division of Waterloo Maple Inc., Waterloo, Ontario*, 2019.
[32] M. B. Monagan and R. Pearce. An algorithm for spliting polynomial systems based on F4. In J. Faugère, M. B. Monagan, and H. Loidl, editors, *Proceedings of the Int. Workshop PASCO@ISSAC 2017, Germany*, pages 12:1–12:5. ACM, 2017.
[33] B. Parisse and R. De Graeve. Giac/Xcas, version 1.5.0, 2018. http://www-fourier.univ-grenoble-alpes.fr/~parisse/giac.html.
[34] F. Rouillier and P. Zimmermann. Efficient isolation of polynomial's real roots. *Journal of Computational and Applied Mathematics*, 162(1):33–50, 2004.
[35] M. Sagraloff and K. Mehlhorn. Computing real roots of real polynomials. *Journal of Symbolic Computation*, 73:46–86, 2016.
[36] P.-J. Spaenlehauer. tinygb, 2021. https://gitlab.inria.fr/pspaenle/tinygb.
[37] C. Traverso. Gröbner trace algorithms. In P. M. Gianni, editor, *ISSAC'88, Rome, Italy, July 4-8, 1988, Proceedings*, volume 358 of *Lecture Notes in Computer Science*, pages 125–138. Springer, 1988.
[38] E. Tsigaridas. SLV: a software for real root isolation. *ACM Communications in Computer Algebra*, 50(3):117–120, 2016.
[39] A. Vincent. Mémoire sur la résolution des équations numériques. *Mémoires de la Société Royale des Sciences, de L'Agriculture et des Arts, de Lille*, pages 1–34, 1834.