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 International Symposium on Symbolic and Algebraic Computation (ISSAC ’21), July 18–22, 2021, Saint Petersburg, Russia. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/xxx

1 INTRODUCTION
Problem statements and motivation. Polynomial systems arise in a wide range of areas of scientific engineering such as robotics, chemistry, biology, quantum mechanics and computing sciences such as cryptography, coding theory, computer vision to cite a few.

The end user may ask various questions on the solution set: is it finite over an algebraic closure of the ground field, or if the ground field is finite, is it finite over this field. Moreover, when the solution set is not finite over complex numbers, all complex solutions or only the real ones might be of interest.

Polynomial system solving is \( \NP \)-hard, even when the ground field is finite [23, Appendix A7.2]. Moreover, the non-linearity of such systems make reliability issues topical, in particular when complete and exhaustive outputs are required, in the context of numerical algorithms.

We tackle the problem of designing a software library, for solving multivariate polynomial systems, with a focus on those which have finitely many solutions in an algebraic closure of the ground field. We rely on computer algebra methods yielding algebraic parametrizations of the solution sets. This allows us to bypass the commonly encountered issues met by numerical methods.

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 [26] and Gröbner bases which consist in computing 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 (hence with a unique remainder) 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:

\[
\begin{align*}
 w(x_n), x_{n-1} + u_{n-1}(x_n), \ldots, x_1 + u_1(x_n).
\end{align*}
\]

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

\[
\begin{align*}
 w(x_n), w'(x_n)x_{n-1} + v_{n-1}(x_n), \ldots, w'(x_n)x_1 + v_1(x_n)
\end{align*}
\]

where \( w' \) is the derivative of \( w \). Such representations go back to Kronecker and appear in many works in computer algebra (see e.g. [3, 32]) and, under the above assumptions, are computed by regular chains and geometric resolution algorithms.

Several libraries for computing Gröbner bases can be found, most of them being either tailored for crypto applications (see e.g. [2]) or are not designed for practical efficiency but more for algorithmic experimentation (see e.g. [37]). Recently, maple and magma have greatly improved, from the practical efficiency viewpoint, their Gröbner bases engines, the one in maple being based several years ago on F6b [20], which is developed by J.-Ch. Faugère.

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 $\approx 2^{52.37}$, 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 [19] (reducing Gröbner bases computations to Gaussian elimination), of a change of orders algorithm due to Faugère and Mou [22] (based on computing minimal polynomials of linear endomorphisms) and a dedicated univariate real root solver. Solving systems with rational coefficients is low compared to the one of maple which we deduce the Krull dimension and the degree of ideals and varieties.

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 dedicated to 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 our implementation.

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. More details can be found in [16]. Let $K$ be a field; we denote by $P = K[x_1, \ldots, x_n]$ the polynomial ring with base field $K$ and variables $x_1, \ldots, x_n$.

Let $<\not$ denote a monomial order on $P$. 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 $<_{\text{DRL}}$, the degree reverse lexicographical order, and $<_{\text{LEX}}$, the lexicographical order (see e.g. [16, Chap. 2, Sec. 2, Def. 3]). Given a monomial order $<\not$ we can highlight the maximal terms of elements in $P$ with respect to $<\not$:

For $f \in P \setminus \{0\}$, $\text{lt}_< (f)$ is the lead term, $\text{lm}_< (f)$ the lead monomial, and $\text{lc}_< (f)$ the lead coefficient of $f$. For any set $F \subset P$ we define the lead ideal $L_<(F) = (\text{lt}_<(f) \mid f \in F)$: for an ideal $I \subset P$, $L_<(I)$ is defined as the ideal generated by lead terms of all elements of $I$. Further, we skip the index $<\not$ when it is clear from the context.

Definition 1. A finite set $G \subset P$ is called a Gröbner basis for an ideal $I \subset P$ w.r.t. a monomial order $<\not$ 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 $\text{lm}_<(g) \mid \text{lm}_<(f)$.

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

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

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

$$\text{sp} (f, g) := \frac{\lambda}{\text{lt}_<(f)} f - \frac{\lambda}{\text{lt}_<(g)} g.$$

(2) We say that $g$ is a reducer of $f$ if for a term $t$ in $f$ there exists a term $\sigma \in P$ such that $\text{lt}_<(\sigma g) = t$. The reduction of $f$ by $g$ is then given by $f - \sum_k g_k \sigma_k$, where no term in $\sum_k g_k \sigma_k$ is reducible by $g$. For any $f \in P$, $\text{sp} (f, g)$ reduces to zero w.r.t. $G$.

Theorem 3 (Buchberger’s criterion). A finite set $G \subset P$ is called a Gröbner basis for an ideal $I \subset P$ w.r.t. a monomial order $<\not$ if for all $f, g \in G$ $\text{sp} (f, g)$ 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 which lie in $I$ 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 $f \in P$, $F \subset P$ and a Gröbner basis $G$ of $(F)$ for a monomial order $<$, one can compute a unique representative of $f$ in the quotient ring $P/G$ which is a $K$-vector space. This property of a Gröbner basis allows one to discover relations, i.e. polynomials which lie in $(F)$ and the Hilbert series/polynomial associated to $(F)$ (see [16, Chap. 10, Sec. 2]) when using the $<_{\text{DRL}}$ order, from which we deduce the Krull dimension and the degree of $(F)$. Recall that the Krull dimension of $(F)$ coincides with the dimension of its associated algebraic set $V(F)$ in $K^n$, i.e. the largest integer $d$ such that the intersection of $V(F)$ with $d$ hyperplanes in $K^n$ is finite and of maximal cardinality. This cardinality is the degree of $(F)$ when it is radical (i.e. $f \in (F)$ for some $k$ implies that $f \in (F)$). We refer to [18, Chapter 12] for more details on the equivalence of the various definitions of dimension and [18, Section 1.9] for the relations between Hilbert series and degree of ideals and varieties.

Elimination orders such as $<_{\text{LEX}}$ allow one to compute a basis for $(F) \cap K[x_1, \ldots, x_n]$ for $1 \leq i \leq n$ 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 $(x_1, \ldots, x_n)$-subspace. It turns out that Gröbner bases w.r.t.
$<\text{LEX}$ 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 $I \subseteq \mathcal{P}$ and $g \in \mathcal{P}$, the saturation of $I$
by $(g)$, 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 $\mathbb{K}^n$ is finite, the quotient ring
$\mathbb{K}[x_1, \ldots, x_n]/(F)$ 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 $(F)$: this is the number of solutions counted with multiplicities.

This property is at the foundations of algorithms based on Gröb-
ner bases for solving polynomial systems. It implies that $(F)$ has a non-zero
intersection with $\mathbb{K}[x_1]$, and in particular to compute rational parametri-
izations as they are defined in Section 1. Such algorithms are important
since computing Gröbner bases w.r.t. $<\text{LEX}$ order is usually way
more expensive than pre-computing a Gröbner basis w.r.t. $<\text{DRL}$
and then applying such a change of orders algorithm [7].

In the following, we assume we are given a finite set of polyno-
mials $F \subseteq \mathcal{P}$ such that $(F)$ is zero-dimensional, that is $F = 0$
has a finite number of solutions in $\mathbb{K}^n$. 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 $(F)$ w.r.t. $<\text{DRL}$.
Note that once we have computed $G$ we can decide whether $(F)$ is
zero-dimensional.
(2) Convert $G$ to the reduced Gröbner basis $H$ of $\sqrt{(F)}$ w.r.t. $<\text{LEX}$
and deduce a rational parametrization $\mathcal{R}$ encoding its solutions, as
in Equation (1).
(3) Apply a univariate solver to the uniquely defined univariate
polynomial $w$ in $\mathcal{R}$. 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 intro-
duced the concept of critical pairs resp. S-pairs and repeatedly
applied a univariate solver to the uniquely defined univariate
polynomial $w$ in $\mathcal{R}$. Go on by substituting variables already solved.

\begin{algorithm}
\begin{algorithmic}[1]
\Function{Faugère’s F4 algorithm}{\text{Input: } I = (f_1, \ldots, f_m) \subseteq \mathcal{P}}
\State \text{Output: } G, \text{ a Gröbner basis for } I \text{ w.r.t. a global monomial order } <$
\State $G \leftarrow \{f_1, \ldots, f_m\}$
\State $P \leftarrow \{\text{sp}\{f_i, f_j\} | 1 \leq i < j \leq m\}$
\While{$(P \neq \emptyset)$}
\State Choose subset $L \subseteq \mathcal{P}, P \leftarrow P \setminus L$
\State $L \leftarrow \text{symbolic preprocessing}(L, G)$
\State $L \leftarrow \text{linear algebra}(L)$
\For{$h \in L$ with $\text{lm}_<(h) \notin L(G)$}
\State $P \leftarrow P \cup \{(h, g) \mid g \in G\}$
\State $G \leftarrow G \cup \{h\}$
\EndFor
\EndWhile
\State \Return $G$
\EndFunction
\end{algorithmic}
\end{algorithm}

• In contrast to Buchberger’s algorithm one can choose in Line 4
several S-pairs at a time, for example, all of the same minimal degree.
These S-pairs are stored in a subset $L \subseteq \mathcal{P}$.
• In Line 5 for all terms of all the generators of the S-pairs we
search in the current intermediate Gröbner basis $G$ for possible
reducers. We add those to $L$ and again search for 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 (Line 6) and recheck
 afterwards which rows of the updated matrix give a new leading
monomial not already in $L(G)$ (Line 7).

In order to optimize the algorithm one can now apply Buch-
berger’s product and chain criteria, see [13, 31]. With these, useless
S-pairs are removed before even added to $P$ thus less zero rows
are computed during the linear algebra part of $F_4$. Still, for bigger
examples there are many zero reductions.

It is known that Buchberger-like algorithms for computing Gröb-
ner bases, as $F_4$, have a worst-case time complexity doubly-exponen-
tial in the number of solutions of the system for $<\text{DRL}$. Still, in
practice these algorithms behave in general way better.

3.2 Gröbner conversion Algorithm
In this subsection, we present the variant of the $\text{fglvm}$ algorithm [21]
due to Faugère and Mou [22] 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 $m$ on the mono-
nomial basis $B$ of $\frac{1}{m}I$ for $<\text{DRL}$, either $\text{ms}_n$ is another monomial in $B$
or it is the leading monomial of a polynomial in $G$.

Since $\frac{1}{m}I$ is a finite dimensional $\mathbb{K}$-algebra, the multiplication
by $x_n$ is a linear map whose associated matrix $M$ is called the
multiplication matrix of $x_n$. Under assumption $(P_1)$, each column
of $M$ 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 $<\text{LEX}$ is given by
$[g_{n_1}(x_n), x_{n-1} + f_{n-1}(x_n), \ldots, x_1 + f_1(x_n)]$ where $g_{n_1}$
has degree $D$ and polyno-
mials $f_1, \ldots, f_{n-1}$, the parametrizations of $x_1, \ldots, x_{n-1}$ have
degrees at most $D - 1$. Furthermore, the radical of $I$, $\mathfrak{V}I = \{h \in \mathcal{P} | \exists k \in \mathbb{N}, h^k \in I\}$, also satisfies $(P_2)$ and its reduced Gröbner basis
for \( \leq \) we set \( \{ w_n(x_n), x_n-1 + u_n-1(x_n), \ldots, x_1 + u_1(x_n) \} \), with \( w_n \) the squarefree part of \( g_n \) and \( \deg u_i < \deg w_n \) for \( 1 \leq i \leq n \).

By construction, the minimal polynomial of \( x_n \) in \( \mathbb{F}_T \) is the same as the minimal polynomial of \( M \) and is of degree at most \( D \). This polynomial is also called the eliminating polynomial of \( x_n \) and is \( g_n \). Using Wiedemann algorithm, \( g_n \) is computed by guessing the minimal recurrence relations of the table \( \{ V_{k,h} \}_{0 \leq k < 2D} \) defined by \( V_0 \in \mathbb{Q}^D \) a column-vector chosen at random, \( V_1 = V_0 M^k \) for all \( 1 \leq k < 2D \) and \( V_{k,1} \) is the first coordinate of \( V_k \). 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 \( (V_{k+h,1})_{0 \leq h, k < D} \) and vector \( (V_{k,1})_{0 \leq k < D} \). Then, \( u_i = f_i \mod w_n \).

Whenever \( i \) does not satisfy assumption \( (P_2) \), a parametrization of the solutions might still be possible. This is the case if \( \tilde{V} \) 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 [29, Algorithm 2].

Let \( d = \deg g_n < D \) and \( h = g_n \sum_{d-1}^{d} V_{k,1} x_n^{-1} - k \) \( \text{quo} \) \( x_n \).

Then,

\[
 u_i = - \left( \sum_{k=0}^{d-1} V_{k,1} x_n^{d-1-k} \text{quo} x_n^k \right) h^{-1} \mod w_n. \quad (2)
\]

This yields the following algorithm in pseudo-code.

**Algorithm 2 Sparse \( fg\text{lm} \)**

**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. \( \leq \) with \( x_1, \ldots, x_n \notin \text{lt}(G) \).

**Output:** A parametrization of the roots of \( (G) \).

1. Build \( M \) the multiplication matrix of \( x_n \).
2. Pick \( V_0 \in \mathbb{Q}^D \) at random.
3. for \( k \) from 0 to \( 2D - 2 \) do
4. \( V_{k+1} \leftarrow M^t V_k \)
5. end for
6. \( g_n \leftarrow \text{Berlekamp–Massey}(V_{0,1}, \ldots, V_{2D-1,1}) \)
7. \( w_n \leftarrow \text{squarefree}(g_n) \)
8. if \( \deg g_n = D \) then
9. for \( i \) from 1 to \( n - 1 \) do
10. Solve the Hankel system to determine \( f_i(x_n) \)
11. \( u_i \leftarrow f_i \mod w_n \)
12. end for
13. else
14. for \( i \) from 1 to \( n - 1 \) do
15. Compute \( u_i \) as in Equation (2).
16. end for
17. end if
18. return \( \{ g_n, w_n, x_n-1 + u_n-1(x_n), \ldots, x_1 + u_1(x_1) \} \).

### 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 \( \tilde{f} = \sigma(f)f \left( \frac{1}{x+1} \right) \), 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 \([0, \frac{1}{2}] \) and \([\frac{1}{2}, 1]\). This is done by mapping them to \([0, 1]\), applying the transformations \( x \rightarrow \frac{x}{2} \) and \( x \rightarrow \frac{x+1}{2} \) to \( \tilde{f} \) and taking the numerator. Termination of this subdivision scheme is ensured by Vincent’s theorem [40].

To get all the real roots of \( f \) it suffices to apply the transformation \( x \rightarrow -x \) 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. [36]).

### 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, 38]. Our implementations of \( F_4 \), the linear algebra routine on which it relies and \( fg\text{lm} \) run over prime fields with characteristic \( < 2^{31} \). In the end, we obtain rational parametrizations with polynomials with coefficients in \( Z \). The real root isolator implemented in \( msolve \) is based on the big 

4.1 Efficiency in F4

For an efficient implementation of \( F_4 \) 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 \( F_4 \) 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 \( 
\text{uint32_t} \) elements more efficient. Using \( AVX2 \) we can store eight 32-bit (unsigned) coefficients in one 256-bit \( _\text{m256i} \) 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 adjust 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 $F_4$ by more than the half.

### 4.2 Probabilistic Linear Algebra

For $F_4$ we use the Gebauer–Möller installation from [25] in order to discard useless critical pairs. Since there 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 [34] (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 $\ell$ rows form one block. Now we take a random linear combination of these $\ell$ 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 $\ell$ rows. We stop with the current block once we have either reduced $\ell$ 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 $F_4$. Further we call this strategy **probabilistic linear algebra**.

### 4.3 $F_4$ Tracer

In order to have more efficient modular runs of $F_4$ 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 $F_4$ with exact linear algebra: **Trace** the main steps of the algorithm, i.e. for the first round of $F_4$

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 $F_4$ 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 $F_4$ 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 $f_{g_{1}}$ to the *generic* situation where $I$ satisfies assumption $(P_2)$ and that the monomial basis $B = \{m_1, \ldots, m_{D}\}$ of $\mathbb{P}^{\sqrt{\ell}}$ ($D$ is the degree of $I$) satisfies assumption $(P_1)$. This is deduced from the reduced $<_{DRL}$ Gröbner basis $G$ of $I$. We denote $M$ the matrix encoding the endomorphism $\varphi : \mathbb{P}^{\sqrt{\ell}} \rightarrow \overline{X}_{R} \in \mathbb{P}^{\mathbb{K}}$.

The algorithm in [22] relies on computing the Krylov sequence:

$$V_{i} = V_{i-1} M^{i}$$

for $1 \leq i < 2D$ where $V_0$ is a randomly chosen vector with coefficients in our base field $\mathbb{K}$ (which is prime of characteristic $< 2^{31}$ in our context).

In [22], Faugère and Mou note that, under assumption $(P_1)$, the matrix $M$ can be read on the $<_{DRL}$ Gröbner basis of $I$ as follows.

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

In the end, observe that the transpose of $M$ 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 [22], the authors analyze the sparsity of $M$ under some genericity assumptions. In [29], 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 $M$, not as a general sparse matrix but as a generalized companion matrix. We encode the transpose of $M$ 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 $M$ 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 __m256 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 $\deg g_{1} = D$, then the returned Gröbner basis is the reduced one of $I$ for $<_{LEX}$. Otherwise, if $\sqrt{\ell}$ 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 $\sqrt{\ell}$ satisfies $(P_2)$. We start by explaining the issue.

Assume, for instance, that the leading terms of the Gröbner basis of $\sqrt{\ell}$ for $<_{LEX}$ are $x_{3}^{d}, x_{2}, x_{1}x_{3}, x_{1}^{2}$. Using [29, Algorithm 2], a
rational parametrization \(\{(w_1(x_3), x_2 + u_2(x_3), x_1 + u_1(x_3))\}\) is returned but only the first two polynomials are correct. Assuming now that the leading terms are \(x_3^2, x_2x_3, x_2^2, x_1\), then \(x_2 + u_2(x_3)\) cannot be correct. Likewise, \(x_1 + u_1(x_3)\) may be false as the correct polynomial could be \(x_1 + ax_2 + u_1(x_3)\) with \(a \neq 0\). As there is no way to distinguish these two cases based on the vectors that we consider, we need a way to verify if the parametrizations are correct or not.

To do so, we compute a second parametrization for each variable and compare it with the previous one. Since computing the sequence terms is actually the bottleneck of this variant of the \(\text{fglm}\) 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 \(\varrho_{11}\), 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 the same result, be it correct or not. Therefore, the idea is to shift the sequence terms in another direction. Assuming \(x_1\) (resp. \(x_2^2\)) is the \(j\)th (resp. \(j'\)th) monomial in \(B\), we replace all instances of \(V_{k-1}\) by \(V_{k-j}\) and all those of \(V_{j-1}\) by \(V_{j-1}\). This makes us compute the parametrization \(x_1 + f_1(x_1)\) of the radical of the colon ideal \(I = (x_1)\), see [8, Th. 3.1].

Obviously, if \(l = 1: (x_1)\), then the parametrizations should be the same. Otherwise, their parametrizations can now be different leaving us unable to compare them to ensure the correctness. This issue is solved by picking at random \(\lambda \in K\) so that \(I = (x_1 + \lambda) = I\). Then, we compute a parametrization of \(x_1\) in the radical of \(I = (x_1 + \lambda)\) by considering instead the sequence terms \(V_{k,j} = V_{k,j} + \lambda V_{k,1}\) on the one hand and \(\tilde{V}_{k,j} = V_{k,j} + \lambda V_{k,1}\) on the other hand.

### 4.5 Multi-modal Approach

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

One starts by picking randomly a prime number \(p_0\) in the interval \([2^{30}, 2^{31}]\) and next \((i)\) run the \(\text{F4}\) tracer on the modular image of our input system in \(\mathbb{F}_{p_0}^{\mathbb{Z}}[x_1, \ldots, x_n]\). \((ii)\) run \(\text{fglm}\) 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 \(\mathbb{F}_{p_0}^{\mathbb{Z}}[x_1, \ldots, x_n]\). For Chinese remainder lifting and rational reconstruction, we use functions from \(\text{FLINT}\) [28] (which we have slightly adapted to our context).

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

This multi-modal 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, 38]).

One choice in the current design of \texttt{msolve}, which is inspired by the last release of \(\text{FGb}\), is that the multi-modal 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 [35] and also used in the \(\text{SLV}\) library [39]. These consist in observing that we only need the two basic operations: \((i)\) shifting \(x \rightarrow x + 1\) in the considered polynomial and \((ii)\) scaling the coefficients by the transformation \(x \rightarrow 2^k x\) for \(k \in \mathbb{Z}\) which can be handled by specific GMP \texttt{mpz}\_shift operators [27]. The single innovations in \texttt{msolve} are motivated by the large bit sizes of the coefficients (several tens of thousands) and the large degrees (several thousands) of the polynomials output by \texttt{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 \texttt{GMP}.

Secondly, to tackle large degrees, we revisit asymptotically fast algorithms for Taylor shift (see [24]) (which we combine with the above dyadic approximation technique) and implement them carefully using the FFT-based multiplication of \(\text{FLINT}\) 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 [30, 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 \(\text{SLV}\).

### 5 Experimental Results

We compare \texttt{msolve} with two other computer algebra systems:

- \texttt{magma -v2.23-6 [10]: using the command \texttt{Variety()}.}
- \texttt{maple -v2019 [33]: using the command \texttt{PolynomialSystem()}}
- \texttt{from the module \texttt{SolveTools} with option engine=\texttt{gröbner}.}

All compared implementations use Faugère’s \texttt{F4} algorithm and the \texttt{fglm} algorithm and then solve univariate polynomials.

The \texttt{SINGULAR} system version 4.1-3 [17] is too slow. E.g., it solves \texttt{Eco-10} in 16,566 seconds and \texttt{Katsura-10} in 533,876 seconds.

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. \texttt{CP(d, nv, np)} describes critical points for a system of \texttt{np} polynomials in \texttt{nv} variables of degree \texttt{d}. 
For each system we give its degree and if it is 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 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.

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 a not optimized fglm.

For nearly all systems, msolve is faster, sometimes by an order of magnitude, than maple. Especially optimizing fglm on the dense parts of the multiplication matrices gives a speedup over maple, which seems to apply sparse probabilistic linear algebra (like described in Section 4.2). maple’s approach is beneficial for very few examples like Cyclic-n where even the “dense” parts of the multiplication matrix are still sparse. Moreover, the other main difference of msolve to maple and magma is the fact that msolve does a modular F4 computation, follows by a modular fglm, then does a rational reconstruction on the parametrization, maple and magma, on the other hand, first apply a multi-modular F4 algorithm, computing the reduced Gröbner basis w.r.t. <DRL> over Q, then convert, then solve. This allows msolve to use way less memory.

There are, of course, few examples, where such a strategy is not the best, one of which is also given in Table 1: Noon-n is solved way faster by maple than msolve. As for the univariate solver in msolve, we compare with maple\textsuperscript{1} and tdescl\textsuperscript{e}r\textsuperscript{c}tes (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.

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 ≤ n ≤ 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.
[3] M.-E. Alonso, E. Becker, M. F. Roy, and T. Wormann. 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, 38(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.
[8] J. Berthomieu, C. Eder, and M. Safey El Din. Computing colon ideals through sequences. 2021. https://www.poly-lisp.fr/~berthomieu/colon/colon.pdf.
[9] J. Böhmer, W. Decker, C. Fieker, S. Lablaplange, and G. Pfister. Bad primes in computational algebraic geometry. In G.-M. Greuel, T. Koch, F. Paule, and A. Sommese, editors, Mathematical Software – ICMS 2016, pages 93–101, Cham, 2016. Springer International Publishing.

\textsuperscript{1}We use maple\textsuperscript{v16} as it is faster than the -v2019 for real root isolation on our benchmarks.
ISSAC ’21, July 18–22, 2021, Saint Petersburg, Russia
Jérémy Berthomieu, Christian Eder, and Mohab Safey El Din

| Examples | System data | msolve single modular computation | maple | Others |
|----------|-------------|-----------------------------------|-------|--------|
|          | degree      | F₄ (prob.) | F₄ (learn) | F₄ (apply) | fglm | # primes | trace | independent | magma |
| CP(3,5,2) | yes         | 0.03      | 0.04       | 0.01       | 0.03 | 326      | 18.1  | 19.2        | 249   |
| CP(3,6,2) | yes         | 0.22      | 0.59       | 0.12       | 0.16 | 1,042    | 390   | 450         | 23,440|
| CP(3,7,2) | yes         | 1.78      | 1.97       | 8.18       | 1.23 | 3,037    | 9,643 | 11,511      | –     |
| CP(3,8,2) | yes         | 4.032     | 18.5       | 111.5      | 12.2 | 8,211    | 269,766 | 323,838   | –     |
| CP(4,4,3) | yes         | 0.04      | 0.06       | 0.03       | 0.07 | 393      | 40.9  | 41.8        | 916   |
| CP(4,5,3) | yes         | 3.456     | 3.24       | 8.60       | 2.23 | 2,747    | 21,528 | 23,559     | –     |
| CP(3,6,6) | yes         | 0.18      | 0.42       | 0.11       | 0.15 | 799      | 255   | 294         | –     |
| CP(4,6,6) | yes         | 4.096     | 7.70       | 25.6       | 5.44 | 3,476    | 71,472 | 77,941     | –     |
| CP(3,7,7) | yes         | 2.185     | 2.49       | 8.97       | 1.58 | 1,86     | 2,795 | 14,375      | –     |

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

| Examples | #sols | msolve | maple | SLV | tdescartes |
|----------|-------|--------|-------|-----|------------|
| 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| 252,185    | 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