An Algorithm For Splitting Polynomial Systems Based On F4

Michael Monagan
mmonagan@cecm.sfu.ca
Simon Fraser University
Burnaby, BC, Canada

Roman Pearce
rpearcea@cecm.sfu.ca
Simon Fraser University
Burnaby, BC, Canada

ABSTRACT

We present algorithms for splitting polynomial systems using Gröbner bases. For zero dimensional systems, we use FGLM to compute univariate polynomials and factor them, placing the ideal into general position if necessary. For positive dimensional systems, we successively eliminate variables using F4 and use the leading coefficients of the last variable to split the system. We also present a known optimization to reduce the cost of zero-reductions in F4, an improvement for FGLM over the rationals, and an algorithm for quickly detecting redundant ideals in a decomposition.

CCS CONCEPTS

• **Computing methodologies** → **Algebraic algorithms**;

ACM Reference format:
Michael Monagan and Roman Pearce. 2017. An Algorithm For Splitting Polynomial Systems Based On F4. In *Proceedings of , Kaiserslautern, Germany, July 2017 (Submitted to PASCO'17),* 6 pages.
https://doi.org/

1 INTRODUCTION

Gröbner bases [7] have become a practical tool for handling polynomial systems in computer algebra. In a Gröbner basis computation, pairs of polynomials (S-pairs) are chosen, the polynomials are scaled to have equal leading terms which are cancelled, and the difference is reduced by the current basis with respect to a term ordering. Many pairs reduce to zero, but if a new polynomial is found it is added to the basis and new pairs are generated with it. The resulting Gröbner basis, in which all pairs reduce to zero, can be used to test ideal membership or determine properties such as the dimension of the solutions. Further processing can split the system into irreducible components with or without multiplicities.

Early efforts at improving the efficiency of Gröbner bases focused on avoiding unnecessary pairs. Buchberger [7] presents two criteria which were refined by Gebauer and Möller in [22]. Faugère in the F4 algorithm [15] replaces the polynomial division process with sparse linear algebra. F4 constructs a matrix whose columns are indexed by the monomials appearing in the divisions. The problem of finding new polynomials among a batch of pairs is reduced to finding new elements of the row space. Still, a large number of rows reduce to zero in F4. A known improvement is to reduce random linear combinations of rows, stopping (probabilistically) after zero is obtained a sufficient number of times.

In his F5 algorithm [17], Faugère addresses the problem zero-reductions by computing Gröbner bases incrementally. Given a Gröbner basis G and a polynomial f to be added, F5 uses criteria based on signatures in the syzygy module to discard pairs whose leading monomial is reducible by G. A survey of this and related methods is given in [14].

One often wants to compute Gröbner bases for orderings that eliminate variables. Czapor [10] suggests selecting pairs with the smallest total degree. When a Gröbner basis is known, the FGLM algorithm [19] can compute a Gröbner basis for another ordering by finding linear dependencies among the monomials in increasing order. This requires a system with a finite number of solutions or practical bounds on the degrees. In the general case, the Gröbner Walk [8] may be used to convert a basis to another ordering by lifting it through the Gröbner fan along a path of adjacent orderings.

Symbolic solutions of a polynomial system can be represented by triangular sets. A lexicographical Gröbner basis is one example, but these can be expensive to compute due to their size and degrees [11]. Other algorithms, such as [3] compute triangular sets directly, splitting the system when a zero divisor is found, or in the case of [24] when a polynomial factors. When a system can be split, its representation as the intersection of components can be much more compact [16].

Faugère mentions another algorithm F7 in [16] that splits polynomial systems while computing Gröbner bases. We do not know the details of this algorithm, but one idea that could be relevant is the following. In an incremental algorithm when we are adding a new polynomial f to a Gröbner basis G, every basis element and matrix row is either zero modulo G or a polynomial multiple of f modulo G. If a row reduces to zero, we have $g \cdot f \equiv 0 \mod G$, and G can be split into $G + g$ and $G : g^\infty$. Other incremental algorithms simultaneously compute a Gröbner basis for $G + f$ and $G : f$ [21], so it seems reasonable to investigate incremental Gröbner basis algorithms that split systems.

This paper presents several routines which can be used to factor polynomial systems. We describe our implementations of F4 and FGLM which compute Gröbner bases in grevlex, elimination, and lex order over $\mathbb{Z}_p$ and $\mathbb{Q}$, and a routine to factor zero dimensional systems based on Gianni, Trager, and Zacharias [23]. For positive dimensional systems we eliminate variables one by one using F4, which in some cases produces generators that factor. In general we use extension and contraction with extra splitting where possible.

We compare our software libmgb [25] with Magma v2.22-5 [5] on a Core i5 4570 at 3.20 GHz running 64-bit Linux. Computations modulo p use the largest machine prime available, which is $2^{31} - 1$ for libmgb and 11863279 for Magma. The benchmark systems are available on the second author's webpage.

Submitted to PASCO'17, July 2017, Kaiserslautern, Germany

ACM ISBN .
https://doi.org/

2 THE F4 ALGORITHM

The F4 algorithm [15] selects batches of pairs of the form $S(g_i, g_j) = m_i g_i - m_j g_j$. The polynomials $m_i g_i$ and $m_j g_j$ are inserted into a matrix as rows, the columns of this matrix corresponding to monomials in descending order. The usual strategy is to select all pairs of least total degree in each step. The trailing monomials are tested for divisibility, and if they are reducible, multiples of basis elements $m_k g_k$ are inserted into the matrix to cancel them off. Faugère calls this step symbolic preprocessing. Gaussian elimination is applied to find new rows with smaller leading terms. These polynomials are added to the basis, and new pairs are generated with them.

Our C implementation of F4 runs over $\mathbb{Z}_p$ [25]. We first move the sparsest row for each leading term into a matrix of pivots. The remaining n rows to be reduced are sorted and partitioned into sets of size $\sqrt{n}$. For each set, we construct random linear combinations of rows and reduce them, inserting new rows into the pivot matrix as they are found. The probability of obtaining zero by chance is approximately $1/p$, so if zero is obtained k times in a row we discard the set of rows with a probability of error of $1/p^k$. Once all of the sets have been processed, we extract the new pivots and perform back substitution to obtain a unique reduced result.

The probabilistic strategy was first developed by Allan Steel [29] and we figured it out independently from studying Magma timings. It works best on dense systems with many zero reductions when n and p are large. Below is a table comparing our probabilistic F4 for $k = 1, 2, 4$ to the exact routine and Magma. The times are for one core but we show the parallel speedup on 4 cores for $k = 1$, where sets of rows are processed in parallel.

The block size of $\sqrt{n}$ was chosen experimentally. Larger blocks can reduce the cost of zero-reductions, but smaller blocks allow for more parallelism and faster discovery of pivots when blocks are processed in parallel. When running in parallel, new pivots are inserted with a non-blocking compare and swap [25]. If the compare and swap fails we reduce the row further and try again. Our attempts to parallelize other parts of the algorithm, e.g. symbolic preprocessing, have so far been met with uneven success.

Table 1: Probabilistic F4 modulo p with k check primes.

	speedup	k=1	k=2	k=4	exact	Magma
bayes148	1.24x	18.939	19.792	21.276	25.035	16.130
cyclic8	1.67x	1.077	1.200	1.446	2.577	1.134
cyclic9	2.05x	74.172	81.075	94.571	308.901	54.621
katsura10	1.75x	0.839	1.024	1.383	4.079	0.750
katsura12	2.08x	29.505	35.094	47.387	264.337	22.602
gametwo7	2.08x	8.758	9.731	11.725	28.337	6.354
jason210	1.32x	3.436	3.667	4.093	3.064	14.054
noon9	1.47x	8.888	9.932	11.390	8.965	29.754
reimer8	1.32x	19.908	22.680	28.075	39.208	18.775

Over the rationals we use Chinese remaindering and rational reconstruction. As basis elements are recovered, they are added to the input for the modular algorithm. This substantially improves performance on systems where degree drops occur, e.g. cyclic-n, as subsequent images are computed faster. For example, for cyclic-8 over $\mathbb{Q}$ the sequential time is reduced from 52.36 to 16.67 seconds, and the parallel time is reduced from 32.51 to 11.53 seconds.

3 THE FGLM ALGORITHM

We implemented FGLM [19] on top of the linear algebra routines used in F4. This algorithm takes a Gröbner basis G for a zero-dimensional ideal I as input and computes a lexicographic basis L. It first generates the monomials in the quotient ring $k[x_1, \ldots, x_n]/I$ using the input basis. These monomials are multiplied by each variable and symbolic preprocessing is used to obtain a matrix A to reduce polynomials to normal form using row reductions.

The matrix A can reduce the product of any variable and normal form in $k[x_1, \ldots, x_n]/I$. This task is usually performed using multiplication matrices for each variable [20]. Compared to those matrices, the matrix A has extra rows and columns and is not as dense. We tried using back substitution to obtain the rows of the multiplication matrices in A, but it was rarely worthwhile.

The FGLM algorithm counts up through the monomials in the target ordering starting from 1. For each monomial m_i it checks if the normal form of $m_i \mod G$ is a linear combination of the previous normal forms. When a linear dependency is found, the linear combination of monomials is a new polynomial to be added to the lex basis L. Subsequent monomials considered by the algorithm should not be reducible by any element of L.

Our first implementation of FGLM uses three auxiliary arrays: m is an array of target monomials, R is an array of normal forms, and B is a triangular matrix that is used to find dependencies. To compute the normal form of a new monomial m_i, there are two cases. If m_i is not reducible by G then it is already in normal form, e.g. 1. Otherwise, we write m_i as the product of a previous monomial m_j for $j < i$ and some variable x_k (usually the last variable) and compute the normal form $R[m_i]$ by reducing $R[m_j] \cdot x_k$ using the matrix A. We copy $R[m_i]$ and attach an additional column, $1 \cdot e_i$, which is used to track linear dependencies.

That is, we append rows of an identity matrix to the normal forms before the linear algebra reduction. The augmented normal form is reduced by the matrix of pivots B, and we obtain either a linear dependency in the columns $\{e_j : j \le i\}$ or a new pivot that is added to B with updated dependency information. The algorithm stops when no more monomials can be generated that are not reducible by L.

Below is a table of timings for FGLM modulo p. We show the time spent computing normal forms and the time to detect linear dependencies, and the total time. The cost to generate A is small, and is equal to the total time minus the others. We include Magma timings for comparison. Magma has a fast dense algorithm for the shape lemma case. The timings are competitive with the version of sparse FGLM presented in [20], but not with the version presented by Faugère, Gaudry, and Renault in [18].

Table 2: First Version of FGLM algorithm modulo p.

	sols	n.form	linalg	total	Magma
cyclic7	924	0.006	0.011	0.025	0.460
cyclic10	34940	63.364	288.541	361.810	> 10000
katsura10	1024	0.381	0.634	1.109	0.620
katsura12	4096	21.388	38.697	61.777	50.459
noon9	19665	78.039	560.858	641.541	> 10000
reimer7	2880	2.886	2.329	5.450	220.330
reimer8	14400	291.750	152.987	451.603	> 10000

We considered parallelizing this code, but the computation of each normal form requires the previous ones, so the maximum speedup we could obtain would be about 3x. Instead, we sought to address an inefficiency pointed out by a referee.

In our second version of the algorithm, we again construct the matrix A and augment it with extra columns to keep track of the dependencies. The difference is that rows are added directly to the matrix A, so that normal forms and depedencies are computed in a single row reduction. When the next monomial m_i is to be reduced, we take the previous row that was added to A after being reduced, and scale it to obtain a row containing m_i in the dependencies. This works provided the ideal is in shape position.

The table below shows the time to construct A and run this algorithm, compared to our previous code. There is a significant improvement for both sparse and dense systems. We are currently working to adapt this routine to the non shape position case. We did not attempt to parallelize the algorithm because each iteration of the main loop uses the result of the previous one, and the time taken is at most a few milliseconds.

Table 3: FGLM algorithm modulo p in shape position.

	sols	setup A	linalg	previous	speedup
eco12	1024	0.029	0.482	2.637	5.16x
eco14	4096	0.798	33.412	182.422	5.33x
katsura10	1024	0.025	0.276	1.109	3.68x
katsura12	4096	0.505	16.633	61.777	3.60x

Over $\mathbb{Q}$, the coefficients of a lex Gröbner basis can be extremely large so instead of applying Chinese remaindering directly, we try to recover a much smaller triangular set [11]. This idea was noted in [2] and has been used in various systems.

When the first image mod p is computed, we identify the univariate polynomial f in the lowest variable. If $\gcd(f, f') = 1 \mod p$, we multiply the remaining polynomials by f' and reduce modulo f. We do this for each image, and reconstruct the basis in that form.

Dahan and Schost [11] show this form is an order of magnitude smaller than a lex Gröbner basis. To recover the lex basis we take the univariate polynomial f and compute the inverse $1/f' \mod f$ over the rationals. The remaining polynomials are multiplied by the inverse and reduced modulo f.

In Table 4 below, we report the times and number of images modulo p required to recover the lex basis using this approach versus direct recovery. On some systems, e.g. cyclic-7, the triangular set method destroys sparsity, which we can see from the increased times for Chinese remaindering and rational reconstruction. For comparison we include timings for FGLM in Magma and Singular and a rational univariate representation [27] computed in Maple.

Making this algorithm effective required several improvements to Maple. We implemented multivariate division by multiple polynomials in C using a heap [26], and used this in the sprem and Rem commands to compute remainders over $\mathbb{Q}$ and $\mathbb{Z}_p$. We implemented rational reconstruction in the Maple kernel in C to support FGLM with an efficient fraction free algorithm for polynomials. We changed the extended Euclidean algorithm to use a sparse primitive remainder sequence to compute the inverses, after we ran into a problem on the cyclic-7 system. We needed to compute the inverse

Table 4: FGLM over $\mathbb{Q}$: triangular set versus direct recovery.

triangular	#p	images	chrem	ratrec	misc	total
cyclic7	25	0.616	0.372	0.449	0.757	2.194
katsura7	22	0.090	0.020	0.053	20.396	20.559
reimer7	7	37.957	0.028	0.003	0.529	38.517
schwarz11	231	26.887	1.011	1.178	11.786	40.862
direct	#p	images	chrem	ratrec	misc	total
cyclic7	36	0.868	0.099	0.104	0.124	1.195
katsura7	2325	9.587	41.730	32.911	2.020	86.248
reimer7	156	848.083	0.559	0.215	2.959	851.816
schwarz11	429	50.330	1.377	5.856	1.046	58.609

other	Magma 2.22-5	Maple RUR	Singular 4.1
cyclic7	3.899	2797.951	64.630
katsura7	33.630	2.329	133.520
reimer7	357.759	> 10000	–
schwarz11	9.400	> 10000	5047.530

of $f'(x)$ in the quotient field $\mathbb{Q}[x]/(f)$ where f is the univariate polynomial for cyclic-7 (see Appendix A). The Maple command

```
> g := gcdex(diff(f,x),f,x,'s');
```

solves $sf' + tf = g$ for $s, t, g \in \mathbb{Q}[x]$ where $g = \gcd(f', f) = 1$. Thus the polynomial $s(x)$ is the desired inverse. Maple is using Collins' reduced pseudo remainder sequence (PRS), see [9]. For this input it does not terminate because of an exponential coefficient blowup. That this can happen was first pointed out by Brown and Traub in [6]. The authors write:

> *In a normal reduced PRS, the coefficient growth is essentially linear. In an abnormal PRS, the growth can be exponential, but of course not as badly as in the corresponding Euclidean PRS.*

Switching to use Brown and Traub's subresultant PRS eliminates the exponential blowup. The largest integer coefficient in the subresultant PRS sequence has 81,353 digits which is still very big. We noticed that if we use a primitive PRS instead, and we use sparse pseudo-division, the largest coefficient in the PRS is only 1,352 digits and the computing time is reduced from 0.751 second to 0.017 seconds. For clarity, we include details of our sparse primitive PRS in Appendix B.

4 ZERO DIMENSIONAL SYSTEMS

To split zero-dimensional ideals we use the algorithm of Gianni, Trager, and Zacharias [23]. As suggested by Decker, Greuel, and Pfister [13] we factor univariate polynomials in each variable before placing the ideal into general position, which tends to destroy sparsity. In the last section we found lex Gröbner bases unweildy, so we will try to avoid their construction.

Let x be the chosen variable. To split the ideal we use FGLM to find the univariate generator of $I \cap k[x]$ and factor it. Let $f = f_1^{e_1} f_2^{e_2} \dots f_k^{e_k}$ be the factorization. For a primary decomposition, we would add each $f_i^{e_i}$ to the ideal to obtain the components, and for prime decomposition of the radical we would add each f_i.

Our implementation of FGLM can optionally return the normal forms of the target monomials $\{1, x, x^2, \dots\}$. This allows us to rewrite each factor $f_i^{e_i}$ as a linear combination of normal forms to obtain equivalent polynomials of low degree. Our F4 code accepts

an initial Gröbner basis as input to allow for faster recomputation of a grevlex basis for each component.

Note that if the ideal is found to be in general position with respect to x, that is $\deg(f)$ is equal to the number of solutions over the algebraic closure, then the components we obtain are primary (or prime) and the algorithm can stop [23]. Otherwise, as a final step we introduce a new variable z and a random linear form $f = z + c_1x_1 + \cdots + c_nx_n$ and factor the univariate polynomial in z, as in [4]. Note that if G is a Gröbner basis for $I \subset k[x_1, \ldots, x_n]$ in grevlex order then $G \cup \{f\}$ is a Gröbner basis in $k[z, x_1, \ldots, x_n]$ in grevlex order. That is, by placing z first we can avoid recomputing a grevlex basis to run FGLM. The table below shows the time to compute a prime decomposition of the radical modulo p.

Table 5: Zero Dimensional Prime Decomposition modulo p.

	sols	#P	F4	FGLM	factor	1 core	4 cores
cyclic7	924	588	0.143	0.010	0.013	3.260	1.920
eco12	1024	120	2.201	2.391	5.869	177.600	64.520
gametwo7	1854	4	9.359	7.623	22.231	61.160	48.090
katsura10	1024	14	1.073	1.136	5.239	20.770	12.060
katsura11	2048	13	5.685	8.103	29.011	121.200	72.040
reimer7	2880	264	0.806	0.205	0.012	6.520	3.640
reimer8	14400	480	21.282	4.027	0.026	231.600	99.010
schwarz11	2048	2048	0.372	0.090	0.468	271.200	119.290

We report the number of solutions, the number of components (in column #P), and the times for the first calls of F4, FGLM, and factorization. We see that on some systems, e.g. eco-12, recomputing grevlex bases was expensive. This prevented us from computing decompositions for cyclic-10 and noon-9 in grevlex order.

One can see that a significant amount of time is spent processing systems after the initial split. For the parallel times, we used Maple's Grid package to spawn extra processes for that purpose.

Over $\mathbb{Q}$, the content of the normal forms of $\{1, x, x^2, \ldots\}$ can be very large. To overcome this problem, we can construct the normal forms of the factors modulo p for several primes, make the images monic with respect to their leading terms, and apply Chinese remaindering and rational reconstruction.

5 POSITIVE DIMENSIONAL SYSTEMS

For positive dimensional systems, we again face the problem that a lex Gröbner basis is too expensive to compute. E.g. for cyclic-8 modulo p a grevlex basis takes about a second, whereas a lex basis takes over 100 seconds with F4 or over 800 seconds with the Gröbner walk in Magma. The problem is that all of the polynomials in the lex basis have high degrees in the lowest variables. It is much easier to eliminate variables one by one using F4, that is, compute $I_2 = I \cap k[x_2, \ldots, x_n]$ then $I_3 = I_2 \cap k[x_3, \ldots, x_n]$ and so on. For cyclic-8 this takes less than two seconds, and we obtain generators that factor in the end.

Let us review some splitting tools from Decker et al [13]. Let I be an ideal of $R = k[x_1, \ldots, x_n]$, then:

- If $f \in R$ and $I : f^s = I : f^\infty$ then $I = (I : f^s) \cap (I + f^s)$.
- In particular $\sqrt{I} = \sqrt{I : f} \cap \sqrt{I + f}$ for any $f \in R$.
- If $f \cdot g \in I$ and $\langle f, g \rangle = R$ then $I = (I + f) \cap (I + g)$.
- In particular, for any $f \cdot g \in I$, $\sqrt{I} = \sqrt{I + f} \cap \sqrt{I + g}$.

Let c be the co-dimension of the ideal and reorder the variables so that a maximal independent set $\{x_{c+1}, \ldots, x_n\}$ comes last. Then $I_c = I \cap k[x_c, \ldots, x_n]$ is the last non-empty ideal encountered when eliminating variables one by one.

If the generators of I_c factor then we can apply the last splitting tool, but if the factors have high degrees then adding them to the original ideal can produce blowup in F4. We can reduce the degrees of the factors by computing normal forms in reverse: that is, compute the normal form with respect to I_{c-1}, then I_{c-2}, etc.

Alternatively, if there is only one large factor we can get its component by saturating the ideal with respect to the small factors. This is the case, e.g. for cyclic-9, where one factor yields a large zero-dimensional component.

The example of cyclic-9 is worth discussing because the approach is unreasonably effective. Computing a grevlex Gröbner basis in about a minute (modulo p) we find that the dimension of the ideal is 2. We compute $I_7 = I \cap k[x_7, x_8, x_9]$ and find 425 polynomials, all of which have a factor of $f = x_7^3x_8^3x_9^3 - 1$ and another factor in $k[x_8, x_9]$. We can quickly find f by computing a gcd. Adding f to the ideal produces a component with dimension 2 which can be factored further, while saturating by f produces a zero dimensional ideal with 5796 solutions.

Of course, we may not be so lucky as to find generators that factor in I_c. For the general case over $\mathbb{Q}$, we use extension and contraction [4, 23] to reduce the dimension of the ideal. Applied to I_c we would compute a Gröbner basis with $x_c \gg \{x_{c+1}, \ldots, x_n\}$ and take f to be the lcm of the leading coefficients in x_c. Note that this polynomial can be very large, however we can also factor it and use the splitting tools in the following.

For a primary decomposition we find s with $I : f^s = I : f^\infty$ and factor $I_c : f^s$ as a zero-dimensional ideal over $\mathbb{Q}(x_{c+1}, \ldots, x_n)$. Its components are contracted back to the original ring and we recursively factor $I_c + \langle f^s \rangle$ which has lower dimension. For prime decomposition of the radical we can take $s = 1$.

We implemented the full primary decomposition algorithm in Maple but its performance remains unsatisfactory. Over $\mathbb{Z}_p$ we can use the algorithm of [28] but we have not implemented that yet. Instead we will present a splitting algorithm that we are testing as a preprocessor for prime decomposition of the radical. We did not include lifting factors of I_c to normal forms modulo I.

Algorithm 1: Positive Dimensional Splitting

Input: A set of generators for $I \subset k[x_1, \ldots, x_n]$.
Output: Ideals $P_1, \ldots, P_k$ with $\sqrt{I} \subseteq \bigcap P_i \subseteq I$
G[0] := grevlex Gröbner basis for I
for i from 1 to n do
  G[i] := Gröbner basis for $G[i-1] \cap k[x_i, \ldots, x_n]$
    in an elimination order with $x_i \gg \{x_{i+1}, \ldots, x_n\}$.
end loop
g := gcd of the polynomials in the last $G[i] \neq \emptyset$.
F := the set of irreducible factors of g.
return $G[0] : g^\infty$ and $(G[0] \cup f)$ for each $f \in F$.

We report the time and number of components generated and the total number of solutions in zero dimensional components. The F4 algorithm is run with and without parallel linear algebra.

Table 6: Positive Dimensional Splitting over $\mathbb{Z}_p$.

	dim	#P	sols	1 core	4 cores
alea6	1	2	375	1.560	1.280
cyclic8	1	10	864	9.090	7.170
cyclic9	2	4	5796	112.780	64.010

Aggressive use of the splitting tools generates many redundant components, so we need a fast way to detect and remove redundant ideals in an intersection. Let $\{G_1, G_2, \ldots, G_k\}$ be Gröbner bases for the ideals. For each G_i make f_i a random linear combination of the polynomials in the Gröbner basis.

To test which ideals are contained within G_i (so that G_i can be removed) we compute normal forms for all $\{f_j | \ j \neq i\} \mod G_i$ simultaneously using symbolic preprocessing and matrix reduction. If an f_j reduces to zero, we test explicitly if all the elements of G_j reduce to zero mod G_i with a second simultaneous normal form computation. If so then $G_i \cap G_j = G_j$ and we can remove G_i.

We implemented this algorithm over $\mathbb{Z}_p$ and found that it runs almost instantly. It does not detect if G_i contains the intersection of the other ideals, so it does not produce a minimal decomposition, only an irredundant one. Over $\mathbb{Q}$ we can take the G_i to be images mod p and test $f_j \in G_i$ and $G_j \subseteq G_i$ over $\mathbb{Z}_p$ before checking containment of the original ideals over $\mathbb{Q}$.

Finally, we mention our interest in splitting positive dimensional systems is also motivated by problems where it is not practical to compute or use a grevlex Gröbner basis for the entire system, e.g. cyclic-10. For these problems we hoped to proceed incrementally, computing a Gröbner basis for $\langle f_1, f_2 \rangle$ and splitting, then adding f_3 to each component and splitting again, etc. Unfortunately we found that while computing the first few incremental bases is fast, eliminating variables from those bases can be very slow.

6 CONCLUSION

In this paper we presented our routines for splitting polynomial systems based on F4. We first compute a grevlex basis, and if the ideal is zero-dimensional we apply the FGLM algorithm to split the univariate polynomials, stopping after the ideal has been in shape position. If necessary, we introduce a new variable and random linear form to put the ideal into shape position. When factors are found, we rewrite them to obtain low degree polynomials that can be added to the grevlex basis efficiently.

For positive dimensional ideals, we successively eliminate variables to intersect the ideal with a subring containing a maximal independent set of variables and one other variable. Sometimes we are lucky and the generators factor. Otherwise, we use extension and contraction, splitting where possible.

Splitting positive dimensional ideals can generate many redundant components. We presented a randomized algorithm to detect these efficiently. We also presented a probabilistic improvement to F4 which was known, and our implementation of a triangular set approach to recovering lex bases over rationals using FGLM. Our implementation of FGLM is based on the linear algebra from F4, and further improvements appear to be possible.

In the future we plan to incorporate these ideas and routines into a new general purpose polynomial system solver for Maple.

7 ACKNOWLEDGEMENTS

This work was supported by NSERC of Canada and Maplesoft. We would also like to thank the referees for their helpful comments.

REFERENCES

[1] W.W. Adams and P. Loustaunau. An Introduction to Gröbner Bases. Graduate Studies in Mathematics **3**, Ameri. Math. Soc., 1994.
[2] M.-E. Alonso, E. Becker, M.-F. Roy, and T. Wörmann. Zeroes, multiplicities and idempotents for zero-dimensional systems, *proceedings of MEGA '94*, in Progress in Mathematics, **142**, 1–15, Birkhäuser, 1996.
[3] P. Aubry, D. Lazard, M. Moreno Maza. On the Theories of Triangular Sets. *J. Symb. Comp.* **28**(1-2), 105–124, 1999.
[4] T. Becker, V. Weispfenning. Gröbner Bases: A Computational Approach to Commutative Algebra. Graduate Texts in Mathematics **141**, Springer-Verlag, 1993.
[5] W. Bosma, J. Cannon, C. Playoust. The Magma algebra system. I. The user language, *J. Symb. Comput.*, **24**, 235–265, 1997.
[6] W.S. Brown, J.S. Traub. On Euclid's algorithm and the theory of subresultants *J. ACM*, **18**(4) 505–514, 1971.
[7] B. Buchberger. Gröbner-Bases: An Algorithmic Method in Polynomial Ideal Theory. *Multidimensional Systems Theory - Progress, Directions and Open Problems in Multidimensional Systems*, 184–232, Reidel, 1985.
[8] S. Collart, M. Kalkbrener, D. Mall. Converting Bases with the Gröbner Walk. *J. Symb. Comput.* **24**, 465–469, 1997.
[9] G.E. Collins, Subresultants and reduced polynomial remainder sequences. *J. ACM*, **14**(1) 128–142, 1967.
[10] S. Czapor. A heuristic selection strategy for lexicographic Gröbner bases? *Proc. ISSAC 1991*, ACM Press, 39–48, 1991.
[11] X. Dahan and E. Schost. Sharp estimates for triangular sets. *Proc. ISSAC 2004*, 103–110, 2004.
[12] K.O. Geddes, S.R. Czapor, G. Labahn, Algorithms for Computer Algebra, Kluwer Academic Publishing, 1992.
[13] W. Decker, G.M. Greuel, G. Pfister. Primary Decomposition: Algorithms and Comparisons. in: Algorithmic Algebra and Number Theory, Springer, Berlin, Heidelberg, 187–220, 1998.
[14] C. Eder, J.C. Faugère. A survey on signature-based Gröbner basis computations. *ACM Comm. in Computer Algebra*, **49**(2), 2015.
[15] J.C. Faugère. A new efficient algorithm for computing Gröbner bases (F4). *J. of Pure and Applied Algebra*, **139**, 61–88, 1999.
[16] J.C. Faugère. Finding all the solutions of Cyclic 9 using Gröbner basis techniques. *Proc. of ASCM 2001*, World Scientific, 1–12, 2001.
[17] J.C. Faugère. A new efficient algorithm for computing Gröbner basis without reduction to zero F5, *Proc. of ISSAC 2002*, ACM Press, 75–83, 2002.
[18] J.C. Faugère, P. Gaudry, L. Huot, G. Renault. Sub-Cubic change of ordering for Gröbner basis: a probabilistic approach. *Proc. of ISSAC 2014*, ACM Press, 170–177, 2014.
[19] J.C. Faugère, P. Gianni, D. Lazard, T. Mora. Efficient Computation of Zero-dimensional Gröbner Bases by Change of Ordering. *J. Symb. Comput.*, **16**(4), 329–344, 1993.
[20] J.C. Faugère, C. Mou. Sparse FGLM algorithms. *J. Symb. Comput.*, **80**(3), 538–569, 2017.
[21] S. Gao, Y. Guan, F. Volny. A New Incremental Algorithm for Computing Gröbner Bases. Proc. ISSAC 2010, ACM Press, 13–19, 2010.
[22] R. Gebauer and H.M. Möller. On an Installation of Buchberger's Algorithm. *J. Symb. Comput.*, **6** (2 and 3), 275–286, 1998.
[23] P. Gianni, B. Trager, G. Zacharias. Gröbner bases and primary decomposition of polynomial ideals. *J. Symb. Comput.* **6**, 149–167, 1988.
[24] H.G. Gräbe. Triangular systems and factorized Gröbner bases. in: Applied Algebra, Algebraic Algorithms and Error-Correcting Codes. *LNCS* **948**. Springer, Berlin, Heidelberg, 248–261, 1995.
[25] M. Monagan and R. Pearce. A Compact Parallel Implementation of F4. *Proc. of PASCO 2015*, ACM Press, 95–100, 2015.
[26] M. Monagan and R. Pearce. Fermat Benchmarks for Rational Expressions in Maple. *ACM Comm. in Computer Algebra*, **50**(4), 188–190, 2016.
[27] F. Rouillier. Solving zero-dimensional systems through the Rational Univariate Representation. *AAECC Journal*, **9**(5), 433–461, 1999.
[28] A. Steel. Conquering inseparability: Primary decomposition and multivariate factorization over algebraic function fields of positive characteristic. J. Symb. Comput. **40**(3), 1053–1075, 2005.
[29] A. Steel. Private communication, 2015.

APPENDIX A: CYCLIC 7

The cyclic n system is the system of n polynomial equations

$$\{ f_1 = 0, f_2 = 0, \ldots, f_{n-1} = 0, f_n = n \}$$

where

$$f_i = \sum_{j=1}^{n} \prod_{k=j}^{j+i-1} x_k \text{ for } 1 \le i \le n$$

with $x_{n+j} = x_j$ so that $f_2 = x_1x_2 + x_2x_3 + \cdots + x_{n-1}x_n + x_nx_1$. The univariate polynomial $f(x_1)$ for cyclic 7 which generates the ideal $\langle f_1, f_2, f_{n-1}, \ldots, f_n - n \rangle \cap \mathbb{Q}[x_1]$ is

```
f := 128*x1^203-625591465*x1^196
-1379565483492966*x1^189
-690513600129227139 30*x1^182
+503269676439575515122310*x1^175
-3586077716264197311194890 79*x1^168
-1479528408776356997786673165 08*x1^161
-1147922766589841137530437873498*x1^154
-3283047528072876955188179107557*x1^147
-5494471139223382529181506689788*x1^140
-6295323446544817875033709377526*x1^133
-5439730226339556502298385195313*x1^126
-4047134740954486045236505654710*x1^119
-2699447938384037396439601445910*x1^112
-1005987107212742027676051174393*x1^105
+1005987107212742027676051174393*x1^98
+2699447938384037396439601445910*x1^91
+4047134740954486045236505654710*x1^84
+5439730226339556502298385195313*x1^77
+6295323446544817875033709377526*x1^70
+5494471139223382529181506689788*x1^63
+3283047528072876955188179107557*x1^56
+1147922766589841137530437873498*x1^49
+1479528408776356997786673165 08*x1^42
+3586077716264197311194890 79*x1^35
-503269676439575515122310*x1^28
+690513600129227139 30*x1^21
+1379565483492966*x1^14+625591465*x1^7-128;
```

APPENDIX B: THE SPARSE PRIMITIVE PRS

Let $a, b \in \mathbb{Z}[x]$ with $\deg a \ge \deg b \ge 0$. Let q, r be the quotient and remainder of $a \div b$. Thus q, r satisfy $a = bq + r$ with $r = 0$ or $\deg r < \deg b$. Recall that the pseudo-quotient and pseudo-remainder $\tilde{q}, \tilde{r}$ of $a \div b$ satisfy

(i) $\tilde{q}, \tilde{r} \in \mathbb{Z}[x]$,
(ii) $\mu a = \tilde{q}b + \tilde{r}$ with $\tilde{r} = 0$ or $\deg \tilde{r} < \deg b$, and
(iii) $q = \tilde{q}/\mu$ and $r = \tilde{r}/\mu$.

where the multiplier $\mu = \text{lcoeff}(b)^{\delta}$ with $\delta = \max(0, \deg a - \deg b + 1)$. For details see Ch 2 of [12]. In the pseudo division algorithm, it is possible that a smaller power may be used for δ such that (i), (ii) and (iii) are still satisfied. A pseudo-division algorithm that uses the smallest δ is called sparse pseudo-division. Here is Maple code for sparse pseudo-division.

```
spdiv := proc(a,b,x)
# Input a,b in Z[x] with b <> 0
# Output mu in Z and pseudo remainder pr
# and pseudo quotient pq in Z[x]
local pr,pq,mu,db,dr,lb,lr;
  db := degree(b,x);
  lb := coeff(b,x,db);
  dr := degree(a,x);
  pr,pq,mu := a,0,1;
  while pr<>0 and dr>=db do
     lr := coeff(pr,x,dr);
     qt := lr*x^(dr-db);
     pr := expand(lb*pr-qt*b);
     pq := lb*pq+qt;
     mu := mu*lb;
     dr := degree(pr,x);
  end do;
  return(mu,pr,pq); # mu a = pq b + pr
end proc;
```

Let $a, b \in \mathbb{Z}[x]$ with $b \neq 0$. To compute a^{-1} in $\mathbb{Q}[x]/(b)$ we want to solve $sa + tb = g$ for $g, s, t \in \mathbb{Q}[x]$ where $g = \gcd(a, b)$. If $g = 1$ then $s = a^{-1}$ otherwise a is not invertible. We can use the ordinary (half) extended Euclidean algorithm (see [12] Ch 2) to do this. Instead, to avoid fractions, we use pseudo-division. The Maple code below uses the primitive PRS with sparse pseudo-division. It outputs $s_1 \in \mathbb{Z}[x]$ and $r_1 \in \mathbb{Z}$ such that $s_1/r_1 = a^{-1}$.

```
speea := proc(a,b,x)
# Input a,b in Z[x] with deg(b)>0
# Output s1 in Z[x] and r1 in Z
local r0,r1,r2,s0,s1,s2,mu,pq,g;
  r0,r1 := a,b;
  s0,s1 := 1,0;
  while r1 <> 0 and degree(r1,x) > 0 do
    mu,r2,pq := spdiv(r0,r1,x);
    s2 := expand(mu*s0-pq*s1);
    if r2 <> 0 then
      g := igcd(icontent(r2),icontent(s2));
      r0,r1 := r1,r2/g;
      s0,s1 := s1,s2/g;
    end if;
  end do;
  if r1=0 then return(FAIL) fi;
  return(s1,r1); # s1/r1 = a^(-1) mod b
end proc;
```